3/22/2022

Nagendra Mirajkar

Zensar AES Practices Front End Engineering SGO

JavaScript BOM

Browser Object Model

Table of Contents

[JavaScript BOM 2](#_Toc99030664)

[JavaScript Window 2](#_Toc99030665)

[The window object is the Global object 2](#_Toc99030666)

[The window object exposes the browser’s functionality 2](#_Toc99030667)

[**1) Window size** 2](#_Toc99030668)

[**2) Open a new window** 3](#_Toc99030669)

[**3) Resize a window** 4](#_Toc99030670)

[**4) Move a window** 4](#_Toc99030671)

[**5) Close a window** 5](#_Toc99030672)

[**6) The window.opener property** 5](#_Toc99030673)

[Summary 5](#_Toc99030674)

[JavaScript alert 5](#_Toc99030675)

[Introduction to JavaScript alert() method 6](#_Toc99030676)

[Summary 7](#_Toc99030677)

[JavaScript confirm 7](#_Toc99030678)

[Introduction to JavaScript confirm() method 7](#_Toc99030679)

[Summary 8](#_Toc99030680)

[JavaScript prompt 8](#_Toc99030681)

[Introduction to JavaScript prompt() method 8](#_Toc99030682)

[JavaScript prompt() examples 8](#_Toc99030683)

[**1) Display a prompt dialog** 9](#_Toc99030684)

[**2) Convert a user input to a number** 9](#_Toc99030685)

[Summary 9](#_Toc99030686)

[JavaScript setTimeout 9](#_Toc99030687)

[Introduction to JavaScript setTimeout() 9](#_Toc99030688)

[JavaScript setTimeout() example 10](#_Toc99030689)

[**HTML** 10](#_Toc99030690)

[**JavaScript** 10](#_Toc99030691)

[**Output** 10](#_Toc99030692)

[How JavaScript setTimeout() works 10](#_Toc99030693)

[Summary 14](#_Toc99030694)

[JavaScript setInterval 14](#_Toc99030695)

[Introduction to JavaScript setInterval() 14](#_Toc99030696)

[JavaScript setInterval() example 15](#_Toc99030697)

[Summary 16](#_Toc99030698)

[JavaScript Location 16](#_Toc99030699)

[JavaScript Location properties 16](#_Toc99030700)

[**Location.href** 16](#_Toc99030701)

[**Location.protocol** 17](#_Toc99030702)

[**Location.host** 17](#_Toc99030703)

[**Location.port** 17](#_Toc99030704)

[**Location.pathname** 17](#_Toc99030705)

[**Location.search** 17](#_Toc99030706)

[**Location.hash** 17](#_Toc99030707)

[**Location.origin** 17](#_Toc99030708)

[**Location.username** 17](#_Toc99030709)

[**Location.password** 17](#_Toc99030710)

[Manipulating the location 18](#_Toc99030711)

[**assign()** 18](#_Toc99030712)

[**replace()** 18](#_Toc99030713)

[**reload()** 18](#_Toc99030714)

[Summary 19](#_Toc99030715)

[How To Get Query String in JavaScript 19](#_Toc99030716)

[Useful URLSearchParams methods 19](#_Toc99030717)

[**keys() example** 20](#_Toc99030718)

[**values() example** 20](#_Toc99030719)

[**entries() example** 20](#_Toc99030720)

[Check if a query string parameter exists 21](#_Toc99030721)

[Summary 21](#_Toc99030722)

[JavaScript Redirect 21](#_Toc99030723)

[Introduction to the JavaScript redirect 21](#_Toc99030724)

[Redirect to a new URL 21](#_Toc99030725)

[Redirect to a relative URL 22](#_Toc99030726)

[Redirect upon on page loading 22](#_Toc99030727)

[Summary 22](#_Toc99030728)

[JavaScript Navigator 23](#_Toc99030729)

[Introduction to the JavaScript Navigator object 23](#_Toc99030730)

[JavaScript Navigator Properties & Methods 23](#_Toc99030731)

[JavaScript Screen 26](#_Toc99030732)

[Introduction to the JavaScript Screen object 26](#_Toc99030733)

[JavaScript Screen properties 26](#_Toc99030734)

[JavaScript History 27](#_Toc99030735)

[Introduction to the JavaScript history object. 27](#_Toc99030736)

[Using JavaScript history for navigation 28](#_Toc99030737)

[**Move backward** 28](#_Toc99030738)

[**Move forward** 28](#_Toc99030739)

[**Move to a specific URL in the history** 28](#_Toc99030740)

[Summary 29](#_Toc99030741)

# **JavaScript BOM**

The Browser Object Model (BOM) is the core of JavaScript on the web. The BOM provides you with objects that expose the web browser’s functionality.

# **JavaScript Window**

**Summary**: in this tutorial, you will learn about the JavaScript window object which is the global object of JavaScript in the browser and exposes the browser ‘s functionality.

## The window object is the Global object

The global object of JavaScript in the browser is the window object. It means that all [variables](https://www.javascripttutorial.net/javascript-variables/) and [functions](https://www.javascripttutorial.net/javascript-function/) declared globally with the var keyword become the [properties](https://www.javascripttutorial.net/javascript-object-properties/) and methods of the window object. For example:

var counter = 1;

var showCounter = () => console.log(counter);

console.log(window.counter);

window.showCounter();

Code language: JavaScript (javascript)

Output:

1

counter 1

Since the counter variable and the showCounter() function are declared globally with the var keyword, they are automatically added to the window object. If you don’t want to pollute the window object, you can use the [let](https://www.javascripttutorial.net/es6/javascript-let/) keyword to declare variables and functions.

