# JavaScript 101:
# JavaScript for Writers and Trainers

*Scott DeLoach*

*JavaScript is the most popular web-based programming language, and it offers many powerful features that can be used by writers and trainers. This paper will provide an introduction to JavaScript and its strengths and weaknesses. It also includes an overview of the Document Object Model and JavaScript events and methods. The paper concludes with some JavaScript examples that can be used to open and close Web-based training and Help systems.*

## OVERVIEW

JavaScript is a web-based programming language that can be used with Cascading Style Sheets (CSS), Dynamic HTML (DHTML), and the Document Object Model (DOM) to create interactive web pages. One of the biggest misconceptions about JavaScript is its relationship with Java. Although they share similar names, JavaScript has nothing to do with Java. JavaScript can be used to control Java applets, but they are very different languages. JavaScripts run on the client, so the user's browser must support JavaScript. JavaScripts are not compiled like Java applets. Instead, they are often stored in a JavaScript file. JavaScript and Java do share a similar syntax, but JavaScript is not a fully object-oriented language. It does use objects, but it does not support typical object-oriented features such as inheritance.

### Advantages and Disadvantages

JavaScript is easier to learn than most programming languages, especially for those who only need to modify existing scripts. From a technical standpoint, JavaScript can be more efficient than other languages, and it is well supported even in older browser versions. JavaScript's weaknesses are based on security issues with Web-based technologies. JavaScript cannot interact with the client's PC, and it cannot read or write files on the server. Since JavaScripts normally run on the client, it is very hard to hide your code from curious users.

## GETTING STARTED WITH JAVASCRIPT

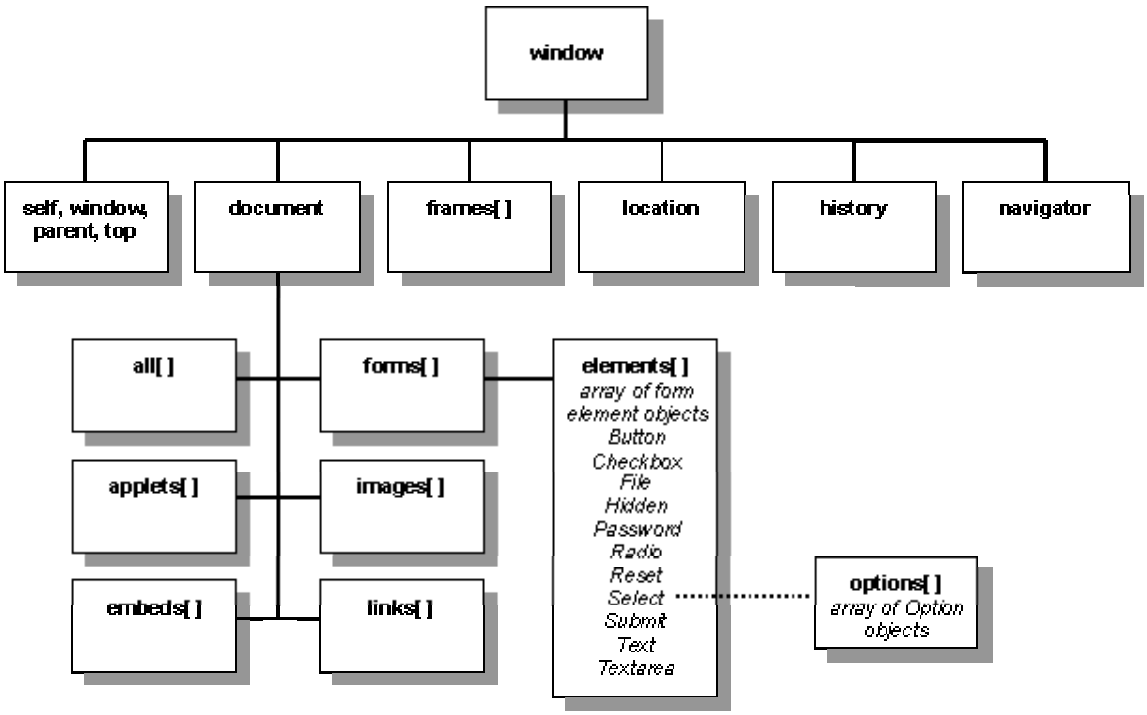
JavaScript revolves around objects. Everything on the page is an object: the text, links, images, forms, and tables. In general, objects are like nouns. Objects have three features: properties (adjectives), and events and methods (verbs). For example, a link could be coded as:

```
<a href="#" onClick="myScript()">
```

In this example, the object is the <a> or "anchor" tag. This tag has an "href" property that makes it a link. It also has an event: "onClick." If a user clicked on this link, it would run a JavaScript function called "myScript."

