**Document Object Model (DOM) and Browser Object Model (BOM)**

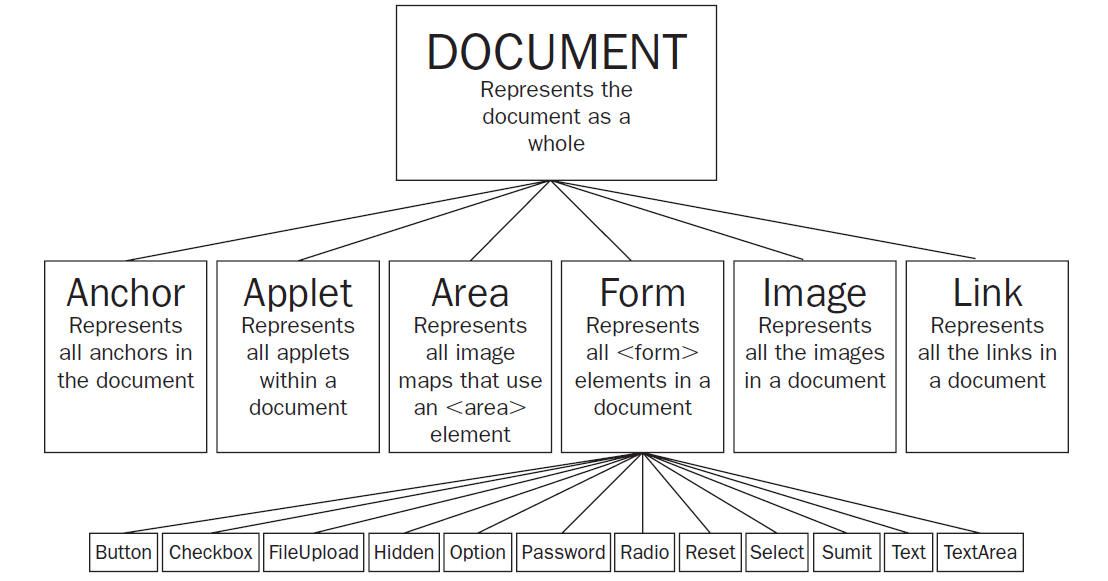
**The Document Object Model:**

As I mentioned at the start of the chapter, JavaScript by itself doesn’ t does much more than allow you to perform calculations or work with basic strings. In order to make a document more interactive, the script needs to be able to access the contents of the document and know when the user is interacting with it. The script does this by interacting with the browser by using the properties, methods and events set out in the interface called the Document Object Model.

The Document Object Model, or DOM, represents the web page that is loaded into the browser using a series of objects. The main object is the document object, which in turn contains several other child objects.

The DOM explains what *properties* of a document a script can retrieve and which ones it can alter; it also defines some *methods* that can be called to perform an action on the document.

Figure below, shows you an illustration of the Level 0 HTML Document Object Model (as you will see shortly, there are different levels of the DOM).



As you can see from Figure, the document object represents the whole document, and then each of the child objects represent a *collection* of similar tags within that document:

* The *anchor* collection represents all the anchors in a document that you can link to ( <a> elements with a name attribute).
* The applet collection represents all the applets within a document.
* The area collection represents all the image maps that use an <area> element in the document.
* The forms collection contains all the <form> tags in the document.
* The image collection represents all the images in a document.
* The link collection represents all the hyperlinks within a page.

The forms collection also has child objects to represent each of the different types of form controls that can appear on a form: Button, CheckBox, FileUpload, Hidden, Option, Password, Radio, Reset, Select, Submit, Text, and TextArea.

To better understand how to access the document using the DOM, take a look at the following simple document, which contains one form and two links:

<h1> User Registration </h1>

<form name=”frmLogin” action=”login.aspx” method=”post”>

Username <input type=”text” name=”txtUsername” size=”12”/ > <br />

Password < input type=”password” name=”pwdPassword” size=”12” / > <br />

<input type=”submit” value=”Log In” />

</form>

<p> New user? < a href=”register.aspx” > Register here </a> |

<a href=”lostPassword.aspx” > Retrieve password < /a > . </p>

The DOM would allow a script to access:

* The content of the form as part of the forms collection
* The two links as part of the links collection

