Part 8

Differences between Browsers

Browsers vary considerably in:

- The HTML tags they recognise, and the way they respond to certain HTML tags.
- The version of the Document Object Model (DOM) they support.
- The versions of JavaScript (and other scripting languages) they support.

Therefore, creating web pages that will display correctly on all (or even most) browsers is not easy, particularly if the pages contain dynamic material.

However, use of a scripting language such as JavaScript allows us to create pages that will load and run correctly on a wide range of browsers.

Major differences between browsers include:

- **The mechanism used to address elements**

  Using Dynamic HTML it is possible to modify various attributes of elements - usually style attributes such as size, position, visibility, etc. - using scripts. For this to work, there has to be a means of uniquely identifying each element within a web-page so that information can be sent to it or retrieved from it.

  Some elements - such as forms and their components - are represented in the object hierarchy and can be addressed directly. However, not all elements are represented in the object hierarchy, and it was not considered practical to introduce new objects for each type of element. Therefore, browser manufacturers introduced other mechanisms for addressing elements:

  - **Microsoft** added the `all` object to Internet Explorer. This is an array of all the elements of a web-page, allowing the individual elements to be addressed in the following way:

    ```
    document.all.myElement.style.attribute
    ```

  - **Netscape** added a mechanism whereby any element could be addressed directly, provided it had been given a name. This allows elements to be addressed in the following way:

    ```
    document.myElement.style.attribute
    ```

  - **The World-Wide Web Consortium** (W3C) subsequently agreed a standard (part of the DOM Level 1 Standard) whereby elements are assigned a recognised name and henceforth addresed directly by that name. For example:

    ```
    var myElement = document.getElementByID("myElementName")

    myElement.style.attribute
    ```

This standard has been adopted in newer browsers (e.g., Netscape 6), but earlier browsers use one of the proprietary systems described above.

- **The use of Divisions and Layers**

  The `<div>` tag is recognised by both Internet Explorer and Netscape Navigator, but they offer different levels of support for it:

  - In Internet Explorer 4 & 5 and Netscape Navigator 6, divisions are dynamic (i.e., they can be repositioned and/or have their size, visibility, etc., modified).
  - In Netscape Navigator 4, divisions can only be static.

  The `<layer>` tag is specific to Netscape Navigator and was supported on versions of Netscape Navigator prior to version 6. On these browsers it offers similar functionality to that available using divisions in Internet Explorer, including dynamic-repositioning, etc..

- **Event handling**

  - In Internet Explorer, events propagate *up* the object hierarchy.

    For example, if the mouse is clicked whilst over a form button, the click event is received by the button object and then passed on to objects further up the hierarchy, such as the Document object.

  - In Netscape Navigator, events propagate *down* the object hierarchy.

    For example, if the mouse is clicked whilst over a form button, the click event is received by the Document object and passed down to the button object.

    (In Netscape Navigator, it is possible to 'capture' events at the Document level and thus prevent them being sent down the hierarchy to form elements such as buttons; this can be used, for example, to disable all the elements in a form until such time as they are required.)

Browsers also differ in the support they provide for the JavaScript language. The level of compatibility offered by the various versions of Netscape and Internet Explorer (for JavaScript and Jscript respectively) is summarised below:

Navigator version	JavaScript version	Comments
2.0	1.0	First version of JavaScript. Included some bugs/limitations, e.g.:

		* `window.open()` method doesn't work on Mac and Unix platforms. * Poor support for arrays. * `onLoad()` event triggered by re-sizing.
3.0	1.1	Support for `<src>` attribute, allowing use of external JavaScript source code.
4.0 - 4.05	1.2	Support for dynamic positioning of elements. Signed scripts for security.
4.06 - 4.5	1.3	ECMAScript 1.0 Compatible
5.0	1.4	ECMAScript 2.0 Compatible More advanced error-checking and control functions.

Explorer version	JScript version	Comments
3.0	1.0	Used a separate *JScript Scripting Engine* which could be upgraded separately from the browser. Many versions in use. Particular problems when handling image attributes.
4.0 - 4.5	3.0	ECMAScript 1.0 Compatible (and therefore largely identical to JavaScript 1.3). Support for `<src>` attribute (but not until v3.02, and even then with bugs), allowing use of external JavaScript source code. Support for dynamic positioning of elements.
5.0	5.0	ECMAScript 2.0 Compatible (and therefore largely identical to JavaScript 1.4)

In addition to the differences listed above, there are many minor and not-so-minor differences in the way the various browsers suppport and interpret HTML and JavaScript. Most of these will be listed in any good HTML or JavaScript reference book.

