Using JavaScript in a compatible way

Rafael Palacios*

*Universidad Pontificia Comillas, Madrid, Spain

Abstract

Many web pages use JavaScript to enhance appearance and to perform some client–side operations such us Form verification. Today the majority of browsers are JavaScript–capable, consequently many system administrators are disregard of the fact that their pages cannot be used properly if JavaScript is not supported or disabled. Browsers without JavaScript support are rare, but the number of users that disable JavaScript is quite significant. Another issue is that sometimes security is relaxed assuming that form fields have been verified by a client–side data validation function. Allowing access to non–JavaScript browsers increases accessibility and encourages programmers to pay more attention to security threats. This paper describes some programming tricks to ensure compatibility in web page design.

1 Introduction

JavaScript is a programming language developed by Netscape in 1995 to add dynamism and more graphical possibilities to existing web pages. It is not a compiled language, source code is embedded inside HTML code that defines a page. HTML is used to define the overall appearance of the page and JavaScript is used to define functions that respond to certain signals such as mouse actions or time events. For similarities in syntax, JavaScript took its name after the Java programming language. Java was developed by Sun Microsystems and became very popular after Netscape announced that it was going to incorporate Java Technology into its browser, in march 1995.

Since JavaScript was officially announced on December 4, 1995; most Internet browsers have adopted this script language as a standard and today it is widely supported. Nevertheless many Internet users have opted to disable this option in their browsers, because JavaScript is often use as an advertising tool and because of apprehension to execute downloaded code. Advertising is sometimes annoying, specially when ads are shown sequentially, associating the close event in a window to the creation of another window with the following advertisment. In addition, some people may have disabled JavaScript inadvertently by selecting the highest level of security in section "Internet Options" of Microsoft's Browser. According to INT Media Group statistics [1], that analyzes an average of 700,000 visitors/hour, 99% of today's Internet browsers can support JavaScript, but it is enabled only in 87%. The number of users accepting JavaScript code had been increasing since the appearance of Netscape Navigator 2 and Internet Explorer 3 (first browsers to support JavaScript 1.0), but the percentage stabilized at 87% since May 2001.

Some web sites are economically supported by advertising companies and will not be able to offer any content without their sustenance. If their advertisements require JavaScript to be shown, then non–JavaScript browsers may not be allowed. But in most cases, and specially in intranets, web pages must be developed to be fully compatible. In this paper it is shown how to develop Forms, image effects and text menus, compatible to all browsers.

2 Form Validation

One of the most important applications of JavaScript is data validation in forms. When the user presses the Submit button, a JavaScript function reads and verifies all the fields in the form before sending any data to the server. If some error is found, the data is not sent and the user is encouraged to correct the error. This kind of data validation is very convenient, it reduces server load and network traffic. It is faster because users get error messages immediately , instead of waiting for server–side validation and networks delays. Nevertheless, JavaScript form validation can not replace server–side verification, otherwise server security will be compromised. Server–side validation of inputs is always necessary as the program in charge of processing the data (typically a Common Gateway Interface CGI) may not work properly. The most common errors are due to values out of range or failure to meet a required length and format, that may yield to a buffer overflow. Anyone can write his own page with a Form tag defined to send any kind of information to any server, acting as a replacement of the original form page. Therefore it is very easy to attack any form processing program with data in wrong format.

Since JavaScript data validation only ensures correct data for harmless users, it must be considered as an additional work to help users. This code, however, should never replace the same functional code in the program in charge of processing the data on the server. Moreover, valid ranges and lengths of each field can be seen in the JavaScript code, making it evident to construct malicious content to be sent to the server.

Web page developers should focus on the server–side to provide data checking and processing. Almost any kind of programming language can be used for this task, from conventional high level programming languages to script languages like PERL, or web oriented languages such as PHP, ASP.NET or JSP [3]. In addition to the server–side code, web developers may also provide some JavaScript code for data validation to reduce server load, network traffic, and annoying delays for capable browsers. In some pages I have seen the call to the verification function associated to the submit button of the form, so if there are no errors the information is sent using `form.submit()`. The code for the button is:

```
<input type="button" value="Submit Data" onClick="verify()">
```

