

VOODOO'S INTRODUCTION TO JAVASCRIPT

© 1996, 1997 by Stefan Koch

Part 5: Statusbar and timeouts

The statusbar

Your JavaScript programs can write to the statusbar - this is the bar at the bottom of your browser window. All you have to do is to assign a string to `window.status`. The following example shows you two buttons which can be used to write to the statusbar and to erase the text again.

(The online version lets you test this script immediately)

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

function statbar(txt) {
    window.status = txt;
}

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

<form>
  <input type="button" name="look" value="Write!"
    onClick="statbar('Hi! This is the statusbar!');">
  <input type="button" name="erase" value="Erase!"
    onClick="statbar('');">
</form>

</body>
</html>
```

We create a form with two buttons. Both buttons call the function `statbar()`. You can see that the function call created by the *Write!* button looks like this:

```
statbar('Hi! This is the statusbar!');
```

Inside the brackets we specify the string *'Hi! This is the statusbar!'* This means this string is passed along to the function `statbar()`. You can see that we defined the function `statbar()` like this:

```
function statbar(txt) {
    window.status = txt;
}
```

What is new is that we use *txt* inside the brackets of the function name. This means the string we passed along to the function is stored in the variable *txt*. Passing variables to functions is an often used way for making functions more flexible. You can pass several values to functions - you just have to separate them through commas. The string *txt* is displayed on the statusbar through *window.status = txt*. Erasing the text on the statusbar is achieved through assigning an empty string to *window.status*.

Displaying text on the statusbar can easily be used in combination with links. Instead of showing the URL of the link you can explain in words what the next page is about. This link demonstrates this - just move your mousepointer over the link. The code for this example looks like this:

```
<a href="dontclck.htm"
  onMouseOver="window.status='Don\'t click me!'; return true;"
  onMouseOut="window.status=";">link</a>
```

Here we are using *onMouseOver* and *onMouseOut* in order to detect when the mousepointer moves across the link. You might wonder why we have to write *return true* inside the *onMouseOver* property. This means that the browser won't execute its own code as a reaction to the *MouseOver* event. Normally the browser displays the URL of the link in the statusbar. If we do not use *return true* the browser will write to the statusbar immediately after our code has been executed - this means it would overwrite our text and the user couldn't read it. In general we can suppress the following actions of the browser by using *return true* in the event-handler.

onMouseOut did not exist in JavaScript 1.0. If you are using the Netscape Navigator 2.x you might get different results on different platforms. On Unix platforms for example the text disappears even though the browser does not know *onMouseOut*. On Windows the text does not disappear. If you want your script to be compatible to Netscape 2.x on Windows you might for example write a function which writes text to the statusbar and erases this text after a certain period of time. This is programmed with a timeout. We will learn more about timeouts in the following paragraph.

In this script you can see another thing - sometimes you want to output quotes. We want to output the text *Don't click me* - as we specify this string inside the *onMouseOver* event-handler we are using the single quotes. But the word *Don't* uses a single quote as well! The browser gets mixed up if you just write *'Don't ...'*. To solve this problem you can just write a backslash \ before the quote - which means that it belongs to the output (you can do the same with double quotes ").

Timeouts

With the help of timeouts (or timer) you can let the computer execute some code after a certain period of time. The following script shows a button which opens up a popup window after 3 seconds.

The script looks like this:

```
<script language="JavaScript">
```

```

<!-- hide

function timer() {
    setTimeout("alert('Time is up!')", 3000);
}

// -->
</script>

...

<form>
    <input type="button" value="Timer" onClick="timer()">
</form>

```

`setTimeout()` is a method of the window-object. It sets a timeout - I think you might have guessed that. The first argument is the JavaScript code which shall be executed after a certain time. In our case this argument is `"alert('Time is up!')"`. Please note that the JavaScript code has to be inside quotes. The second argument tells the computer when the code shall be executed. You have to specify the time in number of milliseconds (3000 milliseconds = 3 seconds).

Scroller

Now that you know how to write to the statusbar and how timeouts work we will have a look at scrollers. You might already know the moving text-strings in the statusbar. They can be seen all over the Internet. We will see how to program a basic scroller. Besides that we will think of possible improvements of the scroller. Scrollers are quite easy to implement. Just let us think about how we could realize a moving text in the statusbar. We have to write a text to the statusbar. After a short period of time we have to write the same text to the statusbar - but we have to move it a little bit to the left side. If we repeat this over and over again the user gets the impression of a moving text. We have to think about how we can determine which part of the text should be displayed as the whole text is normally longer than the statusbar.

(The online version lets you test this script immediately)

Here is the source code - I have added some comments:

```

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

// define the text of the scroller
var scrtxt = "This is JavaScript! " +
    "This is JavaScript! " +
    "This is JavaScript!";
var length = scrtxt.length;
var width = 100;
var pos = -(width + 2);

```

```

function scroll() {

    // display the text at the right position and set a timeout

    // move the position one step further
    pos++;

    // calculate the text which shall be displayed
    var scroller = "";
    if (pos == length) {
        pos = -(width + 2);
    }

    // if the text hasn't reached the left side yet we have to
    // add some spaces - otherwise we have to cut of the first
    // part of the text (which moved already across the left border
    if (pos < 0) {
        for (var i = 1; i <= Math.abs(pos); i++) {
            scroller = scroller + " ";
        }
        scroller = scroller + sctxt.substring(0, width - i + 1);
    }
    else {
        scroller = scroller + sctxt.substring(pos, width + pos);
    }

    // assign the text to the statusbar
    window.status = scroller;

    // call this function again after 100 milliseconds
    setTimeout("scroll()", 100);
}

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

<body onLoad="scroll()">
Your HTML-page goes here.
</body>
</html>

```

The main part of the *scroll()* function is needed for calculating which part of the text is being displayed. I am not explaining the code in detail - you just have to understand how this scroller works in general. In order to start the scroller we are using the *onLoad* event-handler of the *<body>* tag. This means the function *scroll()* will be called right after the HTML-page has been loaded. We call the *scroll()* function with the *onLoad* property. The first step of the scroller is being calculated and displayed. At the end of the *scroll()* function we set a timeout. This causes the *scroll()* function to be executed again after 100 milliseconds. The text is moved one step forward and another timeout is set. This goes on for ever. (There have been some problems with this kind of scroller with Netscape Navigator 2.x. It so-

metimes caused an 'Out of memory'-error. I've got many mails explaining this is because of the recursive call of the scroll() function - finally leading to a memory overflow. But this is not true. This is not a recursive function call! We get recursion if we call the scroll() function inside the scroll() function itself. But this isn't what we are doing here. The old function which sets the timeout is finished before the new function is executed. The problem was that strings could not really be changed in JavaScript. If you tried to do it JavaScript simply created a new object - without removing the old one. This is what filled up the memory.)

Scrollers are used widely in the Internet. There is the risk that they get unpopular quickly. I must admit that I do not like them too much. Especially annoying on most pages is that the URL cannot be read anymore when moving the pointer across a link. This can be solved through stopping the scroller when a MouseOver event occurs - you can start it again with onMouseOut. If you want to have a scroller try not to use the standard scroller - try to add some nice effect. Maybe one part of the text moving from left and the other part is coming from right - when they meet in the middle the text stands still for some seconds. With some phantasy you can certainly find some nice alternatives (I have some examples in my book).

©1996,1997 by Stefan Koch

e-mail:skoch@rumms.uni-mannheim.de

<http://rummelplatz.uni-mannheim.de/~skoch/>

My JavaScript-book: <http://www.dpunkt.de/javascript>