## The window object exposes the browser’s functionality

The window object exposes the functionality of the web browser to the webpage.

### **1) Window size**

The window object has four properties related to the size of the window:

* The innerWidth and innerHeight properties return the size of the page viewport inside the browser window (not including the borders and toolbars).
* The outerWidth and outerHeight properties return the size of the browser window itself.

Also, document.documentElement.clientWidth and document.documentElement.clientHeight properties indicate the width and height of the page viewport. To get the size of the window, you use the following snippet:

const width = window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

const height = window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

Code language: JavaScript (javascript)

### **2) Open a new window**

To open a new window or tab, you use the window.open() method:

window.open(url, windowName, [windowFeatures]);

Code language: CSS (css)

The window.open() method accepts three arguments: the URL to load, the window target and a string of window features.

The third argument is a command-delimited string of settings specifying displaying information for the new window such as width, height, menubar, and resizable.

The window.open() method returns a WindowProxy object, which is a thin wrapper of the window object. In case the new window cannot be opened, it returns null.

For example, to open a new window that loads the page about.html at localhost, you use the following code:

let url = 'http://localhost/js/about.html';

let jsWindow = window.open(url,'about');

Code language: JavaScript (javascript)

The code opens the page about.html in a new tab. If you specify the height and width of the window, it will open the URL in a new separated window instead of a new tab:

let features = 'height=600,width=800',

url = 'http://localhost/js/about.html';

let jsWindow = window.open(url, 'about', features);

Code language: JavaScript (javascript)

To load another URL on an existing window, you pass an existing window name to the window.open() method. The following example loads the contact.html webpage to the contact window:

window.open('http://localhost/js/contact.html','about');

Code language: JavaScript (javascript)

Put it all together. The following code opens a new window that loads the webpage about.html and then after 3 seconds, it loads the webpage contact.html in the same window:

let features = 'height=600,width=800',

url = 'http://localhost/js/about.html';

let jsWindow = window.open(url, 'about', features);

setTimeout(() => {

window.open('http://localhost/js/contact.html', 'about')

}, 3000);

Code language: JavaScript (javascript)

### **3) Resize a window**

To resize a window you use the resizeTo() method of the window object:

window.resizeTo(width,height);

Code language: JavaScript (javascript)

The following example opens a new window that loads the http://localhost/js/about.html page and the resize the window to (600,300) after 3 seconds:

let jsWindow = window.open(

'http://localhost/js/about.html',

'about',

'height=600,width=800');

setTimeout(() => {

jsWindow.resizeTo(600, 300);

}, 3000);

Code language: JavaScript (javascript)

The window.resizeBy() method allows you to resize the current window by a specified amount:

window.resizeBy(deltaX,deltaY);

Code language: JavaScript (javascript)

For example:

let jsWindow = window.open(

'http://localhost/js/about.html',

'about',

'height=600,width=600');

*// shrink the window, or resize the window*

*// to 500x500*

setTimeout(() => {

jsWindow.resizeTo(-100, -100);

}, 3000);

Code language: JavaScript (javascript)

### **4) Move a window**

To move a window to a specified coordinate, you use the moveTo() method:

window.moveTo(x, y);

Code language: JavaScript (javascript)

In this method, x and y are horizontal and vertical coordinates to be moved to. The following example opens a new window and moves it to (0,0) coordinate after 3 seconds:

let jsWindow = window.open(

'http://localhost/js/about.html',

'about',

'height=600,width=600');

setTimeout(() => {

jsWindow.moveTo(500, 500);

}, 3000);

Code language: JavaScript (javascript)

Similarly, you can move the current window by a specified amount:

let jsWindow = window.open(

'http://localhost/js/about.html',

'about',

'height=600,width=600');

setTimeout(() => {

jsWindow.moveBy(100, -100);

}, 3000);

Code language: JavaScript (javascript)

### **5) Close a window**

To close a window, you use the window.open() method:

window.open()

Code language: JavaScript (javascript)

The following example opens a new window and closes it after 3 seconds:

let jsWindow = window.open(

'http://localhost/js/about.html',

'about',

'height=600,width=600');

setTimeout(() => {

jsWindow.close();

}, 3000);

Code language: JavaScript (javascript)

### **6) The window.opener property**

A newly created window can reference back to the window that opened it via the window.opener property. This allows you to exchange data between the two windows.

## Summary

* The window is the global object in the web browser.
* The window object exposes the functionality of the web browser.
* The window object provides methods for manipulating a window such as open(), resize(), resizeBy(), moveTo(), moveBy(), and close().

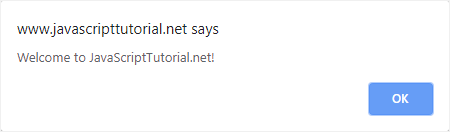
# **JavaScript alert**

**Summary**: in this tutorial, you will learn how to display an alert dialog by using the JavaScript alert() method.

Introduction to JavaScript alert() method

The browser can invoke a system dialog to display information to the user.

[Click here to show the alert system dialog.](https://www.javascripttutorial.net/javascript-bom/javascript-alert/)



The system dialog is not related to the webpage being shown in the browser. It also does not contain any HTML. Its appearance depends solely on the current operating system and browser, rather than CSS.

