Understanding JavaScript

JavaScript is not related to Java; JavaScript and JScript are both implementations of a notional standard language called ECMAScript. We explain the origin of these related languages and show how they can be used to enhance Web pages.

Concluding our two-part article.

By Mike Lewis

One of the nice things about developing in JavaScript is that you don't need to buy any special tools: you already have everything you need to get started. You can write the code using a simple text editor, and you use your normal Web browser to test it. You could also write the code using any Web authoring tool which lets you directly edit the HTML; the authoring tool does not need to be specifically JavaScript-enabled.

You write JavaScript code inside a normal HTML page. To separate the code from the rest of the page, you delimit it with <script> and </script> tags. Everything inside those tags is interpreted as script, everything outside as HTML. The script tag includes a Language attribute which identifies the specific scripting language: JavaScript in this case.

The following code displays a simple message in the Web browser:

```
<script language="JavaScript">
document.write("Hello World")
</script>
```

When the browser loads the page it will work through the contents in sequence, from top to bottom, executing the script when it encounters it.

If you want to restrict the code to a specific version of JavaScript, add the version number to the language name. For example, if you specify the language as "JavaScript1.1", the code will be ignored by Navigator 2.0 and earlier and by Internet Explorer 3.0 and earlier. You can also specify "JScript" as the language, in which case it will be ignored by all versions of Navigator and Opera.

In most cases, however, it is better not to specify a version unless you know for sure which browsers will be used. If necessary, your code can detect the browser version and act accordingly.

If the browser supports no scripting language at all, it will simply disregard the <script> tag. Unfortunately, this means that whatever appears between <script> and </script> will be treated as normal HTML text. The actual code - such as the document.write statement in the above example - will therefore be displayed as part of the document, which is not what you want.

You can prevent this by delimiting the script with the following tags: <!— and //—> (see Figure 2, where a more complete version of the Hello World example uses these conventions). HTML treats these tags as comment delimiters and therefore refrains from displaying the text which appears between them. However, the JavaScript interpreter correctly treats the enclosed text as code and executes it as normal. (The double forward slash is not officially part of the end-of-comment tag, but is required in this context by Netscape Navigator.)

If you want to include a comment within JavaScript itself, you have two choices. If the comment occupies a single line, start it with //. To write a multi-line comment, start the comment with /* and end it with */. Figure 2 includes examples of both these styles.

You might want your HTML document to take some special action if the browser does not support JavaScript, or if the user has chosen to disable it. For example, you might show a message which warns that the page will not be displayed properly. You can do this by means of the <noscript> and </noscript> tags. Any HTML appearing between these tags will be displayed if, and only if, scripting is not currently available. For example:

Browser	Supports
Navigator 2.0	JavaScript 1.0
Navigator 3.0	JavaScript 1.1
Navigator 4.0	JavaScript 1.2
Navigator 4.06 and above	JavaScript 1.3
IE 3	JavaScript 1.0, JScript 3.0
IE 4	JavaScript 1.1, JScript 3.1
IE 5	JavaScript 1.2, JScript 5.0
Opera 3.21 and above	JavaScript 1.1

Figure 1 - Browsers which support JavaScript.

```
<noscript>

    This page requires JavaScript.

</noscript>
```

JS Files

Some JavaScript programmers prefer to keep their code separate from the rest of the document. This makes it easier to read the script without being distracted by HTML tags, and it can also help with re-using the code in other documents.

To achieve this, write your JavaScript in a separate text file, and save it with the extension JS. The file should contain JavaScript code only - that is, the text which would otherwise appear between the <script> and </script> tags. To reference the JS file from within your HTML document, use the SRC attribute in the <script> tag. For example, if your JavaScript code is in a file called HELLO.JS, you would reference it as follows:

```
<script language="JavaScript"
SRC="hello.js">
</script>
```

The filename in the SRC attribute can include a directory path or a URL.

The code in the JS file is executed instead of - not in addition to - any code between the <script> and </script> tags. If there is any non-script HTML in the JS file, an error is generated.

```
<html>
<head>
<title>First Example</title>
</head>

<body>
<script language="JavaScript">
<!–
  document.write("Hello World")
  // This is a 1-line comment

  document.write("<br>")
  /*  You can include HTML
      tags in the output. The
      above inserts a line
      break */

  document.write("Greetings")

// –> </script>
</body>
</html>
```

Figure 2 - Hello World example, with comments.

```
currDate = new Date()
TimeNow = currDate.getHours()

if (TimeNow < 12)
  { Greeting = "Good Morning" }
  else if (TimeNow < 18)
    Greeting = "Good Afternoon"
      else Greeting = "Good
      Evening"

document.write(Greeting)
```

Figure 3 - An if/else construct.