This kind of code will never work on non-JavaScript browser, because the parameter `onClick` will be ignored so clicking on the button will not do anything. The correct way to add data validation to forms is through the `onSubmit` directive of the form tag, as described in [2]. In addition, a good trick is to include a warning if JavaScript is not supported.

```
<html>
<head>
  <title>Example 1</title>
  <script language="JavaScript">
  <!-- Hide code if JavaScript is not supported
  function verify(fm)
  {
      var msg="";
      var obj;

      //Check all the fields first
      for(var i = 0; i < fm.length; i++) {
          obj = fm.elements[i];
          if (obj.required)  {
             if ((obj.value == null) || (obj.value == "")) {
                 msg += "The field '" + obj.name + "' is empty.\n";
             }
          }
      }

      //Print all the error together
      if (msg=="") return true;  //returning true will send the data to the server
      alert(
         " E R R O R, your data was not submitted\n" +
         "-----------------------------------------------\n\n\n" +
         msg +
         "\nPlease correct the error and re-submit.\n");
      return false;
  }
  //End of hidden code -->
  </script>
</head>
<body>
  <form name="form2"
     method="post"
     action="/cgi-bin/server-program"
     onSubmit="
       this.field1.required=true;
       this.field2.required=true;
       this.field3.required=false;
       return verify(this);">
------------  text and <input> tags  ---------------
  <input type="submit" value="Submit Data">
  <NOSCRIPT> Warning, this browser is not JavaScript-capable. Data will be verified at the server. </NOSCRIPT>
  </form>
</body>
</html>
```

In the previous example, `verify` function is very simple and only checks if the required fields are empty, but it shows the recommended basic structure for form verification:

- JavaScript code is defined in the header, hidden to old browsers.
- The call to `verify` is defined as `onSubmit` in the form tag
- A good practice is to define parameter or field ranges also inside the form tag, but they can also be defined in the input tags
- The submit button does not need additional definitions such as `onClick`.
- The text within `<NOSCRIPT>` and `</NOSCRIPT>` is only shown in browser where JavaScript is not supported or has been cancelled.

Since JavaScript treats the `<!--` sequence as a single-line comment and ignores the `-->` sequence, another way to show messages on non-JavaScript browsers is the following [2]:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- --> Your web browser does not support JavaScript.
<!-- Hide code if JavaScript is not supported
        .
        . JavaScript code goes here
        .
//End of hidden code -->
</SCRIPT>
```

The latter way to show HTML text on non-JavaScript browsers requires more code, but it is more backward compatible. To be more precise, the text hidden by `<NOSCRIPT>` tag will be shown in Netscape 2.0 that is a JavaScript1.0–enabled browser. Despite this problem of incompatibility, specific to Netscape 2.0, the `<NOSCRIPT>` tags are preferred for its compact syntax.

3 Forms With Just One Field

For some pages it is very useful to provide a way to make a selection within a list of options in a pull–down menu. This kind of selection object requires just one line of space, as compared to one line per option in the classical HTML way using `<a>` tags. The way to implement the list of items is using a form with a select object of size equal to one line. Although a submit button is usually provided in forms, in the case of form with just one field it is more efficient to process the request as soon as the person makes the change. This can be done in JavaScript associating the submit function to the event `onChange` of the select object, so the submit button is only needed if JavaScript is disabled.

```
<form action=newlanguage.cgi method="POST">
<SELECT name="lang" size="1" onChange="form.submit()">
<OPTION VALUE=1>English
<OPTION VALUE=2>Español
<OPTION VALUE=3>Français
<OPTION VALUE=4>Deutsch
</SELECT>
<NOSCRIPT>
<input type="submit" value="GO">
</NOSCRIPT>
</form>
```

4 Image Changes

One of the most attractive features of JavaScript powered web pages is the ability to change images or make button effects as the mouse moves around the page. Images are usually changed as a consequence of `onMouseOver` and `onMouseOut` events, which are properties of the anchor object defined inside `<a>` tags. In HTML code, an image is included using `<IMG>` tag, that has a property called `SRC` to define the source of the image to be shown. JavaScript is allowed to read or change `SRC` property of any image object. For compatibility with non-JavaScript, it is important to define the image and the link in pure HTML way, and then add extra definitions for JavaScript–enabled browsers.

```
<html>
<head>
  <title>Images Example</title>
  <script language="JavaScript">
  <!-- Hide code if JavaScript is not supported
  function ChangeImage(image_obj,image_src)
  {
     image_obj.src=image_src;
  }
  //End of hidden code -->
  </script>
</head>
<body>
  <h1>Main Menu</h1>
  <a href="page1.html"
     onMouseOver='ChangeImage(img01,"red_ball.gif")'
     onMouseOut ='ChangeImage(img01,"gray_ball.gif")'>
     <img SRC="gray_ball.gif" BORDER=0 name="img01">
     Go to page 1</a><br>