Obtaining Browser Parameters

We can tackle differences between browsers using the *Navigator* object

This object returns properties related to the browser, such as its type, version, etc.. It also returns properties indicating the platform on which the browser is running (Macintosh, Windows PC, Linux PC, etc.),

Navigator object properties include:

`appName` Returns a string representing the browser. For example, the code:

```
alert(navigator.appName);
```

should return a string containing the name of the browser you are using (e.g., "Microsoft Internet Explorer" or "Netscape")

Click here to see this piece of code in operation.
Get Navigator Name

`appCodeName` Returns a string representing the code name of the browser. For example, the code:

```
alert(navigator.appCodeName);
```

should return a string containing the code name of the browser you are using (e.g., "Mozilla")

Click here to see this piece of code in operation.
Get Navigator Code Name

`appVersion` Returns a string representing the browser version. For example, the code:

```
alert(navigator.appVersion);
```

should return a string indicating which version of the browser you are using (e.g., "4.0 (compatible: MSIE 5.0)")

Click here to see this piece of code in operation.
Get Navigator Version

`platform` Returns a string representing the computer platform on which the browser is running. For example, the code:

```
alert(navigator.platform);
```

should return a string containing the name of the platform on which the browser is running (e.g., "Win32" or "MacPPC")

Click here to see this piece of code in operation.
Get Platform Type

Using some or all of the Navigator object properties, It is possible to determine which browser is being used and then generate the appropriate code.

Click here to see a simple example. Show Example

This example includes a movable bar that has to be coded differently depending upon which browser it is being viewed under. The web-page tests to see what browser is being used, then generates appropriate code. The code used is quite simple and may not accommodate all browsers, but it should work under most widely-used browsers.

Have a look at the source code of the example to see how it works. Note the following points:

- The page contains a script in the `<head>` section. The script declares two global variables and two functions. The purpose of these is discussed below.
- The body of the page contains several sections:
  - A piece of JavaScript code that uses the `navigator` object to determine what browser is being used. If the browser is found to be Netscape Navigator, a further test is performed to find out what version is being used. This is done using the `charAt()` method to obtain the first character of the string (which is always the browser version number). Once obtained, this information is stored in one of the global variables, `browserType`.
  - This is followed by another piece of JavaScript code that uses `document.write()` methods to create a new page. The content of the page varies depending upon the browser used (as determined by the previous piece of code): if the browser is Netscape Navigator 4, the page content is created using a `<layer>` tag; if the browser is Internet Explorer or Netscape Navigator 6, the page content is created using a `<div>` tag.
  - Following this is another piece of code that creates a form with two buttons. This code works equally well with all the target browsers, so it can be written directly to the document without checking the browser type.
  - Finally, there is a piece of code that checks the browser type again and writes either a closing </layer> tag or a closing `</div>` tag to the page.

  Because these pieces of code are placed in the body of the document they will be executed automatically when the document is loaded.

- Having created the page, the division or layer can be moved around using the two functions in the `<head>` section of the document. These interrogate the global variable `browserType` to find out what browser is in use, then use one of the following techniques to move the division or layer:
    - If the code is being viewed under Internet Explorer, the *division* can be moved by altering the value of the `pixelLeft` attribute in its style.

      In order to locate the division, the code uses the `document.all` array. Thus the `pixelLeft` attribute of the division called `myDivision` can be identified using the following code:

      ```
      document.all.myDivision.style.pixelLeft
      ```

    - If the code is being viewed under Netscape Navigator 4, the *layer* can be moved by altering the value of its `pageX` attribute.

      Netscape Navigator does not support the `document.all` array used in Internet Explorer. However, `<layer>` is represented as an object in the Document Object Model. Therefore the value of the `pageX` attribute of the layer called `myLayer` can be read or written simply by specifiying it in terms of its place within the object hierarchy, e.g.:

      ```
      document.mylayer.pageX
      ```

    - If the code is being viewed under Netscape Navigator 6, the *division* can be moved by altering the value of the `left` attribute in its style.

      However, unlike Internet Explorer, Netscape Navigator 6 does not support the `document.all` array, nor does it provide a special object for the `<div>` tag as earlier versions did for the `<layer>` tag.

      Instead, Netscape Navigator 6 uses the `getElementByID()` method, in accordance with the current World-Wide Web Consortium (W3C) standard. The `getElementByID()` method is called and given the name of the division as a parameter, and the result is stored in a variable. Once this has been done, the variable can be used to address the division and its style attributes, e.g.:

      ```
      var myDiv = document.getElementByID("myDivision");

      ...

      mydiv.style.left
      ```

      One slight complication with Netscape Navigator 6 is that the `left` style attribute is stored with the suffix 'px' (meaning pixels). This is also in accordance with the current W3C standard. Before any arithmetical operations can be performed on the `left` attribute, the suffix must be removed. This is done using the `parseInt()` method, which extracts integers from strings.