Language Fundamentals

JavaScript is not so very different from C, Pascal, Basic, XBase and many similar procedural languages. Any programmer who already knows one of these languages will have no trouble understanding JavaScript's fundamental language elements. For the next part of the article, I will give you an overview of these elements, starting with some key points:

- **White space**: JavaScript is a free-flowing language, in that the code can include arbitrary spaces, tabs and line feeds to improve readability. There is no end-of-line delimiter.
- **Case-sensitivity**: you must be careful to observe the correct case when referring to variables, functions, objects etc. For the most part, JavaScript is case-sensitive.
- **Variable names**: these may contain letters, digits and underscores, and must not start with a digit. (They can also contain dollar signs, but the ECMA standard recommends that these be reserved for use by automatic code generators.)
- **Declarations**: you do not have to declare variables in advance, nor do you need to explicitly specify their data types. A variable is created on the fly the first time that a value is assigned to it, and its data type is determined from the value. What's more, the variable can be re-typed on the fly. For example, A=1 creates a numeric variable; A="Hello" changes the same variable to a string.
- **Data types:** the following types are supported: number (integer and floating-point), boolean (true, false) and string (delimited with either single- or double-quotes).
- **Special values**: the value null is an "empty" value which is not equal to any other value that the variable can hold. The value undefined is similar, but it evaluates to the default value for the appropriate data type.
- **Operators**: JavaScript supports around three dozen operators, including many that will be familiar to C programmers, such as ++ (increments a value by one) and += (performs addition and assigns the result).

Data Conversion

JavaScript performs automatic conversion between its various data types, often in interesting ways. For example, a boolean variable contains one of two values: true or false. But it can legally be used in numeric expressions, in which case true is considered to be equal to 1 and false to 0. In the following expression, a tax amount would be added to the total if the boolean variable Taxable was true, but not if it was false:

```
Total = (Qty * Price) + (Taxable *
TaxAmount)
```

Similarly, if a string happens to contain a number, it too can be used in arithmetic expressions. Thus, the following code is valid:

```
Tax = "17"
Gross = 100 + Tax
Withhold = true * Tax
```

Strings

JavaScript strings can contain C-like escape characters, including \t for tab, \n or \r for line feed or carriage-return (these two have the same effect) and \\ for backslash. But care is needed; HTML normally disregards tabs and line feeds, so to use these characters in displayed text you must

JavaScript

format the text with the HTML <PRE> style.

Strings can also contain HTML formatting tags. As an example, the following code displays three names, each on a separate line, with the middle one in bold:

```
document.write("Tom\n<B>Dick<B>-
\nHarry")
```

Arrays

Unlike simple variables, arrays must be explicitly declared before they can be referenced. You declare an array with the operator new, like this:

```
MyArray = new Array(20)
```

This creates an array called MyArray with 20 elements. The elements start life with no specific data type, and their initial value is the special value, undefined. To reference the elements, use square brackets:

```
MyArray[3] = "Hello"
```

Arrays are always zero-based - that is, the first element's subscript is 0. Array elements can contain any data type, not necessarily the same type for all elements. An element can even contain another array:

```
MyArray = new Array(20)
MyArray[5] = new Array(10)
MyArray[5][0] = "Hi"
```

You are not obliged to specify the length of the array when you create it, and your code can change the length dynamically. In this example, the array will initially be of length zero:

```
Arr1 = new Array()
```

To increase the length, simply assign a value to an element beyond the end of the array:

```
Arr1[9] = "Greetings"
```

Program-Flow Constructs

JavaScript supports a range of branching and looping constructs, similar to those found in other languages. You can use if / else or a switch statement to test conditions. Three constructs are available for executing a block of code repeatedly: for, while, and do / while. The syntax of these constructs closely resembles that of C.

Figure 3 shows a simple if / else construct. Note that I have used braces to enclose the block of statements following the first test. These braces are in fact optional if the block contains just one statement, as is the case here. Note also that, unlike in C, there is no semicolon at the end of the construct (although no harm is done if this is included).