To invoke an alert system dialog, you invoke the alert() method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object.

window.alert(message);

Code language: JavaScript (javascript)

Or

alert(message);

The message is a string that contains information that you want to show to users.

For example:

alert('Welcome to JavaScriptTutorial.net!');

Code language: JavaScript (javascript)

When the alert() method is invoked, a system dialog shows the specified message to the user followed by a single **OK** button.

You use the alert dialog to inform users something that they have no control over e.g., an error. The only choice that users can make is to dismiss the dialog after reading the message.

Note that the alert dialog is synchronous and modal. It means that the code execution stops when a dialog is displayed and resumes after it has been dismissed. For example, the following code display an alert dialog after three seconds:

setTimeout(() => {

alert('3 seconds has been passed!')

}, 3000);

Code language: JavaScript (javascript)

Summary

* The alert() is a method of the window object.
* The alert() method is modal and synchronous.
* Use the alert() method to display information that you want users to acknowledge.

# **JavaScript confirm**

**Summary**: in this tutorial, you will learn how to display a confirmation dialog by using the JavaScript confirm() method.

Introduction to JavaScript confirm() method

To invoke a dialog with a question and two buttons OK and Cancel, you use the confirm() method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object:

let result = window.confirm(question);

Code language: JavaScript (javascript)

In this syntax:

* The question is an optional string to display in the dialog.
* The result is a Boolean value indicating whether the OK or Cancel button was clicked. If the OK button is clicked, the result is true; otherwise, the result is false.

Note that if a browser ignores in-page dialogs, then the result is always false.

The confirmation dialog is modal and synchronous. It means that the code execution stops when a dialog is displayed and resumes after it has been dismissed.

The following example uses the confirm() method to invoke a confirmation dialog. Based on the user’s selection, it displays the corresponding message based using the [alert()](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) method:

let result = confirm('Are you sure you want to delete?');

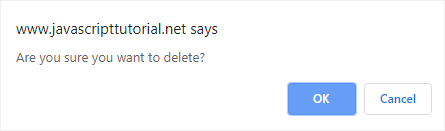
let message = result ? 'You clicked the OK button' :

'You clicked the Cancel button';

alert(message);

Code language: JavaScript (javascript)

[Click here to show the confirmation dialog](https://www.javascripttutorial.net/javascript-bom/javascript-confirm/)



Summary

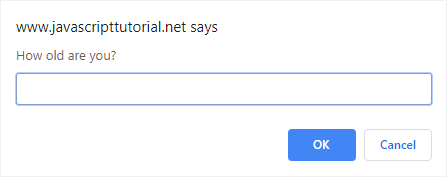
* The confirm() is a method of the window object.
* The confirm() shows a system dialog that consists of a question and two buttons: OK and Cancel.
* The confirm() returns true if the OK button was clicked or false if the Cancel button was selected.

# **JavaScript prompt**

**Summary**: in this tutorial, you will learn how to use the JavaScript prompt() method to display a dialog with a message prompting for user input.

## Introduction to JavaScript prompt() method

The prompt() is a method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object. The prompt() method instructs the web browser to display a dialog with a text, text input field, and two buttons OK and Cancel.



The dialog prompts the user to enter some text and wait until the user either submits or cancels it. The following illustrates the syntax of the prompt() method:

let result = window.prompt(message, default);

Code language: JavaScript (javascript)

In this syntax:

* The message is a string to display. If you omit it, nothing will display on the dialog.
* The default is a string containing the default value of the text input field.

The result is a string that contains the text entered by the user or null.

Like [alert()](https://www.javascripttutorial.net/javascript-bom/javascript-confirm/) and [confirm()](https://www.javascripttutorial.net/javascript-bom/javascript-confirm/), the prompt() is modal and synchronous. In other words, the code execution stops when the dialog is displayed and resumes after the dialog has been dismissed.

## JavaScript prompt() examples

Let’s take some examples to see how the prompt() works.

### **1) Display a prompt dialog**

The following example uses the prompt() to display a dialog that prompts users for their favorite programming languages:

let lang = prompt('What is your favorite programming language?');

let feedback = lang.toLowerCase() === 'javascript' ? `It's great!` :

`It's ${lang}`;

alert(feedback);

Code language: JavaScript (javascript)

### **2) Convert a user input to a number**

The result of the prompt() is a string. If you want to get the answer as a number, you should always cast the string into a number.

The following example uses prompt() to display a dialog that asks users for their ages. If users are 16 years old or above, they are eligible to join. Otherwise, they will not be.

let ageStr = prompt('How old are you?');

let age = Number(ageStr);

let feedback = age >= 16 ?

'You're eligible to join.' :

'You must be at least 16 year old to join.';

alert(feedback);

Code language: PHP (php)

## Summary

* The prompt() is a method of the window object.
* The prompt() shows a dialog that prompts the user to enter a text and wait until the user submit or cancel the dialog.
* The prompt() returns a string containing a string entered by the user or null if the user did not enter anything.

# **JavaScript setTimeout**

**Summary**: in this tutorial, you will learn how to use the JavaScript setTimeout() that sets a timer and executes a callback function after the timer expires.

## Introduction to JavaScript setTimeout()

The setTimeout() is a method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object. The setTimeout()  sets a timer and executes a [callback function](https://www.javascripttutorial.net/javascript-callback/) after the timer expires.