**DOM (figure and html code discussed in classroom):**

<html>

<head>

<title>My Title</title>

</head>

<body>

<h1>Heading</h1>

<div id = ”one”>

<p>First Paragraph</p>

</div>

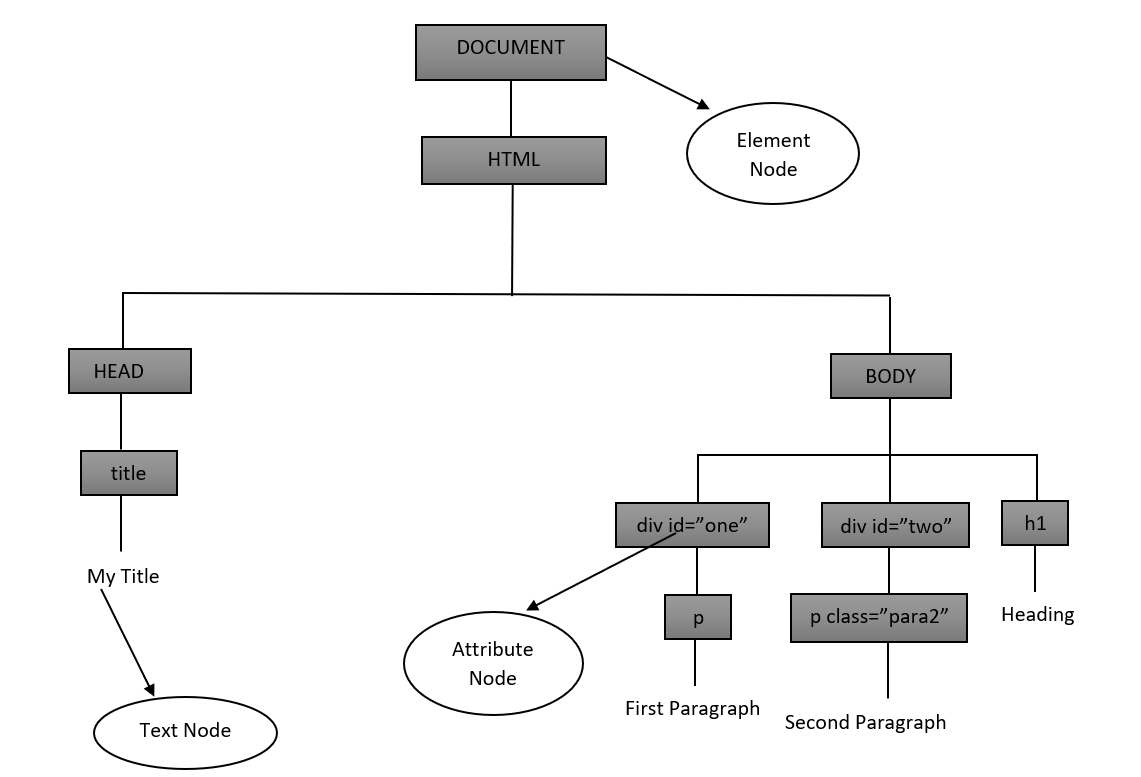
<div id = ”two”>

<p class = “para2”>Second Paragraph</p>

</div>

</body>

</html>



***Note: Every elements are treated as objects.***

**Accessing Values Using Dot Notation:**

In order to access the different objects in the DOM, you list the objects in the tree shown in Figure, starting with the document object, working down to the object that holds the data you are after. Each object is separated by a period or full - stop character; hence, this is known as a *dot notation*.

For example, in order to access the first link in the document, you would want the links object, which is a child of the document object, so you could use something like this:

document.links[0].href

There are four parts of this statement, three of which are separated by periods, to get to the first link:

* The word document indicates I am accessing the document object.
* The word links corresponds to the links collection (after all, this example is to retrieve the value of the first link in the document).
* The [0] indicates that I want the first link in the document. Rather confusingly, the items of a collection are numbered from 0 rather than 1, which means the second link in the links collection is represented using [1], the third using [2], and so on.
* I have indicated that I want to retrieve the href property for this link.

Each object has different properties that correspond to that type of element; for example, links have properties such as the href property that accesses the value of the href attribute on this <a> element. Similarly, a <textarea> object has properties such as cols , disabled, readOnly, and rows, which correspond to the attributes on that element.

Rather than using the names of the type of object (such as forms and links), you can use the value of name attributes on the elements to navigate through the document. For example, the following line requests the value of the password box:

document.frmLogin.pwdPassword.value

Again, there are four parts to this statement:

* The document object comes first again as it represents the whole page (it is the top - level object).
* The name of the form, frmLogin.
* This is followed by the name of the form control, pwdPassword.
* Finally, the property I am interested in is the value of the password box, and this property is called value.

Both of these approaches enable you to navigate through a document, choosing the elements and properties of those elements you are interested in. Then you can retrieve those values, perform calculations upon them, and provide alternative values.