For an example of a for loop, look at Figure 4. Here we are executing a document.write statement six times. Each time round the loop, we generate a message consisting of the name of the next consecutive HTML heading style (using the <Hn> and </Hn> tags). In each case, the message is formatted in the style in question.

```
for (i=1;i<7;i++){
  tag1 = "<H"+i+">"
  tag2 = "</H"+i+">"
  Document.write(tag1+"Heading
  level "+i+tag2)}
```

Figure 4 - A for loop.

```
<html>

<head>
<title>Function call
       example</title>
<script language="JavaScript">
<!–
function ShowHeading(str,lev)
{document.write("<H"+lev+">-
 "+str+"</H"+lev+">")}
// –> </script>
</head>

<body>
<script language="JavaScript">
<!–
for (i=1;i<7;i++)
  ShowHeading("Heading level "-
  +i,i)
// –> </script>
</body>
</html>
```

Figure 5 - A function call.

Functions

As you would expect, JavaScript lets you define functions, which you can subsequently call from elsewhere in your code. Optionally, functions can receive parameters and return a result. The result can be used just like any other value, for instance in an expression.

For example, here is a function which returns the mean average of the two numbers which are passed to it as parameters:

```
function MeanAv(n1,n2)
{ return (n1+n2)/2 }
```

Note the use of braces to enclose the body of the function. These are mandatory, even if the function contains only one statement.

JavaScript programmers usually place their function definitions in the document's head - that is, between the <HEAD> and </HEAD> tags. This is not obligatory, but it does ensure that the function is defined before it is called. This is especially important where event-handlers are used (more of this later). Figure 5 shows an example. This performs the same action as the code in Figure 4, but it uses a separate function to perform the actual output.

Object Orientation

The language elements I have described so far can be used to produce worthwhile JavaScript programs which can greatly enhance your Web pages. However, the real power of JavaScript only becomes apparent when you make use of two further concepts: objects and events.

JavaScript is not a pure object-oriented language. It does not let you create new classes through inheritance, nor does it support the concept of a class hierarchy. However, it does allow you to create new instances of objects and to access their properties and methods. More importantly, the browser environment supplies a range of pre-defined objects which let you interact with Web pages in many useful ways.

We have already used one of these built-in browser objects: the document. The document object embodies the properties and methods of the document currently loaded in the browser window. When you issue a document.write statement, you are in fact calling the write method of the document object.

Other pre-defined browser objects include window, history and navigator. The various elements that can appear in forms (text boxes, radio buttons, submit buttons etc) are also objects. For details of these and other browser objects, see Figure 6.

JavaScript also lets you define entirely new object types and to create instances of objects based on them. There are also several built-in object types (built-in object types are not the same as built-in objects). For an example, look back at Figure 3. The first line in this code creates an object called currDate, which is based on the built-in Date object type.

Although it might not be obvious, JavaScript arrays are also objects. When you declare an array, you are in fact creating a new instance of an object based on the built-in Array object type.

Object Syntax

The basic syntax for working with objects is sometimes called dot notation. This simply means that, when referring to a property or method, you precede its name with that of the object that it belongs to, using a dot as a separator.

For example, to change the current document's background colour, you would assign a value to the document object's bgColor property, like this:

```
document.bgColor = “Blue”
```

Similarly, to display a message box, you would call the window object's alert method:

```
window.alert(“Time for lunch”)
```

The window object is different from the others in that it is assumed to present if no object is specified. In other words, if you omitted the word window and the subsequent dot in the above statement, it would work in just the same way.

Most properties are simple values, such as the bgColor property shown above. However, it is also possible for a property to be an array. What's more, an array can itself hold objects. For example, the document object has a property called links. This is an array, which in turn holds a set of objects, each of which is of type link. Each link object is a reference to an instance of a hyperlink within the document. (Care is needed here: links, in the plural, is the array; link, in the singular, is an individual object within the array.)

The following code iterates the links array and displays the URL associated with each link contained in it:

```
for (i=0;i<document.links.-
length;i++)
document.write(document.links-
[i].href)
```

In the first line of this example, we use the length property of the links array (remember, an array is also an object) to determine how many hyperlinks exist in the document. In the second line, we display each individual link object's href property, which contains the target URL.

Events

To a large extent, JavaScript is an event-driven environment. An event occurs as a result of some action, typically performed by the user. For example, when the user clicks on a button, the button-click event occurs; when a document is loaded in a window, the window-load event occurs; and so forth.

By writing “event handlers” you can make your documents respond to these events. Thus, if you wanted to prompt the user for confirmation before following a link, you could write an event handler for the link's click event. The event handler would contain the code which prompts for confirmation. The code would be executed whenever the user clicked on the link.

The syntax for event-handlers is somewhat different from the code that we have seen so far. Each event is associated with an HTML element. When you write JavaScript code to respond to an event, you do not place it within the <script> and </script> tags. Rather, you make it an attribute of the tag of the element to which it relates.