The following illustrates the syntax of setTimeout():

let timeoutID = setTimeout(cb [,delay], arg1, arg2,...);

Code language: JavaScript (javascript)

In this syntax:

* cb is a [callback function](https://www.javascripttutorial.net/javascript-callback/) to be executed after the timer expires.
* delay is the time in milliseconds that the timer should wait before executing the callback function. If you omit it, the delay defaults to 0.
* arg1, arg2, … are arguments passed to the cb callback function.

The setTimeout() returns a timeoutID which is a positive integer identifying the timer created as a result of calling the method.

The timeoutID can be used to cancel timeout by passing it to the clearTimeout() method.

## JavaScript setTimeout() example

The following creates two simple buttons and hooks them to the setTimeout() and clearTimeout().

When you click the Show button, the showAlert() is invoked and shows an alert dialog after 3 seconds. To cancel the timeout, you click the Cancel button.

### **HTML**

**<p>**JavaScript setTimeout Demo**</p>**

**<button onclick="showAlert();">**Show**</button>**

**<button onclick="cancelAlert();">**Cancel**</button>**

Code language: HTML, XML (xml)

### **JavaScript**

var timeoutID;

function showAlert() {

timeoutID = setTimeout(alert, 3000, 'setTimeout Demo!');

}

function clearAlert() {

clearTimeout(timeoutID);

}

Code language: JavaScript (javascript)

### **Output**

JavaScript setTimeout Demo

Show Cancel

## How JavaScript setTimeout() works

JavaScript is single-threaded therefore it can only do one task at a time. It means that it can only carry a single task a given time. Besides the JavaScript engine, the web browser has other components such as [Event Loop](https://www.javascripttutorial.net/javascript-event-loop/), [Call Stack](https://www.javascripttutorial.net/javascript-call-stack/), and Web API.

When you call the setTimeout(), the JavaScript engine creates a new function [execution context](https://www.javascripttutorial.net/javascript-execution-context/) and places it on the [call stack](https://www.javascripttutorial.net/javascript-call-stack/).

The setTimeout() executes and creates a timer in the Web APIs component of the web browser. When the timer expires, the callback function that was passed in the setTimeout() is placed to the callback queue.

The [event loop](https://www.javascripttutorial.net/javascript-event-loop/) monitors both the call stack and the callback queue. It removes the callback function from the callback queue and places it to call stack when the call stack is empty.

Once the callback function is on the call stack, it is executed.

See the following example:

function task() {

console.log('setTimeout Demo!')

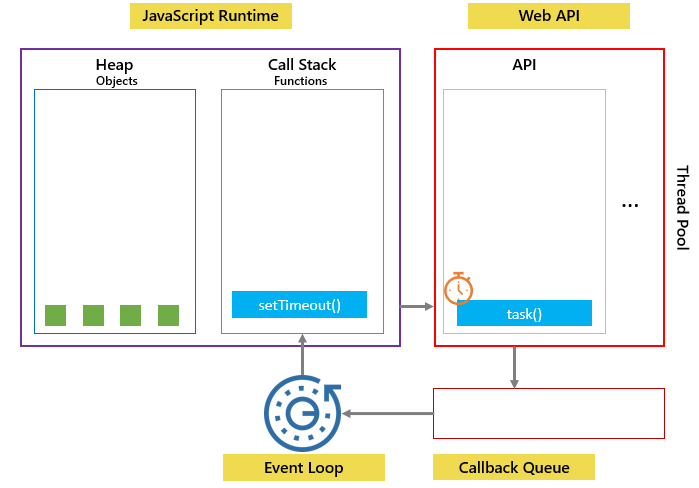
}

setTimeout(task, 3000);

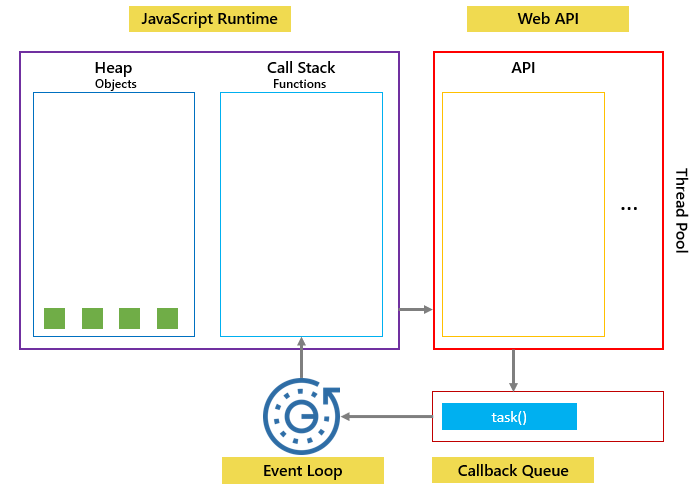
Code language: JavaScript (javascript)

In this example:

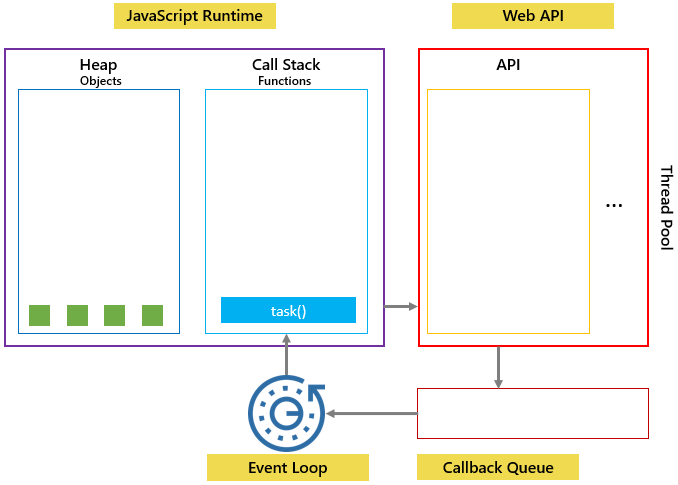
First, the setTimeout() is placed on the call stack. It creates a timer on Web API.



