# 9. THE JAVASCRIPT DOCUMENT OBJECT MODEL

## INTRODUCTION

An HTML page is rendered (*painted*) in a browser. The browser assembles all the elements (*objects*) contained in the HTML page, downloaded from the web server, in its memory. Once done the browser then renders (*paints*) these objects in the browser window. Once the HTML page is rendered (*painted*) in the browser window, the browser can no longer recognize individual HTML elements (*objects*).

To create an interactive web page it is imperative that the browser continues to recognize individual HTML objects even after they are rendered in the browser window. This allows the browser to access the properties of these objects using the built-in methods of the object. Once the properties of an object are accessible then the functionality of the object can be controlled at will.

JavaScript enabled browsers are capable of recognizing individual objects in an HTML page, after the page has been rendered in the browser, because the JavaScript enabled browser recognizes and uses the Document Object Model (i.e *the DOM*).

Using the Document Object Model (DOM) JavaScript enabled browsers identify the collection of web page objects (*web page elements*) that have to be dealt with while rendering an HTML based, web page in the browser window.

The HTML objects (i.e. *the collection of web page elements*), which belong to the DOM, have a *descending* relationship with each other.

The topmost object in the DOM is the **Navigator** (i.e. *the browser*) itself. The next level in the DOM is the browser's **Window**. The next level in the DOM is the **Document** displayed in the browser's window.

Should the document displayed in the browser's window have an HTML **Form** coded in it, then the next level in the DOM is the **Form** itself.

The DOM hierarchy continues downward to encompass individual elements on a **FORM**, such as Text boxes, Labels, Radio buttons, Check boxes, Push buttons and so on, which belong to the form.

JavaScript's object hierarchy is mapped to the DOM, which in turn is mapped to the web page elements in the browser window. Hence, when a web page is rendered in a JavaScript enabled browser window, JavaScript is capable of uniquely identifying each element in the web page, because major elements of a web page are bound to the DOM.

The DOM that JavaScript recognizes is described in diagram 9.1.

JavaScript's DOM is referred to as an *instance hierarchy*.

### Instance

No HTML object is registered in the DOM by a JavaScript enabled browser unless they are assembled in memory prior being rendered in the browser window.

**The Navigator** – i.e. Netscape Navigator, Internet Explorer, Opera, Mosaic and so on.

```
Window
|-> Document
    |-> Anchor
    |-> Link
    |-> Form
        |-> textbox
        |-> textarea
        |-> radiabutton
        |-> checkbox
        |-> select
        |-> button
```

**Diagram 9.1:** JavaScript's DOM.

What this means is, if a document does not have any **Anchors** described in it the **Anchors** object will exist but it will be empty. If the document does not have any **Links** described in it the **Links** object will exist but it will be empty.

### Hierarchy

All objects on a web page are not created equal. Each exists in a set relationship with other objects on the web page. From diagram 9.1 the navigator occupies the topmost slot in the DOM followed by the Window object and so on.

Below the Window is the **Document** object. Below the document object three other objects exist. They are the **Anchor**, **Link** and **Form** objects. Individual form elements are found under the **Form** object.

In addition to the DOM, other objects currently recognized by a JavaScript enabled browser are *Plug-ins*, *Applets* and *Images*. Hence using a JavaScript enabled browser and JavaScript most of the major web page objects are accessible.

However, every single element of a web page rendered in the browser window, is **not** part of the DOM.

For example, HTML tags such as **<HEAD> . . . </HEAD>** or **<BODY> . . . </BODY>** are not part of the DOM. Presentation styles, headings, body text, H1 to H6 and so on are **not** part of the DOM hence **not** recognized by JavaScript.

Diagram 9.1 shows web page objects that are part of the DOM.

JavaScript however, recognizes presentation styles, headings, body text, H1 to H6 and so on, when JavaScript assisted Style Sheets [JSSS] are in a web page. JSSS is usually between the **<HEAD> . . . </HEAD>** HTML tags in a web page.

