VOODOO'S INTRODUCTION TO JAVASCRIPT

Part 4: Windows and on-the-fly documents

Creating windows

Opening new browser windows is a great feature of JavaScript. You can either load a new document (for example a HTML-document) to the new window or you can create new documents (on-the-fly). We will first have a look at how we can open a new window, load a HTML-page to this window and then close it again. The following script opens a new browser window and loads a meaningless page:

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function openWin() {
  myWin= open("bla.htm");
}

// -->
</script>
</head>
<body>

<form>
<input type="button" value="Open new window" onClick="openWin()">
</form>

</body>
</html>
```

(The online version lets you test this script immediately)

The page *bla.htm* is loaded into the new window through the *open()* method.

You can control the appearance of the new window. For example you can decide if the window shall have a statusbar, a toolbar or a menubar. Besides that you can specify the size of the window. The following script opens a new window which has got the size 400x300. The window does not have a statusbar, toolbar or menubar.

```
<html>
<head>
```

```
<script language="JavaScript">
<!-- hide

function openWin2() {
  myWin= open("bla.htm", "displayWindow",
    "width=400,height=300,status=no,toolbar=no,menubar=no");
}

// -->
</script>
</head>
<body>

<form>
<input type="button" value="Open new window" onClick="openWin2()">
</form>

</body>
</html>
```

(The online version lets you test this script immediately)

You can see that we specify the properties in the string "width=400,height=300,status=no,toolbar=no,menubar=no". Please note that you must not use spaces inside this string!

Here is a list of the properties a window can have:

directories	yes|no
height	*number of pixels*
location	yes|no
menubar	yes|no
resizable	yes|no
scrollbars	yes|no
status	yes|no
toolbar	yes|no
width	*number of pixels*

Some properties have been added with JavaScript 1.2 (i.e. Netscape Navigator 4.0). You cannot use these properties in Netscape 2.x or 3.x or Microsoft Internet Explorer 3.x as these browsers do not understand JavaScript 1.2. Here are the new properties:

alwaysLowered	yes|no
alwaysRaised	yes|no
dependent	yes|no
hotkeys	yes|no
innerWidth	*number of pixels* (replaces width)

innerHeight	*number of pixels* (replaces height)
outerWidth	*number of pixels*
outerHeight	*number of pixels*
screenX	*position in pixels*
screenY	*position in pixels*
titlebar	yes|no
z-lock	yes|no

You can find an explanation of these properties in the JavaScript 1.2 guide. I will have an explanation and some examples in the future.
With the help of these properties you can now define at which position a window shall open. You cannot do this with the older versions of JavaScript.

The name of a window

As you have seen we have used three arguments for opening a window:

```
myWin= open("bla.htm", "displayWindow",
  "width=400,height=300,status=no,toolbar=no,menubar=no");
```

What is the second argument for? This is the name of the window. We have seen how to use the target-property earlier. If you know the name of an existing window you can load a new page to it with

```
<a href="bla.html" target="displayWindow">
```

Here you need the name of the window (if the window does not exist, a new window is created through this code). Please note that *myWin* is not the name of the window. You can just access the window through this variable. As this is a normal variable it is only valid inside the script in which it is defined. The window name (here displayWindow) is a unique name which can be used by all existing browser windows.

Closing windows

You can close windows through JavaScript. For this you need the close() method. Let's open a new window as shown before. In this window we load the following page:

```
<html>
<script language="JavaScript">
<!-- hide

function closeIt() {
  close();
}

// -->
</script>
```

```
<center>
<form>
<input type=button value="Close it" onClick="closeIt()">
</form>
</center>

</html>
```

(The online version lets you test this script immediately)

If you hit the button in the new window the window is being closed. *open()* and *close()* are methods of the window-object. Normally we should think that we have to write *window.open()* and *window.close()* instead of *open()* and *close()*. This is true - but the window-object is an exception here. You do not have to write window if you want to call a method of the window-object (this is only true for this object).

Creating documents on-the-fly

We are coming now to a cool feature of JavaScript - creating documents on-the-fly. This means you can let your JavaScript code create a new HTML-page. Furthermore you can create other documents - like VRML-scenes etc.. You can output these documents in a separate window or in a frame.
First we will create a simple HTML-document which will be displayed in a new window. Here is the script we are going to have a look at now.

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function openWin3() {
  myWin= open("", "displayWindow",
    "width=500,height=400,status=yes,toolbar=yes,menubar=yes");

  // open document for further output
  myWin.document.open();

  // create document
  myWin.document.write("<html><head><title>On-the-fly");
  myWin.document.write("</title></head><body>");
  myWin.document.write("<center><font size=+3>");
  myWin.document.write("This HTML-document has been created ");
  myWin.document.write("with the help of JavaScript!");
  myWin.document.write("</font></center>");
  myWin.document.write("</body></html>");

  // close the document - (not the window!)
  myWin.document.close();
}
```

```
// -->
</script>
</head>
<body>

<form>
<input type=button value="On-the-fly" onClick="openWin3()">
</form>

</body>
</html>
```

(The online version lets you test this script immediately)

