VOODOO'S INTRODUCTION TO JAVASCRIPT

Part 6: Predefined objects

The Date-object

JavaScript lets you use some predefined objects. This is for example the Date-object, the Array-object or the Math-object. There are several other objects - please refer to the documentation provided by Netscape for a complete reference.
We are going to have a look at the Date-object first. As the name implies this object lets you work with time and date. For example you can easily calculate how many days are left until next christmas. Or you can add the actual time to your HTML-document.
So let's begin with an example which displays the actual time. First we have to create a new Date-object. For this purpose we are using the new operator. Look at this line of code:

today= new Date()

This creates a new Date-object called today. If you do not specify a certain date and time when creating a new Date-object the actual date and time is used. This means after executing *today= new Date()* the new Date-object *today* represents the date and time of this specific moment.
The Date-object offers some methods which can now be used with our object today. This is for example *getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth()* and so on. You can find a complete reference of the Date-object and its methods in Netscapes JavaScript documentation.
Please note that a Date-object does only represent a certain date and time. It is not like a clock which changes the time every second or millisecond automatically.
In order to get another date and time we can use another constructor (this is the *Date()* method which is called through the *new* operator when constructing a new Date-object):

today= new Date(1997, 0, 1, 17, 35, 23)

This will create a Date-object which represents the first of january 1997 at 17:35 and 23 seconds. So you specify the date and time like this:

Date(year, month, day, hours, minutes, seconds)

Please note that you have to use 0 for january - and not 1 as you might think. 1 stands for february and so on.

Now we will write a script which outputs the actual date and time. The result will look like this:

Time: 17:53
Date: 4/3/2010

The code looks like this:

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

now= new Date();

document.write("Time: " + now.getHours() + ":" + now.getMinutes() + "<br>");
document.write("Date: " + (now.getMonth() + 1) + "/" + now.getDate() + "/" +
        (1900 + now.getYear()));

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

Here we use methods like *getHours()* in order to display the time and date specified in out Date-object now. You can see that we are adding 1900 to the year. The method *getYear()* returns the number of years since 1900. This means if the year is 1997 it will return 97 if the year is 2010 it will return 110 - not 10! If we add 1900 we won't have the year 2000 problem. Remember that we have to increment the number we receive through *getMonth()* by one.
This script does not check whether the number of minutes is less than 10. This means you can get a time which looks like this: *14:3* which actually means *14:03*. We will see in the next script how to solve this problem.

Now we will have a look at a script which displays a working clock:

```
<html>
<head>

<script Language="JavaScript">
<!-- hide

var timeStr, dateStr;

function clock() {
    now= new Date();

    // time
    hours= now.getHours();
    minutes= now.getMinutes();
    seconds= now.getSeconds();
    timeStr= "" + hours;
    timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
    timeStr+= ((seconds < 10) ? ":0" : ":") + seconds;
    document.clock.time.value = timeStr;

    // date
    date= now.getDate();
    month= now.getMonth()+1;
    year= now.getYear();
    dateStr= "" + month;
    dateStr+= ((date < 10) ? "/0" : "/") + date;
```

```
    dateStr+= "/" + year;
    document.clock.date.value = dateStr;

    Timer= setTimeout("clock()",1000);
}

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

<body onLoad="clock()">

<form name="clock">
  Time:
  <input type="text" name="time" size="8"  value=""><br>
  Date:
  <input type="text" name="date" size="8" value="">
</form>

</body>
</html>
```

(The online version lets you test this script immediately)

We use the *setTimeout()* method for setting the time and date every second. So we create every second a new Date-object with the actual time. You can see that the function *clock()* is called with the *onLoad* event-handler in the <*body*> tag. In the body-part of our HTML-page we have two text-elements. The function *clock()* writes the time and date into these two form-elements in the right format. You can see that we are using two strings *timeStr* and *dateStr* for this purpose.
We have mentioned earlier that there is a problem with minutes less than 10 - this script solves this problem through this line of code:

```
timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
```

Here the number of minutes are added to the string *timeStr*. If the minutes are less than 10 we have to add a 0. This line of code might look a little bit strange to you. You could also write it like this which might look more familar:

```
if (minutes < 10) timeStr+= ":0" + minutes
  else timeStr+= ":" + minutes;
```

The Array-object

Arrays are very important. Just think of an example where you want to store 100 different names. How could you do this with JavaScript? Well, you could define 100 variables and assign the different names to them. This is quite complicated.
Arrays can be seen as many variables bundled together. You can access them through one name and a number.
Let's say out array is called *names*. Then we can access the first name through *names[0]*. The

second name is called *name[1]* and so on.
Since JavaScript 1.1 (Netscape Navigator 3.0) you can use the Array-object. You can create a new array through *myArray= new Array()*. Now you can assign values to this array:

```
myArray[0]= 17;
myArray[1]= "Stefan";
myArray[2]= "Koch";
```

JavaScript arrays are really flexible. You do not have to bother about the size of the array - its size is being set dynamically. If you write *myArray[99]= "xyz"* the size of the array get 100 elements (a JavaScript array can only grow - it hasn't got the ability to shrink. So keep your arrays as small as possible.). It doesn't matter if you store numbers, strings or other objects in an array. I haven't mentioned every detail of arrays here but I hope you will see that arrays are a very important concept.
Certainly many things get clearer by looking at an example. The output of the following example is:

first element
second element
third element

Here is the source code:

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

myArray= new Array();

myArray[0]= "first element";
myArray[1]= "second element";
myArray[2]= "third element";

for (var i= 0; i< 3; i++) {
  document.write(myArray[i] + "<br>");
}

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

First we are creating a new array called *myArray*. Then we assign three different values to the array. After this we start a loop. This loop executes the command *document.write(myArray[i] + "
");* three times. The variable *i* counts from 0 to 2 with this for-loop. You can see that we are using *myArray[i]* inside the for-loop. As *i* counts from 0 to 2 we get three *document.write()* calls. We could rewrite the loop as:

```
document.write(myArray[0] + "<br>");
document.write(myArray[1] + "<br>");
document.write(myArray[2] + "<br>");
```

Arrays with JavaScript 1.0

As the Array-object does not exist in JavaScript 1.0 (Netscape Navigator 2.x and Microsoft Internet Explorer 3.x) we have to think of an alternative. This piece of code could be found in the Netscape documentation:

```
function initArray() {
  this.length = initArray.arguments.length
  for (var i = 0; i < this.length; i++)
    this[i+1] = initArray.arguments[i]
}
```

You can now create an array with:

```
myArray= new initArray(17, 3, 5);
```

The numbers inside the brackets are the values the array is being initialized with (this can also be done with the Array-object from JavaScript 1.1). Please note that this kind of array does not implement all elements the Array-object from JavaScript 1.1 has (there is for example a sort() method which lets you sort all elements in a specific).

The Math-object

If you need to do mathematical calculations you will find some methods in the Math-object which might help you further. There is for example a sine-method *sin()*. You will find a complete reference in the Netscape documentation. I want to demonstrate the *random()* method. If you have read the first version of this tutorial you know that there have been some problems with the *random()* method. We wrote a function which let us create random numbers. We don't need that anymore as the *random()* method now works on all platforms.
If you call *Math.random()* you will receive a random number between 0 and 1. Here is one possible output of *document.write(Math.random())*:

```
.7184317731538611
```


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