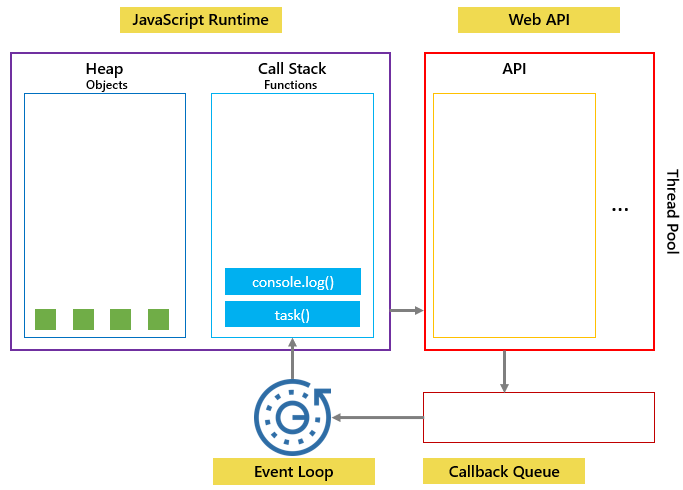
Second, after roughly 3 seconds, the timer expires, the task is pushed to the callback queue and waited for the next opportunity to execute.



Third, because the call stack is empty, the event loop removes the task() from the callback queue, places it on the call stack, and executes it:



Fourth, the console.log() in the setTimeout() executes that creates a new function execution context.



Finally, the console.log() and task() are popped out of the call stack once they are completed.

## Summary

* setTimeout() is a method of the window object.
* setTimeout() sets a timer and executes a callback function when the timer expires.

# **JavaScript setInterval**

**Summary**: in this tutorial, you will learn how to use the JavaScript setInterval() to repeatedly call a function with a fixed delay between each call.

Introduction to JavaScript setInterval()

The setInterval() is a method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object. The setInterval() repeatedly calls a function with a fixed delay between each call.

The following illustrates the syntax of the setInterval():

let intervalID = setInterval(callback, delay,[arg1, arg2, ...]);

Code language: JavaScript (javascript)

In this syntax:

* The callback is a [callback function](https://www.javascripttutorial.net/javascript-callback/) to be executed every delay milliseconds.
* The delay is the time (in milliseconds) that the timer should delay between executions of the callback function.
* The arg1, … argN are the arguments that are passed to the callback function.

The setInterval() returns a numeric, non-zero number that identifies the created timer. You can pass the intervalID to the clearInterval() to cancel the timeout.

Note that the setInterval() works like the [setTimeout()](https://www.javascripttutorial.net/javascript-bom/javascript-settimeout/) but it repeatedly executes a callback once every specified delay.

JavaScript setInterval() example

The following example uses the setInterval() and clearInterval() to change the color of a heading once a second once you press the Start button. If you stop the button, the clearInterval() will cancel the timeout.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="UTF-8" />**

**<title>**JavaScript setInterval Demo**</title>**

**<script>**

let intervalID;

function toggleColor() {

let e = document.getElementById('flashtext');

e.style.color = e.style.color == 'red' ? 'blue' : 'red';

}

function stop() {

clearInterval(intervalID);

}

function start() {

intervalID = setInterval(toggleColor, 1000);

}

**</script>**

**</head>**

**<body>**

**<p id="flashtext">**JavScript setInterval Demo**</p>**

**<button onclick="start()">**Start**</button>**

**<button onclick="stop()">**Stop**</button>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

JavaScript setInterval Demo

Start Stop

Summary

* The setInterval() repeatedly calls a function once a fixed delay between each call.
* The setInterval() returns a timeoutID that can be passed to the clearInterval() to cancel the timeout.

# **JavaScript Location**

**Summary**: in this tutorial, you will learn about the JavaScript Location object and how to manipulate the location effectively.

The Location object represents the current location (URL) of a document. You can access the Location object by referencing the location property of the window or document object.

Both window.location and document.location link to the same Location object.

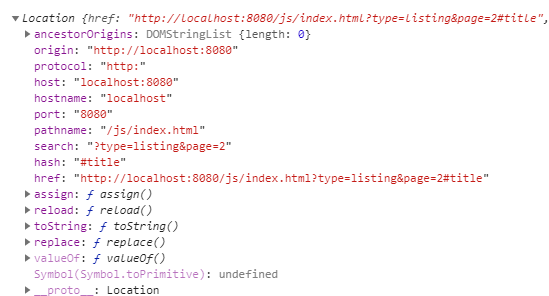
## JavaScript Location properties

Suppose that the current URL is:

http:*//localhost:8080/js/index.html?type=listing&page=2#title*

Code language: JavaScript (javascript)

The following picture illustrates the properties of the Location object:



### **Location.href**

The location.href is a string that contains the entire URL.

