# Window Object in JavaScript

1.The **window** object in JavaScript stands for the current web page that is being seen in a browser window. It gives access to the browser’s methods and attributes as the global object in the browser environment. The window object is the global object in the web browser environment. It represents the current window or frames that the JavaScript code is running in.

2.A browser window is represented by the window object. The browser produces one window object for the HTML content and one extra window object for each frame when a document contains frames (iframe> elements).

3.Using the *document.defaultView*property, one may get a window for a certain document. The window object’s characteristics include global variables. The window object’s methods are known as global functions.

|  |  |
| --- | --- |
| **Property Name** | **Description** |
| **window.document** | The HTML document that is shown in the window is represented by the Document object, which is referred to by the document property. |
| **window.console** | The console gives the window’s Console Object back. |
| **window.location** | The location attribute makes reference to the Location object, which has data about the page’s current URL. |
| **window.defaultStatus** | It is now Abandoned. |
| **window.closed** | If a window is closed, it returns a true boolean value. |
| **window.frameElement** | It gives the window’s current frame back. |
| **window.frame** | returns every window item currently active in the window. |
| **window.history** | Retrieves the window’s History object. |
| **window.length** | It gives the number of iframe> elements currently present in the window. |
| **window.localStorage** | provides the ability to store key/value pairs in a web browser. stores data without a time. |
| **window.innerWidth and window.innerHeight** | Without including the toolbars and scrollbars, these characteristics describe the width & height of the browser window. |
| **window.opener** | It returns a pointer to the window that created the window in the opener function. |
| **window.outerHeight** | You can use outerHeight to return the height of the browser window, including toolbars and scrollbars. |
| **window.outerWidth** | You can outerWidth to get the width of the browser window, including toolbars and scrollbars. |
| **window.name** | Returns or sets a window’s name. |
| **window.parent** | Brings up the current window’s parent window. |
| **window.sessionStorage** | Provides the ability to store key/value pairs in a web browser. Contains data for a single session. |
| **window.scrollX** | It is a pageXOffset alias. |
| **window.scrollY** | It is a pageYOffset alias. |
| **window.self** | It provides the window’s current state. |
| **window.status** | It is now Deprecated. Don’t employ it. |
| **window.top** | It provides the top-most browser window back. |
| **window.screen** | The screen attribute makes reference to the Screen object, which stands in for the screen that the browser is shown on. |
| **window.history** | The History object, which includes details about the current page’s browsing history, is the subject of the history property. |
| **window.pageXOffset** | The number of pixels that the current document has been scrolled horizontally. |
| **window.pageYOffset** | The number of pixels that the current document has been scrolled vertically. |
| **window.screenLeft**: | The x-coordinate of the current window relative to the screen. |
| **window.screenTop** | The y-coordinate of the current window relative to the screen. |
| **window.screenX** | The x-coordinate of the current window relative to the screen (deprecated). |
| **window.screenY** | The y-coordinate of the current window relative to the screen (deprecated). |
| **window.navigator** | An object representing the browser and its capabilities |

**Example 1:**We will use the console property in this example to show how it works.

* Javascript

|  |
| --- |
| // This is console property  console.log(window.location) |

**Output:**

*Location {ancestorOrigins: DOMStringList, href: ‘chrome://new-tab-page/’, origin: ‘chrome://new-tab-page’, protocol: ‘chrome:’, host: ‘new-tab-page’, …}*

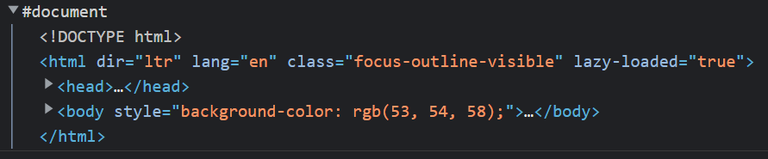
*window.location* object is a useful tool for interacting with the current URL of the window and redirecting the user to a different URL.

**Example 2:**In this example, we’ll use the document property in this example to show how it works:

* Javascript

|  |
| --- |
| // This is document property  window.document |

**Output:**



In JavaScript, the *window.document*property refers to the current HTML document that is being displayed in a window or frame. It allows you to access and manipulate the content of the HTML document, including the element nodes, attributes, and text content.

**Methods:**A method in JavaScript is a function connected to an object. You may conduct operations or compute values by calling methods on objects.