## THE JAVASCRIPT ASSISTED STYLE SHEETS DOM [JSSS DOM]

JSSS use JavaScript syntax to control a document's presentation style. When a JSSS is embedded in an HTML page within the **<HEAD> . . . </HEAD>** tags, then the JavaScript DOM picks up a whole new set of objects, which **add** to the standard DOM objects already recognized by JavaScript. The additional objects brought into the DOM by JSSS are shown in diagram 9.2.

By extending the DOM recognized by JavaScript by embedding JSSS in a web page, developers of web pages can access every element of a web page whether this element appears on the page when it is rendered in a client browser or not.

By accessing appropriate properties of the **Navigator** object, (i.e. *the Browser*), the topmost object in the DOM, JavaScript can recognize the browser type (i.e. *Netscape Navigator, Internet Explorer, Opera, Mosaic and so on*) and subsequently dispatch all HTML pages to the browser from the web server, with a **style** based on this knowledge. This is where the power of JavaScript really becomes visible in providing finely tuned web page content to a client's browser.

**Objects added to the DOM by the use of JSSS are:**

```
|-> Document
    |-> Tags
    |   |-> P
    |   |-> DIV
    |   |-> SPAN
    |   |-> H1 through H6
    |->classes
    |   |-> Tag Names
    |-> IDS
```

Diagram 9.2: JSSS's DOM.

Since JavaScript understands the DOM and can extend the DOM with the use of JSSS in a web page JavaScript understands Objects.

All objects have:

- **Properties** that determine the functionality of the object
- **Methods** that allow access to these properties
- **Events** that allow JavaScript code snippets to be connected to the object by being mapped to appropriate JavaScript event handlers.

Hence when a pre-determined event occurs the code snippet will execute. This is the traditional Object, Event driven, Code execution model of any object based programming environment.

Using appropriate JavaScript code snippets, which reference the properties of an object via its built-in methods, developers of web pages can actually control the functionality of any HTML object in the DOM (*or extended DOM*) while the HTML program executes (i.e. *at run time*).

JavaScript can access the methods of all objects belonging to the DOM and JSSS DOM. Hence using JavaScript, truly interactive web pages can be created.

**Note**

JavaScript is an **object-based** programming language it is **not** fully object oriented. It does not fully support basic object oriented programming capabilities such as classification, inheritance, encapsulation and information hiding.

JavaScript features are geared towards providing developers the capability to quickly generate scripts that will execute in the context of a web page within a JavaScript enabled browser **or** on a web server that understands JavaScript.

Although JavaScript does not provide all of the features of a full object oriented programming language, it does provide a suite of object-based features that are especially tailored to Browser or Server side scripting.

These features include the recognition of a number of predefined browser and server objects. JavaScript has the ability to control the behavior of these objects through their properties and methods.

In the following chapters the focus will be only on **browser** side JavaScripting. The browser in which JavaScript code snippets will always run correctly will be Netscape Communicator as JavaScript is a Netscape product. JavaScript is the natural language of Netscape Communicator.

## UNDERSTANDING OBJECTS IN HTML

HTML can be used to create a Graphical User Interface (GUI) in a web page. HTML is capable of accessing and using a number of objects that actually belong to the operating system (i.e. Windows). One such object is a *textbox*. A 'textbox' is used in an HTML form to accept user input.

The text box is an object, which belongs to the DOM. JavaScript recognizes a text box. JavaScript facilitates access to all the *methods* of a *textbox*. One of the methods of the textbox permits access to the contents of the text box. Hence, JavaScript can process the contents of any textbox in an HTML form.

### Properties Of HTML Objects

Just like real world objects, HTML objects have a number of **properties** that determine the behavior of that object. An object's properties can be referenced as:

ObjectName.PropertyName

For example, a *textbox* can have properties like *name, size* and so on:

## Methods Of HTML Objects

As seen, properties determine the state of an object. While building interactive web pages an object's properties need to be set dynamically, *i.e. when the object is being used.* Thus all object based programming environments **must** have facilities to *set* or *get* the value of object properties. This allows program code to control the state of the object at run time.