Let's have a look at the function *winOpen3()*. You can see that we open a new browser window first. As you can see the first argument is an empty string "" - this means we do not specify an URL. The browser should not just fetch an existing document - JavaScript shall create a new document.
We define the variable *myWin*. With the help of this variable we can access the new window. Please note that we cannot use the name of the window (*displayWindow*) for this task. After opening the window we have to open the document. This is done through:

```
// open document for further output
myWin.document.open();
```

We call the *open()* method of the document-object - this is a different method than the *open()* method of the window-object! This command does not open a new window - it prepares the document for further output. We have to put *myWin* before the *document.open()* in order to access the new window.
The following lines create the document with *document.write()*:

```
// create document
myWin.document.write("<html><head><title>On-the-fly");
myWin.document.write("</title></head><body>");
myWin.document.write("<center><font size=+3>");
myWin.document.write("This HTML-document has been created ");
myWin.document.write("with the help of JavaScript!");
myWin.document.write("</font></center>");
myWin.document.write("</body></html>");
```

You can see that we write normal HTML-tags to the document. We create HTML-code! You can write any HTML-tags here.
After the output we have to close the document again. The following code does this:

```
// close the document - (not the window!)
myWin.document.close();
```

As I told you before you can create documents on-the-fly and display them in a frame as well. If you for example have got two frames with the names *frame1* and *frame2* and want create a new document in *frame2* you can write the following in *frame1*:

```
parent.frame2.document.open();

parent.frame2.document.write("Here goes your HTML-code");

parent.frame2.document.close();
```

Creating VRML-scenes on-the-fly

In order to demonstrate the flexibility of JavaScript we are now going to create a VRML-scene on-the-fly. VRML stands for Vitual Reality Modelling Language. This is a language for creating 3D scenes. So get your 3D glasses and enjoy the ride... No, it's just a simple example - a blue cube. You will need a VRML plug-in in order to view this example. This script doesn't check if a VRML plug-in is available (this is no problem to implement).

(The online version lets you test this script immediately)

Here is the source code:

```
<html>
<head>
<script language="JavaScript">
<!-- hide

function vrmlScene() {
  vrml= open("", "displayWindow",
    "width=500,height=400,status=yes,toolbar=yes,menubar=yes");

  // open document for further output
  vrml.document.open("x-world/x-vrml");

  vr= vrml.document;

  // create VRML-scene
  vr.writeln("#VRML V1.0 ascii");

  // Light
  vr.write("Separator { DirectionalLight { ");
  vr.write("direction 3 -1 -2.5 } ");

  // Camera
  vr.write("PerspectiveCamera { position -8.6 2.1 5.6 ");
  vr.write("orientation -0.1352 -0.9831 -0.1233 1.1417 ");
  vr.write("focalDistance 10.84 } ");

  // Cube
  vr.write("Separator { Material { diffuseColor 0 0 1 } ");
  vr.write("Transform { translation -2.4 .2 1 rotation 0 0.5 1 .9 } ");
  vr.write("Cube {} } }");

  // close the document - (not the window!)
```

```
  vrml.document.close();
}

// -->
</script>
</head>
<body>

<form>
<input type=button value="VRML on-the-fly" onClick="vrmlScene()">
</form>

</body>
</html>
```

This source code is quite similar to the last example. First we open a new window. Then we have to open the document in order to prepare it for the output. Look at this code:

```
  // open document for further output
  vrml.document.open("x-world/x-vrml");
```

In the last example we did not write anything into the brackets. What does the *"x-world/x-vrml"* mean? It's the MIME-type of the file we want to create. So here we tell the browser what kind of data follows. If we do not write anything into the brackets the MIME-type is set to *"text/html"* by default (this is the MIME-type of HTML-files).
(There are different ways for getting to know a certain MIME-type - the browser itself has a list of the known MIME-types. You can find this list in the option or preference menu.)
We have to write *vrml.document.write()* for creating the 3D scene. This is quite long - therefore we define *vr=vrml.document.* Now we can write *vr.write()* instead of *vrml.document.write().*
Now we can output normal VRML-code. I am not going to describe the elements of a VRML-scene. There are several good VRML sources available on the Internet. The plain VRML-code looks like this:

```
#VRML V1.0 ascii

Separator {

  DirectionalLight { direction 3 -1 -2.5 }

  PerspectiveCamera {
    position -8.6 2.1 5.6
    orientation -0.1352 -0.9831 -0.1233 1.1417
    focalDistance 10.84
  }

  Separator {
    Material {
      diffuseColor 0 0 1
    }
    Transform {
      translation -2.4 .2 1
```

```
    rotation 0 0.5 1 .9
   }
   Cube {}
  }
}
```

This is the code which we output through the *document.write()* commands.
Of course it is quite meaningless to create a scene on-the-fly which can also be loaded as a normal VRML-file.
It gets more interesting if you for example make a form where the user can enter different objects - like a spehre, cylinder, cone etc. - and JavaScript creates a scene from this data (I have an example of this in my JS-book).

e-mail:skoch@rumms.uni-mannheim.de
http://rummelplatz.uni-mannheim.de/~skoch/
My JavaScript-book: http://www.dpunkt.de/javascript