"http://localhost:8080/js/index.html?type=listing&page=2#title"

Code language: JSON / JSON with Comments (json)

### **Location.protocol**

The location.protocol represents the protocol scheme of the URL including the final colon (:).

'http:'

Code language: JavaScript (javascript)

### **Location.host**

The location.host represents the hostname:

"localhost:8080"

Code language: JSON / JSON with Comments (json)

### **Location.port**

The location.port represents the port number of the URL.

"8080"

Code language: JSON / JSON with Comments (json)

### **Location.pathname**

The location.pathname contains an initial '/' followed by the path of the URL.

"/js/index.html"

Code language: JSON / JSON with Comments (json)

### **Location.search**

The location.search is a string that represents the query string of the URL:

"?type=listing&page=2"

Code language: JSON / JSON with Comments (json)

### **Location.hash**

The location.hash returns a string that contains a ‘#’ followed by the fragment identifier of the URL.

"#title"

Code language: JSON / JSON with Comments (json)

### **Location.origin**

The location.origin is a string that contains the canonical form of the origin of the specific location.

"http://localhost:8080"

Code language: JSON / JSON with Comments (json)

### **Location.username**

The location.username is a string that contains the username before the domain name.

### **Location.password**

THe location.password is a string that represents the password specified before the domain name.

## Manipulating the location

The Location object has a number of useful methods: assign(), reload(), and replace().

### **assign()**

The assign() method accepts an URL, navigate to the URL immediately, and make an entry in the browser’s history stack.

location.assign('https://www.javascripttutorial.net');

Code language: JavaScript (javascript)

When the window.location or location.href is set to a URL, the assign() method is called implicitly:

window.location = 'https://www.javascripttutorial.net';

location.href = 'https://www.javascripttutorial.net';

Code language: JavaScript (javascript)

If you change hostname, pathname, or port property, the page reloads with the new value. Note that changing hash property doesn’t reload the page but does record a new entry in the browser’s history stack.

When a new entry is created in the browser’s history stack, you can click the back button of the browser to navigate to the previous page.

### **replace()**

The replace() method is similar to the assign() method except it doesn’t create a new entry in the browser’s history stack. Therefore, you cannot click the back button to go to the previous page.

The following code uses the replace() method to navigate to the URL https://www.javascripttutorial.net after 3 seconds:

setTimeout(() => {

location.replace('https://www.javascripttutorial.net');

}, 3000);

Code language: JavaScript (javascript)

### **reload()**

The reload() method reloads the currently displayed page. When you call the reload() method with no argument, the browser will reload the page in the most efficient way e.g., it loads the page resources from the browser’s cache if they haven’t changed since the last request.

reload();

To force a reload from the server, you pass true to the reload() method:

reload(true);

Code language: JavaScript (javascript)

Note that the code after the reload() may or may not execute, depending on many factors like network latency and system resources. Therefore, it is a good practice to place the reload()  as the last line in the code.

## Summary

* The Location object represents the current URL of a page. It can be accessed via window.location or document.location.
* The Location object has a number of properties that represent the URL such as protocol, host, pathname, and search.
* To manipulate the location, you set its properties new values or use assign(), replace(), and reload() methods.

# **How To Get Query String in JavaScript**

**Summary**: in this tutorial, you will learn how to use the URLSearchParams  to get query string parameters in JavaScript.

To get a query string you can access the search property of the [location](https://www.javascripttutorial.net/javascript-bom/javascript-location/) object:

location.search

Code language: CSS (css)

Assuming that the value of the location.search is:

'?type=list&page=20'

Code language: JavaScript (javascript)

To work with the query string, you can use the URLSearchParams object.

const urlParams = new URLSearchParams(location.search);

Code language: JavaScript (javascript)

The URLSearchParams is an [iterable object](https://www.javascripttutorial.net/es6/javascript-iterator/), therefore you can use the [for...of](https://www.javascripttutorial.net/es6/javascript-for-of/) structure to iterate over its elements which are query string parameters:

const urlParams = new URLSearchParams(location.search);

for (const [key, value] of urlParams) {

console.log(`${key}:${value}`);

}

Code language: JavaScript (javascript)

Output:

type:list

page:20

Code language: CSS (css)

## Useful URLSearchParams methods

The URLSearchParams has some useful methods that return iterators of parameter keys, values, and entries:

* keys() returns an iterator that iterates over the parameter keys.
* values() returns an iterator that iterates over the parameter values.
* entries() returns an iterator that iterates over the (key, value) pairs of the parameters.