**Methods** of an object are used to *set* or *get* a value of an object's property. Thus determining how an object behaves. An object's methods can be referenced as:

**ObjectName.MethodName**

Using JavaScript it is possible to use an object's built-in methods and manipulate the object's properties at run time. This gives a great deal of creative control to web page developers when web pages have to be created on demand.

Once JavaScript code accepts client side input and / or reads a client's browser parameters, on demand web pages can be structured and sent to the browser from a web server.

# BROWSER OBJECTS

When any JavaScript enabled browser loads a web page, the browser automatically creates a number of JavaScript objects that map to the DOM (*or JSSS DOM*). It is the DOM, which provides JavaScript access to the HTML objects that are contained in the web page.

The JavaScript code snippets imbedded as part of the web page itself (i.e. embedded within the <SCRIPT> </SCRIPT> tags of *filename*.html) makes use of these objects to interact with the HTML objects in the web page.

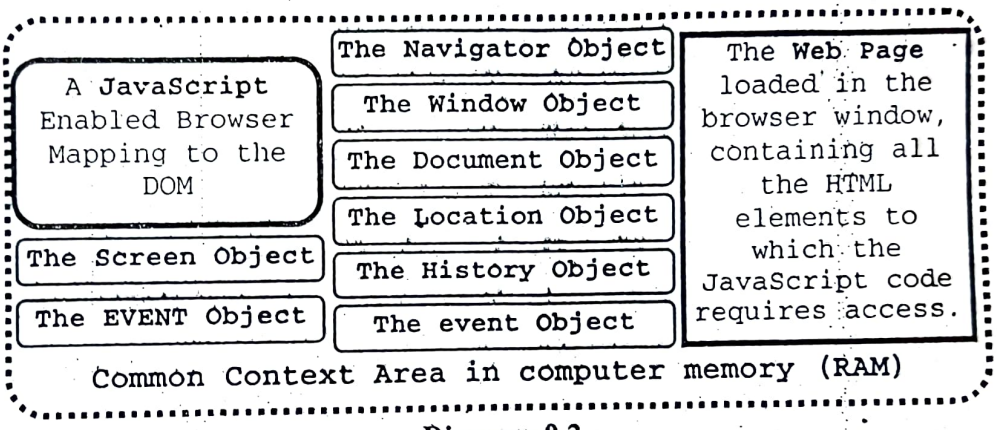


**Diagram 9.3**

The JavaScript objects created by [Netscape Communicator] are listed below:

| Object Name | Its Use |
|---|---|
| navigator | To access information about the browser that is executing the current script. |
| window | To access a browser *window* or a *frame* within the window. The window object is assumed to exist and does not require the **window** prefix when referring to its properties and methods. |
| document | To access the document currently loaded into a window. The document object refers to an HTML document that provides content, that is, one that has HEAD and BODY tags. |
| location | To represent a URL. It can be used to create a URL object, access parts of a URL, or modify an existing URL. |
| history | To maintain a history of the URL's accessed within a window. |
| event | To access information about the occurrence of an event. |
| EVENT | The EVENT (*capitalized*) object provides constants that are used to identify events. |
| screen | To access information about the size and color depth of a client computer's screen in which the current browser is running. |

**Table 9.1**

## How A JavaScript Enabled Browser Handles The Document Object

Any document (i.e. *the HTML page*) can contain various HTML objects such as:

- Images
- Image Maps
- Hyperlinks
- Frames
- Anchors
- Applets
- Multimedia/objects such a audio files, streaming video files
- A Form with various form elements

The browser creates one array in memory per HTML object in the document, thus registering each of these HTML objects.