For example, the following code displays an alert when the user clicks on a link:

```
<p><a href=“www.whatever.com”
OnClick = “alert(‘You are leaving
the page’)”>
</a></p>
```

Object	**Details**
navigator	The browser itself
window	The browser window or a frame within the window
history	The list of URLs recently accessed from the window
document	The currently-loaded HTML document
frame	An HTML frame
link	A hypertext link
image	An image in an HTML document
location	A URL
form	An HTML form
button	A button in a form
checkbox	A checkbox in a form
hidden	A hidden field in a form
password	A password field in a form
radio	A set of radio buttons in a form
reset	A reset button in a form
select	An option list (listbox or combo box) in a form
option	An item within an option list
submit	A submit button in a form
text	A one-line text field in a form
textarea	A scrolling text field within a form
screen	The screen display
embed	An embedded object in a document
applet	A Java applet embedded in an HTML document
mimeType	A mime type supported by the browser
plugin	A plug-in supported by the browser

Figure 6 - Browser objects.

The event we are responding to here is the click event associated with the link. The code which is to be executed when this event happens is stored in the string to the right of the equals sign in the second line. This string is assigned to the OnClick attribute of the link. (The relevant attribute names are always the same as the corresponding event names but preceded by On.)

Note that, although the string contains JavaScript code, the entire construct is HTML, not JavaScript. For that reason, the event attribute name (OnClick in this case) is not case-sensitive.

In the above example, we are executing a single JavaScript statement in response to the event. It is possible to place multiple statements in the string, using a semi-colon as a separator. However, in these cases it is usually more convenient to place the code in a function, and to call the function from the event attribute. We'll see an example of this in a moment.

Form Validation

For a more complete example of event processing, we will return to the scenario I mentioned at the start of the article: validating the contents of an order form for an online shopping site. Let's suppose that the order is subject to a minimum quantity of 100 units. If the user enters a value below that figure, we want to display an alert and reject the value.

A good place to do this check would be in the blur event of the order quantity field. As its name might suggest, this event occurs when the field loses focus.

Figure 7 shows how the code might look (for clarity, I have excluded details which are not relevant to the example). As you can see, a separate function, called ValidateQty(), performs the validation and displays the alert. If the check fails, this function also calls the field's focus() and select() methods to ensure that focus remains in the field. If it did not do this, the user could simply ignore the error.

Note that the validation function treats a value of zero as being valid. This allows the user to skip the order quantity and return to it later. To prevent a zero order quantity from being sent to the server, the field should be re-checked when the user submits the form (that is, in the form's submit event). For that matter, all checks of this type should also be repeated by the server just in case the user does not have JavaScript enabled.

```
<html>
<head>
<script language="JavaScript">
<!—
function ValidateQty(fld)
{ if (fld.value !=0 &&
  fld.value < 100)
  { alert("Minimum order is
    100 units")
  fld.focus()
  fld.select()  }
 }
// —> </script>
</head>

<body>
  <p><input type"=TEXT"
name="Qty" size="20"
  OnBlur = "Vali-
dateQty(Qty)"></p>
</form>
</body>
</html>
```

Figure 7 - Form validation example.

Component Scripting

Although JavaScript and Java are quite separate, they can interact with each other in several useful ways. Specifically, you can use JavaScript to programmatically access the variables (properties) and methods of a Java applet, and you can similarly use Java to interface with JavaScript objects. It is also possible for JavaScript to interact with ActiveX controls.

These features can do a great deal to enhance the functionality and appearance of your JavaScript applications. Java applets and ActiveX controls can take you well beyond what JavaScript on its own can achieve. For example, there are applets and controls which can supply sophisticated user interface elements, such as pop-up calendars, expandable trees, progress bars and Windows Explorer-style views.

As an example, consider a Java applet which displays various types of charts. Let's suppose that the applet has a class called Charts; this in turn has a method called BarChart(). To access the applet from your JavaScript code, you first incorporate it into the page using the standard HTML applet tag:

```
<applet code="Charts.class"
name="mychart"
width=100 height=50> </applet>
```

Once this has been done, JavaScript can access the applet's variables and methods through the browser's built-in applet object. This object, which is a property of the document object, lets you reference specific applets by means of their class names. So the code to call the Chart applet's BarChart() method would look like this:

```
document.Chart.BarChart()
```

For this to work, the applet's class name, and the names of the relevant variables and methods, must be declared public. This is simply a matter of adding the keyword public to the appropriate declarations in the Java code.

ActiveX controls are slightly more difficult to handle. Essentially, you use the <object> tag to insert the control into the document and to specify the values of its various attributes. These include the control's CLSID (a unique code that identifies every COM component) and the initial values of its properties. The control's events are declared by means of additional attributes to the <script> tag.