### **keys() example**

The following example uses the keys() method to list all parameter names of a query string:

const urlParams = new URLSearchParams('?type=list&page=20');

for (const key of urlParams.keys()) {

console.log(key);

}

Code language: JavaScript (javascript)

Output:

type

page

### **values() example**

The following example uses the keys() method to list all parameter values of a query string:

const urlParams = new URLSearchParams('?type=list&page=20');

for (const value of urlParams.values()) {

console.log(value);

}

Code language: JavaScript (javascript)

Output:

list

20

Code language: PHP (php)

### **entries() example**

The following example uses the entries() method to list all pairs of parameter key/value of a query string:

const urlParams = new URLSearchParams('?type=list&page=20');

for (const entry of urlParams.entries()) {

console.log(entry);

}

Code language: JavaScript (javascript)

Output:

["type", "list"]

["page", "20"]

Code language: JSON / JSON with Comments (json)

## Check if a query string parameter exists

The URLSearchParams.has() method returns true if a parameter with a specified name exists.

const urlParams = new URLSearchParams('?type=list&page=20');

console.log(urlParams.has('type')); *// true*

console.log(urlParams.has('foo')); *// false*

Code language: JavaScript (javascript)

Output

true

false

Code language: JavaScript (javascript)

## Summary

* The URLSearchParams provides an interface to work with query string parameters
* The URLSearchParams is an iterable so you can use the for...of construct to iterate over query string parameters.
* The has() method of the URLSearchParams determines if a parameter with a specified name exists.

# **JavaScript Redirect**

**Summary**: in this tutorial, you will learn how to use JavaScript to redirect to a new URL or page.

Introduction to the JavaScript redirect

Sometimes, you want to redirect users to a new URL e.g., after users log in, you want to redirect them to the admin homepage.

JavaScript has the APIs that allow you to redirect to a new URL or page. However, JavaScript redirection runs entirely on the client side. Therefore it doesn’t return the status code 301 (move permanently) like a server redirection.

If you move the site to a separate domain or create a new URL for an old page, you should always use the server redirection.

Redirect to a new URL

To redirect to a new URL from the current page, you use the [location](https://www.javascripttutorial.net/javascript-bom/javascript-location/) object:

location.href = 'new\_url';

Code language: JavaScript (javascript)

For example:

location.href = 'https://www.javascripttutorial.net/';

Code language: JavaScript (javascript)

Assigning a value to the href property of the location object has the same effect as calling the assign() method of the location object:

location.assign('https://www.javascripttutorial.net/');

Code language: JavaScript (javascript)

Either of these calls will redirect to the new URL and create an entry in the history stack of the browser. It means that you can go back to the previous page via the **Back** button of the browser.

To redirect to a new URL without creating a new entry in the history stack of the browser, you use the replace() method of the location object:

location.replace('https://www.javascripttutorial.net/');

Code language: JavaScript (javascript)

Redirect to a relative URL

The following script redirects to the about.html which is on the same level as the current page.

location.href = 'about.html';

Code language: JavaScript (javascript)

The following script redirects to contact.html page located in the root folder:

location.href = '/contact.html';

Code language: JavaScript (javascript)

Redirect upon on page loading

If you want to redirect to a new page upon loading, you use the following code:

window.onload = function() {

location.href = "https://www.javascripttutorial.net/";

}

Code language: JavaScript (javascript)

Summary

* To redirect to a new URL or page, you assign the new URL to the location.href property or use the location.assign() method.
* The location.replace() method does redirect to a new URL but does not create an entry in the history stack of the browser.

# **JavaScript Navigator**

**Summary**: in this tutorial, you will learn about the JavaScript Navigator object and its properties.

## Introduction to the JavaScript Navigator object

The JavaScript Navigator provides information about the web browser and its capabilities. You can reference the Navigator object via the read-only window.navigator property.

The Navigator object has properties that convey the browser’s information. For example, the userAgent is a property of the window.navigator object. It is a long string that identifies the web browser.

window.navigator.userAgent

Code language: CSS (css)

In Google Chrome, you may see the following output:

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.117 Safari/537.36"

Code language: JSON / JSON with Comments (json)

Note that the userAgent may be a little bit different, depending on the Google Chrome version.

The different web browser provides specific capabilities which are not standardized. It’s better not to use the userAgent to identify the web browser because some web browsers allow users to modify the userAgent to pretend they are using a different browser.

For example, you may use the following code to detect if the current web browser is Internet Explorer:

if(navigator.userAgent.includes('MSIE')) {

*// IE, use specific features of IE*

} else {

*// not IE*

}

Code language: JavaScript (javascript)

To use a specific feature of a web browser, you can use the capability detection. For example:

if( typeof window.addEventListener === 'function' ) {

*// let's use addEventListener*

} else {

*// addEventListener is not supported, use another way*

}

Code language: JavaScript (javascript)

## JavaScript Navigator Properties & Methods

The following table illustrates the JavaScript Navigator properties and methods:

| **Property / Method** | **Description** |
| --- | --- |
| activeVrDisplays | Returns an array of every VRDisplay instance with its presenting property is set to true |
| appCodeName | Returns “Mozilla” even in non-Mozilla browsers. |
| appName | Returns the full browser name. |
| appVersion | Returns the browser version. However, it typically does not correspond to the actual version of the browser. |
| battery | Returns a BatteryManager object to interact with the Battery status API |
| buildId | Return the build number for the web browser. |
| connection | Returns a NetworkInformation object to interact with the Network information API |
| cookieEnabled | Returns true if if cookies are enabled; otherwise false. |
| credentials | Returns a CredentialsContainer to interact with the Credentials Management API |
| deviceMemory | Returns the amount of device memory in gigabytes. |
| doNotTrack | Returns the user’s preference of do-not-track . |
| geolocation | Returns a Geolocation object to interact with the Geolocation API. |
| getVRDisplays() | Returns an array of every VRDisplay instance if available. |
| getUserMedia() | Returns the stream associated with the available media device hardware. |
| hardwareConcurrency | Returns the number of processor cores of the device |
| javaEnabled | Determines if Java is enabled in the browser. |
| language | Returns the browser’s primary language. |
| languages | Returns an array of all the browser’s preferred languages. |
| locks | Returns a LockManager object to interact with the Web Locks API. |
| mediaCapabilities | Returns a MediaCapabilities object to interact with the Media capabilities API |
| mediaDevices | Returns the available media devices. |
| maxTouchPoints | Returns the maximum number of supported touch points for the device’s touchscreen |
| mimeTypes | Returns an array of MIME types registered with the browser. |
| onLine | Specifies if the browser is connected to the Internet. |
| oscpu | The operating system (OS) and/or CPU on which the browser is running. |
| permissions | Returns the Permissions object to interact with the Permissions API. |
| platform | Returns the system platform on which the browser is running. |
| plugins | Returns an array of installed browser’s plug-ins. |
| product | Returns the name of the product. |
| productSub | Returns the extra information about the product. |
| registerProtocolHandler() | Registers a website as a handler for a particular protocol. |
| requestMediaKeySystemAccess() | Returns a [Promise](https://www.javascripttutorial.net/es6/javascript-promises/) which resolves to a MediaKeySystemAccess object. |
| sendBeacon() | Asynchronously transmits a small payload. |
| serviceWorker | Returns the ServiceWorkerContainer used to interact with ServiceWorker object |
| share() | Calls the current platform’s native sharing mechanism. |
| storage | Returns the StorageManager object to interact with the Storage API. |
| userAgent | Represents the user-agent string of the browser. |
| vendor | Returns the brand name of the browser. |
| vendorSub | Returns extra information about the browser’s vendor. |
| vibrate() | Triggers the device to vibrate if vibration is supported. |
| webdriver | Determines if the browser is currently controlled by automation. |

# **JavaScript Screen**

**Summary**: in this tutorial, you will learn how to use the JavaScript Screen object to get the screen’s information of the current.

## Introduction to the JavaScript Screen object

The Screen object provides the attributes of the screen on which the current window is being rendered.

To access the Screen object, you use the screen property of the window object:

window.screen

Code language: JavaScript (javascript)

The Screen object is typically used by the web analytic software like Google Analytics to collect information of the client device on which the web browsers are running.

## JavaScript Screen properties

The window.screen object provides the following properties:

| **Property** | **Description** |
| --- | --- |
| availTop | A read-only property that returns the first pixel from the top that is not taken up by system elements. |
| availWidth | A read-only property that returns the pixel width of the screen minus system elements. |
| colorDepth | A read-only property that returns the number of bits used to represent colors. |
| height | Represents the pixel height of the screen. |
| left | Represents the pixel distance of the current screen’s left side. |
| pixelDepth | A read-only property that returns the bit depth of the screen. |
| top | Represents the pixel distance of the current screen’s top. |
| width | Represents the pixel width of the screen. |
| orientation | Returns the screen orientation as specified in the Screen Orientation API |
| availTop | A read-only property that returns the first pixel from the top that is not taken up by system elements. |
| availWidth | A read-only property that returns the pixel width of the screen minus system elements. |
| colorDepth | A read-only property that returns the number of bits used to represent colors. |
| height | Represents the pixel height of the screen. |
| left | Represents the pixel distance of the current screen’s left side. |
| pixelDepth | A read-only that returns the bit depth of the screen. |
| top | Represents the pixel distance of the current screen’s top. |
| width | Represents the pixel width of the screen. |
| orientation | Returns the screen orientation as specified in the Screen Orientation API |

# **JavaScript History**

**Summary**: in this tutorial, you will learn how to access the browser’s session history by using the JavaScript history object.

## Introduction to the JavaScript history object.

When you launch the web browser and open a new webpage, the web browser creates a new entry in its history stack.

If you [navigate to another webpage](https://www.javascripttutorial.net/javascript-bom/javascript-redirect/), the web browser also creates a new entry in the history stack.

The history stack stores the current page and previous pages that you visited.

To manipulate the history stack, you use the history object which is a property of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object:

window.history

Code language: JavaScript (javascript)

For the security reason, it’s not possible to query the pages that a user have visited. However, you can use the history object to navigate back and forth without knowing the exact URL.

## Using JavaScript history for navigation

The history object provides three methods for navigating between pages in the history stack:

* back()
* forward()
* go()

### **Move backward**

To move backward through history, you use the back() method:

window.history.back();

Code language: CSS (css)

Or

history.back();

Code language: CSS (css)

This behaves like you click the **Back** button in the toolbar of the web browser.

### **Move forward**

Similarly, you can move forward by using the forward() method:

history.forward();

Code language: CSS (css)

It works like when you click the **Forward** button.

### **Move to a specific URL in the history**

To move to a specific URL in the history stack, you use the go() method. The go() method accepts an integer that is the relative position to the current page. The current page’s position is 0.

For example, to move backward you use:

history.go(-1);

Code language: CSS (css)

It is like the back() method.

To move forward a page, you just call:

history.go(1)

Code language: CSS (css)

To refresh the current page, you either pass 0 or no argument to the go() method:

history.go(0);

history.go();

Code language: CSS (css)

To determine the number of URLs in the history stack, you use the length property:

history.length

Code language: CSS (css)

## Summary

* The window.history object allows you to access the history stack of the browser.
* To navigate to a URL in the history, you use the back(), forward(), and go() methods.
* The history.length returns the number of URLs in the history stack.