If these HTML objects are actually contained in the HTML page then these arrays will hold indexed elements, which will point to the context area in memory where the HTML object are. Otherwise the array will exist, but will be empty (i.e. *have no elements in it*).

If there are multiple, similar HTML objects in the document (i.e *multiple images*) each array will have multiple (*indexed*) elements.

The array index value mapped to an HTML object will correspond to where the HTML object was described in the document. The first image in the document will have the array index as [0] (i.e. Images[0]), the next image in the document will have the array index of [1] and so on.

JavaScript provides access to the arrays and their elements. The values of *any/all* elements of each array can be identified, obtained. The values held in the elements of these arrays point to the context area in memory where the HTML objects actually are.

Once the context area in memory of an HTML object is known then using the **Methods** of the object, specific object **Properties** can be set using JavaScript. Thus the functionality of an HTML object can be controlled while the HTML page is running in the browser.

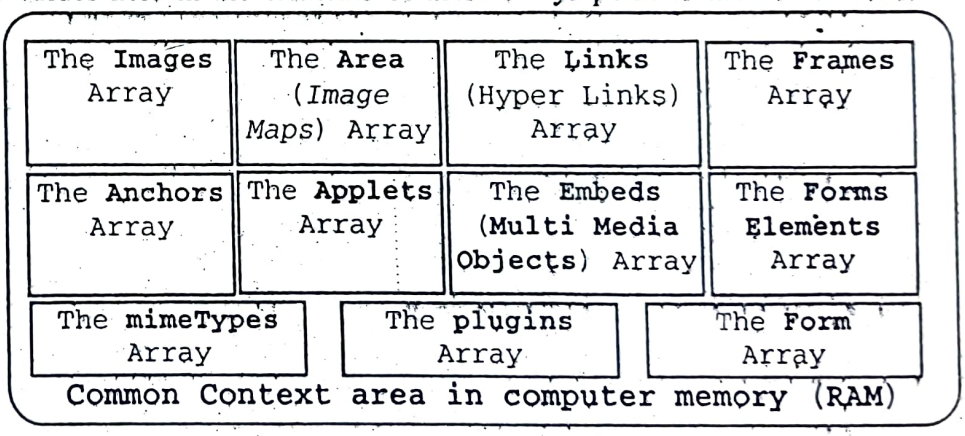


**Diagram 9.4**

The JavaScript arrays [created by Netscape Communicator] are listed below:

| | |
|---|---|
| Image/Images Array | To access an image that is embedded in an HTML document. The **images array** is used to access all image objects in a document. |
| Link / Links Array | To access a text or image-based source anchor of a hypertext link. The **links array** is used to access all link objects within a document. |
| Area | To access an **area** within a client-side image map. |
| Frame / Frames Array | To access an HTML frame. The **frames array** is used to access all frames within a window. |

Table 9.2

The JavaScript arrays [created by Netscape Communicator] are listed below: (Continued)

| | |
|---|---|
| Anchor/Anchors array | To access the target of a hyperlink. The **anchors array** is used to access all anchor objects within a document. |
| Applet/Applets array | To Access a Java applet. The **applets array** being used to access all the applets in a document. |
| Embed / Embeds array | To access an embedded object. The **embeds array** provides access to all the embedded objects in a document. |
| MimeType / MimeTypes array | To access information about a particular MIME type supported by a browser. The **mimeTypes array** is an array of all the mimeType objects supported by a browser. |
| Plugin / Plugins array | To access information about a particular browser plug-in. The **plugins array** is an array of all plug-ins supported by a browser. |
| Form / Forms array | To access an HTML form. The **forms array** is used to access all forms within a document. |