### The Document Object Model

The Document Object Model (or "DOM") is like an org chart that outlines how everything on the page is related and organized. It is used to move from one object to another, or to move down from the top (the window) to a specific object. Learning to navigate through and use the DOM is probably the trickiest part of learning JavaScript. To make matters worse, Internet Explorer and Netscape have different DOMs. With IE 6 and Netscape 6, these two browsers seem to be moving closer to supporting the ECMA's JavaScript standard. However, Netscape 4's and IE 4's DOMs are very different.




### The Document Object Model (high-level overview)

As an example of using the DOM, let's say that you have a frameset with two frames and you want a script in frame 1 to change a graphic in frame 2. You would use the DOM to move from frame 1 to frame 2, find the graphic, and change it's SRC property to a different image.

```
parent.frames(“frame2”).myicon.src = “newgraphic.gif”;
```

## "HELLO, WORLD!"

If you have ever learned a programming language, you know that they always begin with the "Hello, World!" example. So, how do we use JavaScript to write "Hello, World!" on the screen? It's only one line of code:

```
<script>document.write('Hello, World!');</script>
```

If you place this script inside the BODY tag of you page, it would write "Hello, World!" in your document. This script uses the document object's "write" method to write text to the page. Of

course, it's not incredibly useful. We could simply type in "Hello, World" as normal HTML text to get the same effect. However, we can use JavaScript to make this script more powerful.

## MAKING "HELLO, WORLD!" USEFUL

To make the "Hello, World!" example more useful, we could store the script in a JavaScript file so that we could reuse it in other files. We could also use variables to write different things to the page.

### Where to Place Your Scripts

Your scripts can be placed in three places: in the HEAD of your document, in the BODY of your document, or in an external file. Our original "Hello, World!" script is in the BODY of the document. If we moved it to the HEAD, we could reuse the script multiple times in the document. Normally, scripts in the HEAD are placed in functions and run when needed. For example, we could change "Hello, World!" to use a function:

```
<HEAD>
function helloWorld() {
document.write('Hello, World!');
}
</HEAD>
<BODY>
<script>helloWorld();</script>
</BODY>
```

Whenever we wanted to run the helloWorld() function, we would use:

```
<script>helloWorld();</script>
```

We are going to store our script in an external JavaScript "js" file. This will allow us to reuse it in other HTML files. To store the script in an external JS file, we need to link to the JS file from the HEAD of our document:

```
<script src="myscripts.js"></script>
```

Our JS file ("myscripts.js") would contain our function and any other scripts that we want to reuse in our files. We can still use the following code in the BODY to run the script:

```
<script>helloWorld();</script>
```

### Using Variables

Our current script can only write "Hello, World!" to the document. If we use variables, we can write different things to the document using the same script. For example, we could write the author's name or a copyright statement. Since we are going to be writing different things to the document, we should also change the script's name to something like autoText().

To use a variable with our script, we need to add the variable inside the parentheses when we call the script:

```
<script>autoText('author')</script>
```

Then, we need to change the function to do different things based on the variable:

```
function autoText(entry) {
if (entry == 'author') document.write('Scott DeLoach');
if (entry == 'copyright') document.write('Copyright 2002, User First
Services');
}
```

This script can now be used in multiple files to write different things to the document. If you need to update the information, you only have to change the autoText() function.

## JAVASCRIPT SYNTAX

JavaScript has some strict rule for naming and referring to variables and functions. Your function and variable names must begin with a letter, and they can only contain letters, numbers, and underscores. By convention, function and variable names begin with lowercase letters, but they use uppercase for additional words. So, the example above was named "autoText()." You will need to be careful with your names, especially the mixed case names. JavaScript is case-sensitive, so "autotext" is not the same as "autoText."

There are a lot of words that cannot be used as variable names. These "reserve" words are either currently used in JavaScript or they are being held for future use.

**Reserved words in JavaScript**

| | | | |
|---|---|---|---|
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super | |

## TRIGGERING SCRIPTS WITH EVENTS

The autoText() script runs when the page is loaded. However, we can also call scripts when the user clicks on a button, moves the mouse over an image, or presses a specified key. Each of these actions is an "event" in JavaScript: onClick, onMouseOver, and onKeyPress. Some events are used for the document's BODY as a whole, and some are used with specific tags such as images, links, and buttons. Events that are assigned to the BODY tag include:

- onLoad – when the document loads
- onUnload – when a new document is loaded
- onFocus – when the browser window receives focus
- onBlur – when the browser window loses focus
- onHelp – when the user presses F1 (IE only)
- onKeyPress – when the user presses a key (except the function keys)

Events that are assigned to tags include:

- onFocus – when the tag receives focus
- onBlur – when the tag loses focus
- onMouseOver – when the user moves the mouse over the tag
- onMouseOut – when the user moves the mouse away from the tag
- onChange – when a form element's value is changed
- onClick – when the tag is clicked
- onSubmit – when a form is submitted

We can use these events to run our scripts based on different user actions. For example, we can open a WBT system when the user clicks a button, open a Help system when the user presses the F1 key, and automatically close windows when they lose focus or when the user presses the ESC key.