  <a href="page2.html"
     onMouseOver='ChangeImage(img02,"red_ball.gif")'
     onMouseOut ='ChangeImage(img02,"gray_ball.gif")'>
     <img SRC="gray_ball.gif" BORDER=0 name="img02">
     Go to page 2</a><br>
</body>
</html>
```

In the previous code all the definition are together (links address, initial image, onMouseOver image, and onMouseOut image) therefore the possibility making mistakes is diminished. Non–JavaScript browsers will ignore the properties `onMouseOver` and `onMouseOut;` they will only show the initial images defined inside `IMG` tag, but at least one image is shown and the link works. Although the code is easy and clear, it may produce unwanted delays when the cursor is moved over a link for the first time. The delay is due to the loading of the alternative image defined for `onMouseOver`, and can be annoying if the connection is slow or if the images are large. The only images loaded at the same time as the HTML page are those defined in `IMG` tags, all other images are loaded as needed. After loading an image, there are no additional delays using it because it is stored into the browser's cache. To force an image to be cached, it is possible to create an `Image` object in JavaScript. One way to do it is defining a `PreloadImages` function that can be associated to `onLoad` event of the `<body>` tag. Here is an example:

```
function PreloadImages()
{
   img01on =new Image();     img01on.src="red_ball.gif";
   img01off=new Image();     img01off.src="gray_ball.gif";
}
```

To call this function from the body tag, one has to write: `<body onLoad="PreloadImages()">`

After doing these definitions both images are stored in the cache, so all image changes are faster. It is also possible to make calls to the same `ChangeImage` but using the `Image` object just created, instead of using the filename:

```
<a href="page1.html"
   onMouseOver="ChangeImage(img01,img01on.src)"
   onMouseOut ="ChangeImage(img01,img01off.src)">
   <img SRC="gray_ball.gif" BORDER=0 name="img01">
   Go to page 1</a><br>
```

Nevertheless, the former way to call `ChangeImage` is also valid as the browsers are smart enough to detect that the filename in the string is the same as the one cached in `PreloadImages`. Therefore it could be easy to use any variable name in `PreloadImages`, and then filenames in the call to `ChangeImage`.

5 Text Menus

This section describes how to create dynamic menus in text mode. There is a main menu in the web page and several submenus that are shown according to the position of the mouse over the main menu. Both kinds of menus are defined using tables consisting on one column. The sizes of the cells in the main menu are hardcoded, making it possible to know the correct coordinates where the submenu tables should appear. The main menu table is defined inside a `<DIV>` tag, that allows determining the position of table. Each submenu is also defined inside `<DIV>` tags, in order to specify the coordinates of its origin and to make it initially invisible. The entries of the main menu are links to other pages, these links are mainly used by non–Javascript browsers. In JavaScript mode, when the cursor is over one of these links the value of the `visibility` attribute of the corresponding submenu is changed from "hidden" to "visible", so it suddenly appears at the correct coordinates.

All the dynamism of the submenus is defined using `onMouseOver` property of the `<a>` tag. It is important to pay attention to the objects that should become visible or invisible on every event. In the code presented here, the submenu does not disappear unless another submenu is shown, so the user can easily make a selection in the submenu. To hide all submenus, two white images have been added to the first and last rows of the main menu. These images provide the expected behavior when the cursor leaves the main menu without making any selection: cleaning of submenus and recovery of whatever information was located under the submenu area. Similar objects could be added to submenu tables, but most people find the selection process very sensitive to mouse movements and at the end more difficult.

The menu tables are drawn as transparent objects over the current page, and the background is recovered when the submenu is hidden. The parameter `bgcolor` in `<table>` tag can be initialized to provide a background color to the menus. In fact, `white.gif` in the code is an image in GIF format made of one pixel of transparent color, so even the first and last rows of the main menu will show the background of the page (if `bgcolor` is not defined) or the background of the table.

In order to have access to `DIV` parameters, it is necessary to take into account the different implementations of this kind of objects in Netscape4, Netscape6 and Internet Explorer. Up to this section all the code was compatible with all browsers, but the access to `DIV` objects is not standard. Bill Pena proposed a set of very useful JavaScript functions to find objects in different browsers [4]. The code to identify the version of the browser and these functions are located inside the header of the page. Once these functions have been declared, object access is as simple as using variables:

```
getMyStyle('submenu02').visibility='hidden';
```

The code for a web page that contains one menu with two submenus is presented below:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
```

```
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
   <meta name="Author" content="Rafael Palacios">
   <title>Text Menus</title>

<script  language="JavaScript">
<!-- Hide code if JavaScript is not supported
/*
getMyStyle functions will return the object of a DIV tag.
This function is different for Internet Explorer, Netscape 4 and Netscape 6
Here is a variation of the method proposed by Bill Pena
http://www.oreillynet.com/pub/a/javascript/2002/01/23/crossbrowser_style.html?page=2
*/

//document.write('appName = ' + navigator.appName + '<br>');
//document.write('appVersion = ' + navigator.appVersion + '<br>');
//document.write('appVersion.substring(0,1) = ' + navigator.appVersion.substring(0,1) + '<br>');
//document.write('appCodeName = ' + navigator.appCodeName + '<br>');

if (navigator.appName == 'Netscape' && navigator.appVersion.substring(0,1) < 5) {
   document.getMyStyle = nav4GetStyle;
} else  {
   document.getMyStyle = nav5GetStyle;
}

function nav5GetStyle(id)  {
  var obj = document.getElementById(id)
  return obj.style;
}

function nav4GetStyle(id) {
  return nav4FindLayer(this, id);
}

function nav4FindLayer(doc, id) {
  var i;
  var subdoc;
  var obj;
  for (i = 0; i < doc.layers.length; ++i) {
    if (doc.layers[i].id && id == doc.layers[i].id)
    return doc.layers[i];
    subdoc = doc.layers[i].document;
    obj = nav4FindLayer(subdoc, id);
    if (obj != null) {
      return obj;
    }
  }
  return null;
}
//End of hidden code -->
</script>
</head>


<body bgcolor="#FFFFFF" text="#000000">

Hi, This is an example of text menus by Rafael Palacios

<div id='mainmenu' style="position:absolute; left:10; top:50;">
<table width="100" border="0" cellpadding="0" cellspacing="0" bgcolor=#FFFFFF>
  <tr><td align="right" height="20"><!-- First cell used to delete all submenus -->
  <a href="" OnMouseOver=
     "getMyStyle('submenu01').visibility='hidden';
      getMyStyle('submenu02').visibility='hidden';">
  <img src="white.gif" width="100" height="10" border="0"></a>
  </td></tr>

  <tr><td align="right" height="20">
  <a href="option1.html" OnMouseOver="
      getMyStyle('submenu01').visibility='visible';
      getMyStyle('submenu02').visibility='hidden';">
  option 1</a>
  </td></tr>

  <tr><td align="right" height="20">
  <a href="option2.html" OnMouseOver="
      getMyStyle('submenu01').visibility='hidden';
      getMyStyle('submenu02').visibility='visible';">
  option 2</a>
  </td></tr>

  <tr><td align="right" height="20"><!-- Last cell used to delete all submenus -->
  <a href="" OnMouseOver=
     "getMyStyle('submenu01').visibility='hidden';
      getMyStyle('submenu02').visibility='hidden';">
  <img src="white.gif" width="100" height="10" border="0"></a>
  </td></tr>
</table>
</div>

<!-- SUBMENUS -->
<div name="submenu01" id="submenu01" style="position:absolute; left:112; top:70; visibility:hidden">
<table width="100" border="1" cellpadding="0" cellspacing="0">
  <tr><td>
  <a href="option1-1.html">option 1.1</a>
  </td></tr>

  <tr><td>
  <a href="option1-2.html">option 1.2</a>
  </td></tr>
</table>
</div>

<div name="submenu02" id="submenu02" style="position:absolute; left:112; top:90; visibility:hidden">
<table width="100" border="1" cellpadding="0" cellspacing="0">
  <tr><td>
  <a href="option2-1.html">option 2.1</a>
  </td></tr>

  <tr><td>
  <a href="option2-2.html">option 2.2</a>
  </td></tr>

  <tr><td>
  <a href="option2-3.html">option 2.3</a>
  </td></tr>
</table>
</div>

</body>
</html>
```

Another difference in web browsers is that Netscape4 ignores `<DIV>` tags if JavaScript is disabled, therefore submenus appear at the end of the web page. In Netscape6 or Internet Explorer, visibility attribute is taken into account even if JavaScript is disabled. One easy workaround is to define a non–JavaScript file that only contains general HTML code that describes the page and the main menu, but without submenus. The header of this page must include the following code that replaces the current page (my_home_page.html) by the JavaScript version shown above:

```
<script  language="JavaScript">
<!-- Hide code if JavaScript is not supported
location.replace("my_home_page_js.html");
//End of hidden code -->
</script>
```

It is also possible to create menus in graphic mode, using images and the properties `onMouseOver` and `onMouseOut` as described in the previous section. Text written in images can show color changes and 3D effects, adding more dynamism and improving the overall appearance. In this case each menu could be built as a column of images or as a big image divided into active regions defined in a `<map>` tag. Graphic menus require more computer power to redraw images and better bandwidth to transfer all the images efficiently. The only concern, in term of compatibility with non–Javascript web browsers, is to define the images of the main menu as described above and to keep submenus hidden.

6 References

[1] The Counter Statistics. http://www.thecounter.com/stats. March 2002.

[2] David Flannagan, JavaScript: The Definitive Guide. Ed. O'Reilly and Associates, ISBN: 1-56592-235-2, 1997.

[3] Brian Jepson, "From Script to Database" in PC Magazine – Ziff Davis Media, Vol 20, No 21, December 11, 2001.

[4] Bill Pena, "O'Reilly Network: Cross-Browser Style Objects", http://www.oreillynet.com/pub/a/javascript /2002/01/23/crossbrowser_style.html, April 2002.