**Table 9.2** (Continued)

The JavaScript **form elements** array [created by Netscape Communicator]:

| | |
|---|---|
| elements | Access to all the form elements in the form. |
| text | To access a text field of a form. |
| textarea | To access a text area of a form. |
| radio | To access a set of radio buttons on the form or to access an individual radio button within the set. |
| checkbox | To access a checkbox on a form. |
| button | To access a form button that is not a reset or submit button. |
| submit | To access a submit button on a form. |
| reset | To access a reset button on a form. |
| select | To access a select list of a form. |
| option | The option object is used to access the elements of a select list. |
| password | To access a password field on a form. |
| hidden | To access a hidden object on a form. |
| fileupload | To access a file upload element of a form. |

**Table 9.3**

## THE WEB PAGE HTML OBJECT HIERARCHY

This is an *instance hierarchy*. This means if a web page does not have a specific HTML object defined in it the array associated with that specific HTML object will exist, but will have no elements.

### Access To Elements Of A Web Page

Conceptually once a web page is rendered (*painted*) in a browser window it is completely static.

For any program code to be able to interact with the web page, each element of the web page would have to be held in memory, with a unique name. The unique name translates to a context area in memory where the web page element resides.

Hence, while a web page is being assembled in memory (RAM) prior being rendered (*made visible*) in the browser's window, a JavaScript enabled browser creates several arrays as described earlier. These arrays hold references to individual web page objects in their (*indexed*) elements.

Referencing an appropriate element in its associated array provides access to each element of a web page. Hence, using JavaScript web page elements can be updated or processed. Once processed, the web page can be re-rendered in the browser. When re-rendered in the browser changes made to the web page elements will be visible.

If updation or processing of any web page element is done, *based on client input*, the web page is then an *interactive* web page.

## How A Web Page Element Is Manipulated

HTML tags are used to create (*instantiate*) objects in a web page. For example, <INPUT Type="TEXT"> will instantiates a text box on the web page when encountered in HTML code.

Each HTML object instantiated in a web page has **properties** and **methods** that allow access to the object's properties. Each HTML object has an **event** or several **events** bound to the object when the object was created. Thus an HTML object can recognize a specific event when it occurs.

Once the HTML object recognizes that an event has occurred this knowledge has to be passed to JavaScript so that JavaScript also recognizes that the 'event' occurred. To facilitate this JavaScript provides a number of named JavaScript 'Event Handlers'. The names of these event handlers are descriptively bound to an HTML object's event name.

**Example:**
A **Change** event is recognized by a text box when its contents change. JavaScript provides an event handler called **onChange** that is internally bound to the **Change** event of the text box. The **Change** event of the text box talks to the **onChange** event handler of JavaScript. The **onChange** event handler of JavaScript can then execute an appropriate JavaScript code snippet. For example:

```
<INPUT Type="TEXT" onChange="<myFunction>">
```

Here the JavaScript function **myFunction** is bound to the JavaScript event handler onChange. The assignment operator (=) does this binding.

The JavaScript, onChange event handler, is bound to the HTML object **TEXT** by being passed as one of its attributes.

Hence, as soon as the **Change** event of the text box occurs the JavaScript event handler onChange is invoked.

Since the JavaScript function myFunction is bound to the JavaScript event handler onChange, as soon as the onChange event handler is invoked the JavaScript code in myFunction executes.

# HANDLING (WEB PAGE) EVENTS USING JAVASCRIPT

A web page event could be associated with the action of the mouse cursor on the web page. Such as:

- A mouse-click on an object in a web page
- The movement of the mouse cursor across a web page
- The mouse cursor hovering at a specific place on a web page and so on

These will be events recognized by the Window object of the DOM.

Other web page events could be the opening or closing of a Window, the loading of an image in a web page and so on.

JavaScript's approach to working with web page elements is a multi step process:

- Identify a web page object
- Choose an appropriate event associated with the object
- Have a standard method of connecting an object's event and JavaScript code snippets. JavaScript **event handlers** mapped to an object's **events** do this.

JavaScript has several named event handlers that are mapped to an HTML object's events. To work with JavaScript it is necessary to understand how to use JavaScript 'Event Handlers' correctly.

JavaScript, event handlers, can be divided into two types **Interactive** and **Non Interactive**.