**Syntax:**

window.MethodName()

**NOTE:** Depending on the attributes, a parameter could have any value, including a string, number, object, and more.

Here is a list of some of the methods of the window object:

|  |  |
| --- | --- |
| **Property Name** | **Description** |
| **window.open()** | This method opens a new browser window or tab. |
| **window.close()** | This method closes the current window or tab. |
| **window.alert()** | This method displays an alert message to the user. |
| **window.prompt()** | This method displays a prompt message to the user and waits for their input. |
| **window.confirm()** | This method displays a confirm message to the user and waits for their response.window.focus: brings the current window or tab to the front. |
| **window.blur()** | Sends the current window or tab to the back. |
| **window.postMessage()** | Sends a message to the window or frame that is targeted by the specified WindowProxy object. |
| **window.scrollTo()** | Scrolls the window or frame to a specific position. |
| **window.scrollBy()** | Scrolls the window or frame by a specific number of pixels. |
| **window.resizeTo()** | Resizes the window to a specific size. |
| **window.resizeBy()** | Resizes the window by a specific number of pixels. |
| **window.atob()** | A base-64 encoded string is decoded via atob(). |
| **window.btoa()** | Base-64 encoding is done with btoa(). |
| **window.clearInterval()** | A timer set with setInterval() is reset. |
| **window.clearTimeout()** | The function clearTimeout() resets a timer specified with setTimeout(). |
| **window.focus()** | It switches the focus to the active window. |
| **window.getComputedStyle()** | This function returns the element’s current computed CSS styles. |
| **window.getSelection()** | It provides a Selection object corresponding to the user-selected text selection range. |
| **window.matchMedia()** | The provided CSS media query string is represented by a MediaQueryList object created by the matchMedia() function. |
| **window.moveBy()** | Relocates a window with respect to its present location. |
| **window.moveTo()** | Relocates a window to the given location. |
| **window.print()** | Displays what is currently displayed in the window. |
| **window.requestAnimationFrame()** | Before the subsequent repaint, the browser is asked to invoke a function to update an animation using the requestAnimationFrame() method. |
| **window.setInterval()** | At predetermined intervals, setInterval() calls a function or evaluates an expression (in milliseconds). |
| **window.setTimeout()** | When a certain amount of milliseconds have passed, the setTimeout() method calls a function or evaluates an expression. |
| window.stop() | It halts the loading of the window. |

**Conclusion:**The window object in JavaScript is the global object in the web browser environment. It represents the current window or frames that the JavaScript code is running in and provides a variety of properties and methods for interacting with the web browser and the document being displayed.

Some of the properties of the window object include innerHeight, innerWidth, screen, history, location, and navigator, which allow you to access information about the current window, the screen on which it is displayed, and the web browser.

Some of the methods of the window object include alert, confirm, prompt, open, close, focus, blur, postMessage, scrollTo, scrollBy, resizeTo, and resizeBy, which allow you to interact with the user and control the behavior of the window and the document being displayed.

Overall, the window object is an important part of the JavaScript language and is frequently used in web development to manipulate the web browser and the document being displayed.

# DOM (Document Object Model)

The Document Object Model (DOM) is a *programming interface* for **HTML(HyperText Markup Language)**and **XML**(Extensible markup language) documents. It defines the **logical structure** of documents and the way a document is accessed and manipulated.

**Note**: It is called a Logical structure because DOM doesn’t specify any relationship between objects.

DOM is a way to represent the webpage in a structured hierarchical way so that it will become easier for programmers and users to glide through the document. With DOM, we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object. Using DOM, the JavaScript gets access to HTML as well as CSS of the web page and can also add behavior to the HTML elements. so basically **Document Object Model is an API that represents and interacts with HTML or XML documents.**

**Why DOM is required?**

HTML is used to **structure**the web pages and Javascript is used to add **behavior**to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). **DOM is basically the representation of the same HTML document but in a different format with the use of objects**. Javascript interprets DOM easily i.e javascript can not understand the tags(<h1>H</h1>) in HTML document but can understand object h1 in DOM. Now, Javascript can access each of the objects (h1, p, etc) by using different functions.

**Structure of DOM:** DOM can be thought of as a Tree or Forest(more than one tree). The term **structure model**is sometimes used to describe the tree-like representation of a document.  Each branch of the tree ends in a node, and each node contains objects  Event listeners can be added to nodes and triggered on an occurrence of a given event. One important property of DOM structure models is ***structural isomorphism*:** if any two DOM implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

WhycalledanObjectModel?