In practice, you can avoid much of the coding associated with ActiveX controls by using a suitable HTML authoring tool. In FrontPage 2000, for instance, you can insert the control via the Insert menu, and adjust its properties from a dialog which you reach by right-clicking on the control in the editing window. These actions will automatically generate the <object> tag and its various attributes.

Server-Side Scripts

So far, this article has concentrated

on client-side JavaScript. The main difference in server-side scripting is that the scripts are compiled in advance, and left to execute in the background, waiting for a request from a client. The language and syntax are almost the same, but server-side scripts have access to a different range of built-in objects. There are also some differences in the associated HTML tags.

When writing server-side JavaScript, you need to know what type of server it will run on. In a Netscape environment, the script requires a FastTrack or Enterprise server, and is supported by a Netscape technology called LiveWire. Among other things, LiveWire provides a JavaScript compiler which converts the script to a bytecode file (with the extension WEB). Another LiveWire component, called Application Manager, is then used to activate the compiled code. The script actually runs when the user opens its URL in the browser.

LiveWire also provides a way for JavaScript code to store permanent information on the server. By using the File object, the script can perform low-level input and output to serial files. Alternatively, you can use the database object to access back-end databases via ODBC. These tools make it possible to write full-blown server-side applications which collect data from the user, execute queries, generate reports, and quite a lot more.

The Microsoft equivalent of LiveWire is Active Server Pages (ASP). This runs on Microsoft's Internet Information Server (IIS) and Personal Web Server. It provides a run-time engine which interprets server-side scripts (held in files with the extension ASP) and returns HTML pages to the browser. ASP is not specific to JavaScript; it also supports VBScript and Perl.

"In a Netscape environment, the script requires a FastTrack or Enterprise server, and is supported by a Netscape technology called LiveWire."

Like LiveWire, ASP provides objects for performing low-level file access on the server and for accessing back-end databases via ODBC.

To finish, here is a simple example of a server-side script. These three lines of LiveWire code display the user's IP address in the browser window:

```
<SERVER>
write("Your IP address is "+-
request.ip)
</SERVER>
```

Note the use of the <SERVER> and SERVER> tags in place of <SCRIPT> and </SCRIPT>. The request object provides details of the requesting browser. The ASP equivalent of this code would not include the <SERVER> tags, but would instead have a Language declaration at the top of the document. The syntax for using the Request object (note there is a capital R in the Microsoft version) would also be slightly different.

Conclusion

This article has presented a brief introduction to what is undoubtedly a very substantial and capable language. I have described the basic concepts, but there are many details which I do not have space to cover.

To learn JavaScript properly, you will want to find out more about the built-in objects and the event model, and you will probably want to delve further into server-side scripting as well. Fortunately, there are many books and online resources available to help you do that. My aim has been to encourage you to get started with this very useful technology.

PCSA

JavaScript Resources

If you are interested in going further with JavaScript, you can download a full set of Netscape reference manuals and tutorials from **developer.netscape.com/docs/manuals/**. For the complete JScript documentation, go to **msdn.microsoft.com/scripting** and follow the download link. Both sites also have useful FAQs.

The official ECMA 262 standard is available from **www.ecma.ch**. Note that this is a formal specification and is not intended as a learning aid.

There are no significant third-party sites devoted exclusively to JavaScript. However, Gamelan (**www.gamelan.com**) and Java World (**www.javaworld.com**) are two Java-oriented sites which also provide useful JavaScript content.

The Author

Mike Lewis is a freelance technical journalist and a regular contributor to PCSA. You can contact him by email at mike.lewis@itp-journals.com.

New Reviews from Tech Support Alert

Anti-Trojan Software Reviews
A detailed review of six of the best anti trojan software programs. Two products were impressive with a clear gap between these and other contenders in their ability to detect and remove dangerous modern trojans.

Inkjet Printer Cartridge Suppliers
Everyone gets inundated by hundreds of ads for inkjet printer cartridges, all claiming to be the cheapest or best. But which vendor do you believe? Our editors decided to put them to the test by anonymously buying printer cartridges and testing them in our office inkjet printers. Many suppliers disappointed but we came up with several web sites that offer good quality cheap inkjet cartridges with impressive customer service.

Windows Backup Software
In this review we looked at 18 different backup software products for home or SOHO use. In the end we could only recommend six though only two were good enough to get our “Editor’s Choice” award

The 46 Best Freeware Programs
There are many free utilities that perform as well or better than expensive commercial products. Our Editor Ian Richards picks out his selection of the very best freeware programs and he comes up with some real gems.