An interactive event handler depends on user interaction with an HTML page. For example, the JavaScript **onMouseOver** event handler is an interactive event handler. This requires the user to move the mouse cursor over a web page.

A JavaScript, non-interactive, event handler, does not need user interaction to be invoked. For example the JavaScript 'onLoad' event handler is a non-interactive event handler as it automatically executes whenever a form is loaded into a web page.

Table 9.4 details JavaScript event handlers that descriptively bound to HTML object events. As long as the HTML object has an associated event, JavaScript provides an associated, named, event handler.

## Named JavaScript Event Handlers

| JavaScript Event Handler | Will Be Called When |
|---|---|
| onAbort | The loading of an image is aborted as a result of user action |
| onBlur | A document, window, frame set, or form element loses current input focus. |
| onChange | A text field, text area, file-uploaded field or selection is modified and loses the current input focus. |
| onClick | A link, client-side image map area or document is clicked |
| onDblClick | A link, client-side image map area or document is double clicked |
| onDragDrop | A dragged object is dropped in a window or frame |
| onError | An error occurs during loading of an image, window or frame |
| onFocus | A document, window, frame set, or form element receives the current input focus |
| onKeyDown | The user presses a Key |
| onKeyPress | The user presses and releases a Key |
| onKeyUp | The user releases a Key |
| onLoad | An image, document or frame set is loaded |
| onMouseDown | The user presses a mouse button |
| onMouseMove | The user moves the mouse |
| onMouseOut | The mouse is moved out of a link or an area of a client side image map |
| onMouseOver | The mouse is moved over a link or an area of a client side image map |
| onMouseUp | The user releases a mouse button |
| onReset | The user resets a form by clicking on the form's reset button |
| onResize | The user resizes a window or frame |
| onSelect | Text is selected in a text field or a text area |
| onSubmit | The user presses a form's submit button |
| onUnload | The user exits a document or frame set |

Table 9.4

The naming convention followed by Java Script makes it easy to identify a JavaScript event handler. The JavaScript, event handler's name, simply has the string '*on*' added to the HTML object's event name (e.g. *onMouseOver*).

**Example:**

<A HRef=http://www.sctindia.com onMouseOver="*<JavaScript code snippet itself>* or *<A call to a* JavaScript *function>*">Text associated with the Link</A>

**Tip**

Here, the **onMouseOver** JavaScript event handler is bound to the hyperlink **<A> . . . </A>** HTML tag. When the mouse cursor moves over the hyperlink its **MouseOver** event occurs. When **MouseOver** event occurs a call is made to the JavaScript event handler **onMouseOver**. When the JavaScript **onMouseOver** event handler is invoked a JavaScript code snippet can execute directly or a JavaScript function can be called.

**Note**

If a JavaScript code snippet is directly passed as a value to an HTML attribute **and** is enclosed in **double quotes** ("), then double quotes (") cannot be used within the enclosed JavaScript code. Replacing these double quotes with single quotes (') where necessary.

To use the JavaScript DOM and manipulate its objects properties will help in raising this basic comfort level to a confidence level required for coding in JavaScript.

To achieve this several examples which use the HTML **<FORM> . . . </FORM>** tags along with their attributes mapped to JavaScript code will be used.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. Using the _____ _____ _____ JavaScript enabled browsers identify the collection of web page objects that have to be dealt with while rendering an HTML based web page in the browser window.
2. _____ _____ _____ _____ use JavaScript to control a document's presentation style
3. _____ of an object are used to set or get a value of an object's property.
4. JavaScript event handlers can be divided into two types _____ and _____.
5. _____ object is used to access information about the browser that is executing the current script

### TRUE OR FALSE

6. If the document does not have any links described in it, the Link object does not exist.
7. JavaScript 'onLoad' event handler is an interactive event handler as it automatically executes whenever a form is loaded into a web page.
8. The browser creates one array in memory per HTML object in the document.