Documents are modeled using objects, and the model includes not only the structure of a document but also the behavior of a document and the objects of which it is composed like tag elements with attributes in HTML.

**Properties of DOM:** Let’s see the properties of the document object that can be accessed and modified by the document object.



*Representation of the DOM*

* [**Window Object**](https://www.geeksforgeeks.org/properties-of-window-object/#:~:text=It%20represents%20an%20array%20that,frames%20of%20a%20given%20window.&text=It%20returns%20a%20reference%20to%20a%20DOMPoint%20object%2C%20which%20represents,point%20in%20a%20coordinate%20system.&text=It%20provides%20information%20of%20the%20URLs%20visited%20in%20the%20current%20window.&text=It%20represents%20the%20number%20of%20frames%20in%20the%20current%20window.)**:** Window Object is object of the browser which is always at top of the hierarchy.  It is like an API that is used to set and access all the properties and methods of the browser. It is automatically created by the browser.
* **Document object:** When an HTML document is loaded into a window, it becomes a document object. The ‘document’ object has various properties that refer to other objects which allow access to and modification of the content of the web page. If there is a need to access any element in an HTML page, we always start with accessing the ‘document’ object. Document object is property of window object.
* **Form Object:** It is represented by ***form*** tags.
* [**Link Object**](https://www.geeksforgeeks.org/html-dom-link-object/)**:** It is represented by ***link***tags.
* [**Anchor Object**](https://www.geeksforgeeks.org/html-dom-anchor-object/)**:** It is represented by ***a href*** tags.
* **Form Control Elements:**: Form can have many control elements such as text fields, buttons, radio buttons, checkboxes, etc.

**Methods of Document Object**:

* [**write**](https://www.geeksforgeeks.org/html-dom-write-method/)**(“string”):** Writes the given string on the document.
* [**getElementById()**](https://www.geeksforgeeks.org/html-dom-getelementbyid-method/)**:** returns the element having the given id value.
* [**getElementsByName()**](https://www.geeksforgeeks.org/html-dom-getelementsbyname-method/)**:** returns all the elements having the given name value.
* [**getElementsByTagName():**](https://www.geeksforgeeks.org/html-dom-getelementsbytagname-method/)returns all the elements having the given tag name.
* [**getElementsByClassName()**](https://www.geeksforgeeks.org/html-dom-getelementsbyclassname-method/)**:** returns all the elements having the given class name.

**Example:**In this example, We use HTML element id to find the DOM HTML element.

* HTML

|  |
| --- |
| <!DOCTYPE html>  <**html**>    <**body**>      <**h2**>COMputer</**h2**>        <!-- Finding the HTML Elements by their Id in DOM -->      <**p** id="intro">A Computer Science portal for computer.</**p**>      <**p**>This example illustrates the <**b**>getElementById</**b**> method.</**p**>      <**p** id="demo"></**p**>      <**script**>          const element = document.getElementById("intro");          document.getElementById("demo").innerHTML =            "computer introduction is: " + element.innerHTML;      </**script**>  </**body**>    </**html**> |

*Getting the HTML element by getElementById() Method*

**Example**: This example describes the representation of the HTML elements in the tree structure.

* html

|  |
| --- |
| <**table**>      <**ROWS**>          <**tr**>              <**td**>Car</**td**>              <**td**>Scooter</**td**>          </**tr**>          <**tr**>              <**td**>MotorBike</**td**>              <**td**>Bus</**td**>          </**tr**>      </**ROWS**>  </**table**> |



*HTML elements in tree-like structure*

What DOM is not?

* The Document Object Model is not a **binary description**where it does not define any binary source code in its interfaces.
* The Document Object Model is not used to describe**objects in XML or HTML**whereas the DOM describes XML and HTML documents as objects.
* The Document Object Model is not represented by a **set of data structures**; it is an interface that specifies object representation.
* The Document Object Model does not show the **criticality of objects**in documents i.e it doesn’t have information about which object in the document is appropriate to the context and which is not.

