Introduction to JavaScript

by Marshall Brain

Introduction

The Netscape Browser includes a scripting language called JavaScript that makes certain tasks in HTML much easier. JavaScript follows in the footsteps of other scripting languages like PERL, providing an interpreted text programming language that is easy to use and fairly rich. In many cases you can accomplish simple tasks in JavaScript that would take much more effort to accomplish in Java. It is therefore useful to know both Java and JavaScript so that you can appropriately use one or the other.

Netscape provides on-line JavaScript documentation that is extremely good. If you read through the entire collection of material once (a task that takes about two hours), you will know just about everything there is to know about JavaScript. Rather than duplicate that effort, here are four simple examples that demonstrate how you might use JavaScript yourself.

Example 1

The first example demonstrates that it is possible to create functions in JavaScript that automate certain HTML tasks. By using custom-designed functions you can shorten or simplify HTML documents that contain repetitive elements. The following code demonstrates these sorts of functions:

```
<HTML>
<HEAD>
<TITLE>HTML document for the World Wide Web</TITLE>
<SCRIPT>
<!--- hide script from old browsers
function outputH(text, level)
{
   document.write("<H" + level + ">" + text + "</H" + level + ">")
}

function outputP(text)
{
   document.write("<P>" + text)
}
// end hiding from old browsers -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
<!--- hide script from old browsers
  outputH("This is big", 3)
  outputH("This is bigger", 2)
  outputH("This is biggest", 1)
  outputP("This isn't big")
// end hiding from old browsers -->
</SCRIPT>
</BODY>
</HTML>
```

To see this script in action, click here.

There are several important things to notice in this simple script:

1. In the <head> section there is a <script> tag that contains two functions named outputP and outputH. These functions are not intended to be useful - they simply demonstrate the process.
2. The <body> section also contains a <script> tag that calls the two functions.
3. Note that both pieces of script code are held within comments. These comments protect the code so that it is not displayed in pre-JavaScript browsers, but do not affect the JavaScript interpreter.
4. Note that the functions have parameters but you do not have to worry about parameter types. In general, JavaScript will make all typing and type conversion decisions for you and you do not have to worry about them.
5. In the body, you can see calls to the functions.
6. JavaScript contains objects (document is an object, for example), but does not allow you to create new ones. The objects have methods that you call like functions (see the on-line JavaScript documentation for lists of functions and objects). In this case, the write method of the document object is called to send HTML output into the document. HTML code is sequential, like output from a teletype, and whenever you write something to the document it is appended to what is already there and interpreted immediately.

Example 2

The second example demonstrates how you can call a function in response to an event in a control.

```
<HTML>
<HEAD>
<TITLE> JavaScript Test </TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide this script tag's contents from old browsers
function myFunction(form)
{
        if (form.data.value == "")
                alert ("You typed nothing");
        else
                alert ("You typed: " + form.data.value);
}
<!-- done hiding from old browsers -->
</SCRIPT>
</HEAD>
<BODY>
<P>
Enter text in the field, remove focus and watch it appear in a mesage box.
<FORM method=POST >
<TEXTAREA NAME="data" ROWS=5 COLS=50 onChange=myFunction(this.form)>
Enter your data here.
</TEXTAREA>
</FORM>
</BODY>
</HTML>
```

This script demonstrates that you can call functions in response to events. In this case, when the text area generates an onChange event, the function gets called. It in turn produces a dialog box. The contents of the dialog depends on the contents of the text area (be sure the text area contains absolutely nothing to see the "nothing" dialog). Note that the onChange function is not triggered until you remove focus from the text area. Since there is only one control in this document, you must click somewhere else in the document to change the focus.

Example 3

The third example demonstrates that you can create simple applications in JavaScript. Here the program creates an *extremely* simple application able to multiply two numbers together, but you can easily imagine more complicated scripts:

```
<HTML>
<HEAD>
<TITLE> The JavaScript Interest Calculator</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide this script tag's contents from old browsers

function computeProduct(form)
{
        if ((form.one.value == "") || (form.two.value == ""))
        {
                return;
```

```
        }
        form.product.value = form.one.value * form. two.value;
}

function clearForm(form)
{
        form.one.value = "1";
        form.two.value = "2";
        form.product.value = "Product";
}

<!-- done hiding from old browsers -->
</SCRIPT>
</HEAD>

<BODY>
Enter two numbers to find their product.
<FORM method=POST>
<INPUT NAME=one VALUE=1 onChange=computeProduct(this.form)> *
<INPUT NAME=two VALUE=2 onChange=computeProduct(this.form)> =
<INPUT NAME=product VALUE=Product  onChange=computeProduct(this.form)>
<BR>
<INPUT NAME=clear VALUE=Reset TYPE=BUTTON onClick=clearForm(this.form)>
<INPUT NAME=calc VALUE=Calculate
    TYPE=BUTTON onClick=computeProduct(this.form)>
</FORM>
</BODY>
</HTML>
```

In this program, three editable areas and two buttons create the "application". The two buttons each call functions that perform obvious tasks. The edit controls also call functions when they change.

Example 4

The following example demonstrates how to use the setTimeout feature and the status method of the window class to create a scrolling banner on the status line.

```
<html>
<head>
<SCRIPT LANGUAGE="JavaScript">
<!-- Beginning of JavaScript Applet -------------------

function scroller(distance)
{
        var msg  = "Welcome to the On-Line Training Center! Please
           sign our guest book if you have a minute...";
        var out = " ";
        var cmd = "";
        var i = 0;
        if (distance < 0)
        {
          distance = 300;
          cmd="scroller(" + distance + ")";
          window.setTimeout(cmd,100);
        }
        else if (distance > msg.length)
        {
          out = "";
          for (i=0 ; i < distance - msg.length; i++)
          {
                  out += " ";
          }
          out += msg;
          distance--;
          cmd="scroller(" + distance + ")";
          window.status=out;
          window.setTimeout(cmd,100);
        }
        else
        {
          out = msg.substring(msg.length-distance, msg.length);
          distance--;
          cmd="scroller(" + distance + ")";
```

```
          window.status=out;
          window.setTimeout(cmd,100);
        }
}
// -- End of JavaScript code -------------- -->
</SCRIPT>
</head>
<BODY onLoad="window.setTimeout('scroller(300)',500);" >
<h1>This is a test</h1>
</body>
</html>
```

The code creates a function called scroller. This function is first called by body's the onLoad event when the page loads. The scroller function simply creates a string and sends it to window.status (the status line for the browser window). The message initially consists of a long line of spaces, which are reduced each time the function loops. Eventually the actual message scrolls into view and moves across the status area.

The line of spaces in this code has to be long enough to cover the " widest possible window". In the code that width is set in the onLoad event and in the first if statement. You could change the code to display the whole message left justified for a second or two, and then scroll it off to the left or right. That would get you out of the business of having to guess the widest possible window.

Conclusion

The goal here has been to demonstrate how to create simple JavaScript applications. Obviously your applications can become as complicated as you like, and there are several interesting examples in the JavaScript area of Yahoo that you may find interesting. In addition, JavaScript can let you do several things that are quite unexpected. See, for example, the clock example in the on-line JavaScript documentation (use the index to find the setTimeout function and look for example 2).

This is big

This is bigger

This is biggest

This isn't big