*For the purpose of learning JavaScript, we are dealing with what is often called DOM Level 0 in this chapter because it works in most browsers. Its syntax was created before the W3C created its DOM Level 1, 2, and 3 recommendations (which get more complicated and have varying levels of support in different browsers). Once you are familiar with the basics, you can move on to look at these in more detail if you wish.*

*There is also a second type of object model, the Browser Object Model, which makes features of the browser available to the programmer, such as the* window *object, which can be used to create new pop-up windows. The Browser Object Model can vary from browser to browser, although most browsers have common support for core functionality. You learn about the* window *object later in the chapter.*

**The Document Object:**

In this section, we are going to take a closer look at the document object — this is the main object in the DOM and represents the document as a whole (and therefore allows you to access other child elements).

As you already know, an object can have properties that can tell you about the object, and methods to perform an action upon that object.

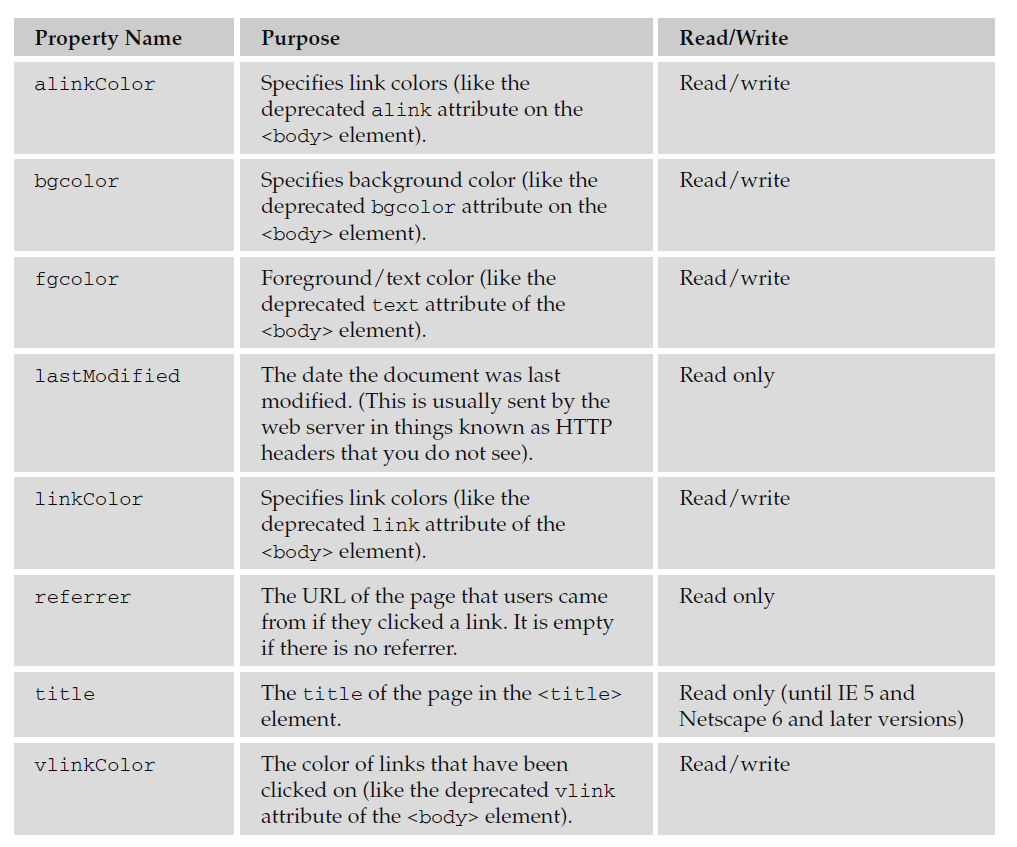
Once you understand how to work with one object, it ’ s much easier to work with all kinds of objects—and you will come across many different types of objects when you start programming.

**Properties of the Document Object:**

In the following table, you can see the properties of the document object. Several of these properties correspond to attributes that would be carried by the <body> element, which contains the document.

Many properties can be set as well as read. If you can set a property, it is known as a read/write property (because you can read it or write to it), whereas the ones you can only read are known as read-only.

You can see which properties can be read and which can be written to in the last column of the table that follows.