**Levels of DOM**:

* **Level 0:**Provides a low-level set of interfaces.
* **Level 1:**DOM level 1 can be described in two parts:**CORE**and **HTML.**
  + **CORE** provides low-level interfaces that can be used to represent any structured document.
  + **HTML**provides high-level interfaces that can be used to represent HTML documents.
* **Level 2:** consists of six specifications: **CORE2**, **VIEWS**, **EVENTS**, **STYLE**, **TRAVERSAL**, and **RANGE**.
  + **CORE2**: extends the functionality of CORE specified by DOM level 1.
  + **VIEWS**: views allows programs to dynamically access and manipulate the content of the document.
  + **EVENTS**: Events are scripts that are either executed by the browser when the user reacts to the web page.
  + **STYLE**: allows programs to dynamically access and manipulate the content of style sheets.
  + **TRAVERSAL**: This allows programs to dynamically traverse the document.
  + **RANGE**: This allows programs to dynamically identify a range of content in the document.
* **Level 3:**consists of five different specifications: **CORE3**, **LOAD and SAVE**, **VALIDATION**, **EVENTS**, and **XPATH**.
  + **CORE3**: extends the functionality of CORE specified by DOM level 2.
  + **LOAD and SAVE**: This allows the program to dynamically load the content of the XML document into the DOM document and save the DOM Document into an XML document by serialization.
  + **VALIDATION**: This allows the program to dynamically update the content and structure of the document while ensuring the document remains valid.
  + **EVENTS**: extends the functionality of Events specified by DOM Level 2.
  + **XPATH**: XPATH is a path language that can be used to access the DOM tree.

**The Window Object Model (Window DOM)**

The Window Object Model represents the entire browser window or tab and serves as the global object for JavaScript when running in a web browser environment. It encompasses the browser window itself, along with various properties, methods, and events associated with it. Here are some key aspects of the Window DOM:

1. **Global Scope**: Variables declared without the **var**, **let**, or **const** keywords inside a script block or included JavaScript file become properties of the window object. This global scope allows for the sharing of variables and functions across different parts of a web page.
2. **Window Properties**: The Window DOM includes properties such as **window.innerWidth**, **window.innerHeight**, **window.location**, and **window.document**. These properties provide information about the browser window, its dimensions, the URL it is displaying, and more.
3. **Window Methods**: It offers methods like **window.alert()**, **window.open()**, and **window.close()**. These methods enable developers to interact with the browser window itself, including opening new windows or displaying alerts.
4. **Window Events**: Events like **window.onload**, **window.onresize**, and **window.onunload** are part of the Window DOM. Developers can use these events to respond to various browser-level actions and user interactions.

**The Document Object Model (Document DOM)**

In contrast to the Window DOM, the Document Object Model represents the structure and content of an HTML document. It provides a hierarchical representation of the HTML elements on a web page, allowing developers to access, manipulate, and modify these elements dynamically. Here are some key aspects of the Document DOM:

1. **Hierarchical Structure**: The Document DOM is structured as a tree, with the **document** object at the root. HTML elements are represented as nodes in this tree, and they can be accessed and manipulated using JavaScript.
2. **Accessing Elements**: Developers can access HTML elements using methods like **document.getElementById()**, **document.querySelector()**, and **document.getElementsByTagName()**. This enables the dynamic modification of content and styles on a web page.
3. **Modifying Content**: The Document DOM allows developers to change the content of HTML elements, add or remove elements, and update element attributes. This capability is fundamental for creating dynamic and interactive web applications.
4. **Handling Events**: Events related to user interactions, such as clicks, keyboard input, and form submissions, can be captured and handled using event listeners attached to specific DOM elements.

**Key Differences and Use Cases**

1. **Scope**: The Window DOM has a global scope, making it suitable for managing browser-related tasks and interactions that affect the entire window or tab. In contrast, the Document DOM focuses on the structure and content of a single HTML document, making it ideal for handling page-specific interactions and dynamic content updates.
2. **Hierarchy**: While both models are hierarchical, the Document DOM is specifically concerned with the structure of HTML documents, whereas the Window DOM deals with the properties and methods associated with the browser window.
3. **Interactions**: The Window DOM is responsible for interactions at the browser level, such as opening new windows or tabs and managing browser history. The Document DOM, on the other hand, is primarily used for interactions within the context of a web page, including modifying content and responding to user actions.
4. **Use Cases**: The Window DOM is typically employed for tasks like opening pop-up windows, handling cookies, and managing the browser's location. The Document DOM is crucial for building interactive web applications, where developers need to access, manipulate, and update the content and structure of HTML documents.

Conclusion

In the world of JavaScript web development, understanding the distinctions between the Window Object Model and the Document Object Model is essential. Each model serves a unique purpose, with the Window DOM handling browser-related interactions and the Document DOM focusing on the structure and content of web pages. By leveraging these two models effectively, developers can create dynamic, interactive, and user-friendly web applications that cater to a wide range of user needs.