## OPENING WBT FROM A BUTTON

We can use the onClick event to open a WBT system when the user clicks a button. The following code will add the "WBT" button:

```
<form>
<input type="button" value=”WBT" onClick="openWBT()">
</form>
```

The onClick event tells the button to run the "openWBT()" function when the button is clicked. The openWBT() function can be coded as:

```
<script>
function openWBT() {
window.open('WBTpage.htm','helpwin','toolbar=0,location=0,directories=0
,status=0,menubar=0,scrollbars=0,resizable=0,width=200,height=300');
}
</script>
```

This script is just one long line of code. It uses the window object's open method to open a new window. The window's properties are turned off by using "0"s and turned on by using "1"s. If you use this code, make sure that you do not use any spaces. A JavaScript bug in this method causes an error message to appear if you have spaces in the code.

## OPENING HELP WITH F1

Internet Explorer provides the onHelp method that runs when the user presses the F1 key. Netscape just ignores this method, so it's OK to use it in cross-browser applications. The onHelp method applies to the entire document, so it is used in the BODY tag:

```
<body onHelp="openHelp();return false">
```

Normally, Internet Explorer will open its Help system when the user presses F1. We are including "return false" to tell Internet Explorer to not perform its default action (opening its Help system). The openHelp function is the same as the openWBT function:

```
function openHelp() {
window.open('helppage.htm',
'helpwin','toolbar=0,location=0,directories=0,status=0,menubar=0,scroll
bars=0,resizable=0,width=200,height=300');
}
```

## OPENING CONTEXT-SENSITIVE HELP

The openHelp() function opens the Help page to its home page rather than a specific Help topic. We can use variables to open the Help to a specific topic depending on the current application topic. We can add a Help link as follows:

```
<a href="javascript:openCSHelp()">open help</a>
```

The openCSHelp() function will detect the current document's filename and open a Help topic with a similar name. For example, if the application page is named "login.htm" the Help page would be named "h_login.htm." The openCSHelp() function would be:

```
function openCSHelp() {
helpurl = location.href;
begin=helpurl.lastIndexOf('/');
begin = begin + 1;
end=helpurl.lastIndexOf('m');
end=end + 1;
helpurl = "h_" + helpurl.substring(begin, end);
window.open(helpurl,'helpwin','toolbar=0,location=0,directories=0,statu
s=0,menubar=0,scrollbars=0,resizable=0,width=300,height=200');
}
```

This function reads the current document's URL (location.href) and assigns it to a variable called "helpurl." Then it removes the "http://" and path and adds "h_" to the beginning. Finally, we use window.open to open a new window that contains the Help topic.

## AUTO-CLOSING POPUPS

You may want your Help windows to automatically close when they lose focus or when the user presses ESC. You can use the onBlur event to see if the window has lost focus and the onKeyPress event to see if the user has pressed the ESC key. Both of these events are used with the BODY tag:

```
<body onBlur="javascript:window.close();" onKeyPress="closeHelp();">
```

Since closing the window is such a short script, we can include it in the BODY tag. Sensing whether the ESC key was pressed takes a few lines of code, so it's better to set up a function. This function can be coded as:

```
<script>
function closeHelp() {
var key = event.keyCode;
if (key == 27) window.close();
}
</script>
```

Every time the user presses a key, the closeHelp script will read the key's ASCII value. The ASCII value of the ESC key is 27, so if the value is 27 the window is closed. You can find a list of ASCII values at www.trainingtools.com/Resources/ASCII_Chart/default.asp.

## ADDING COMMENTS TO SCRIPTS

As technical communicators, we obviously know the value of good documentation. JavaScript programmers can include comments in their code to explain how it works. You can add one-line comments using double slashes:

```
// this is a one-line comment that might explain a variable
```

You can add multi-line comments using slashes and asterisks:

```
/*
this is a multi-line comment
that might introduce a function

*/
```

## RECOMMENDED JAVASCRIPT BOOKS

If you want to learn more about JavaScript, I recommend the following books:

***Designing with JavaScript***
Nick Heinle and Bill Peña

*Designing with JavaScript* provides a good introduction to JavaScript with some useful examples.

***JavaScript Visual Quickstart Guide***
Tom Negrino and Dori Smith

The *JavaScript Visual Quickstart Guide* teaches you how to modify and use some useful example scripts, but it does not really teach you how to write your own scripts.

***JavaScript Bible***
Danny Goodman

The *JavaScript Bible* is a reference book. It discusses every object and outlines how to use its events, methods, and properties. It does provide some introductory chapters, but it's not as approachable as the first two books.

## ABOUT THE AUTHOR

Scott DeLoach is a founding partner of User First Services, an Atlanta-based consulting company that specializes in designing and creating user assistance. Over the last eleven years, Scott has presented over 50 papers on interface design, usability, WBT development, JavaScript coding, and online Help development at conferences across the US and Canada and around the world. He has been developing online Help systems since 1992 and has been coding in JavaScript since 1997. Scott holds a Master's Degree in Technical and Scientific Communication from Miami University.