*The properties that correspond to deprecated attributes of the* <body> element *should generally be avoided because CSS should be used to style text, links, and backgrounds.*

To access any of the properties, again you use dot notation, so you can access the title of a document like so:

document.title

Or you could find out the date a document was last modified like so:

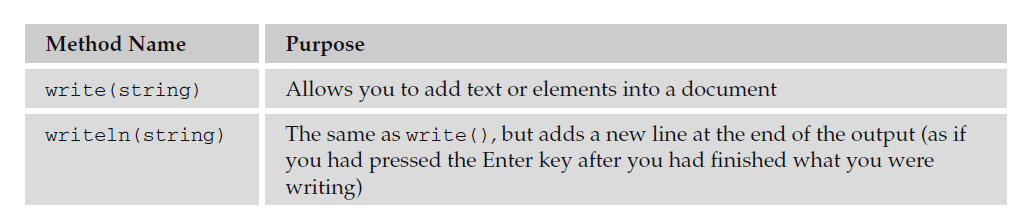
document.lastModified

Note that if the server does not support the lastModified property, IE will display the current date, while other browsers will often display 1 January 1970 (which is the date from which most computers calculate all dates).

**Methods of the Document Object:**

Methods perform actions and are always written followed by a pair of brackets. Inside the brackets of some methods, you can see things known as *parameters* or *arguments*, which can affect what action the method takes.

For example, in the table that follows, you can see two methods that write new content into the web page. Both of these methods need to know what should be written into the page, so they take a string as an argument (a *string* is a sequence of characters that may include letters, numbers, spaces, and punctuation), and the string is what gets written into the page.



You have already seen the write() method of the document object in above example, which showed how it can be used to write content into a document:

document.write(‘This is a document’);

You can also put something called an *expression* as a parameter of the write() method. For example, the following will write the text string Page last modified on followed by the last modified date of the document.

document.write(‘Page last modified on ‘ + document.lastModified);

You will see more about expressions later in the chapter, but in this case, the expression *evaluates* into (or results in) a string. For example, you might see something like Page last modified on 12th December 2009.

Now that you ’ ve seen the properties and methods of the document object, let ’ s look at the properties and methods of some of the other objects, too.

**The Browser Object Model or Window Objects:**

Every browser window and frame have a corresponding window object that is created with every instance of a <body> or <frameset> element. BOM refers to Windows objects in JavaScript.

For example, you can change the text that appears in the browser ’ s status bar using the status property of the window object. To do this, first you need to add a function in the head that is going to be triggered when the page loads, and then you use this function to indicate what should appear in the status bar:

<script type=”text/javascript”>

function statusBarText(){

window.status = “Did you see me down here?”

}

</script>

You then call this function from the <body> element’ s onload event, like so:

<body onload=”statusBarText()”>

**NOTE: windows.status, defaultStatus is not supported by the browser. This example is just for reference.**

**<scroll more>**

**<scroll more>**

**<scroll more>**

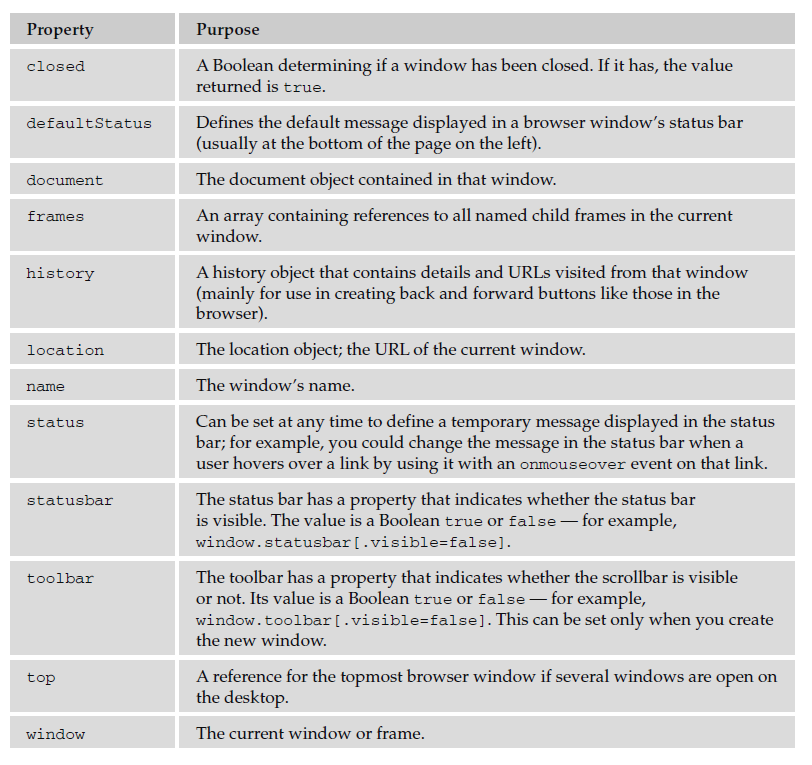
**<scroll more>**

**<scroll more>**

**<scroll more>**

**Properties:**

The table that follows lists the properties of the window object.



**Methods:**

The table that follows lists the methods of the window object.

