

# VOODOO'S INTRODUCTION TO JAVASCRIPT

© 1996, 1997 by Stefan Koch

## Part 11: JavaScript 1.2 event model

### New events

Time to have a look at one of the new features of the Netscape Navigator 4.x: the event model of JavaScript 1.2. The examples shown here will only work in Netscape Navigator 4.x (most examples will also work in preview releases).

The following events are supported in JavaScript 1.2 (check out Netscape's JS 1.2 documentation if you want to find out more about these events - [http://developer.netscape.com/library/documentation/communicator/jsguide/js1\\_2.htm](http://developer.netscape.com/library/documentation/communicator/jsguide/js1_2.htm)):

Abort	Focus	MouseOut	Submit
Blur	KeyDown	MouseOver	Unload
Click	KeyPress	MouseUp	
Change	KeyUp	Move	
DblClick	Load	Reset	
DragDrop	MouseDown	Resize	
Error	MouseMove	Select	

You can see that some new events have been implemented. We are going to have a look at some of these events during this lesson.

First let's see what the `Resize` event is for. With the help of this event we can detect whenever the window is being resized by the user. The following script demonstrates this:

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

window.onresize= message;

function message() {
  alert("The window has been resized!");
}

</script>
</head>
<body>
Please resize the window.
</body>
</html>
```

With the line

```
window.onresize= message;
```

we define the event handler. This means that the function `message()` is being called as soon as the window is being resized. You might not be familiar with this way of defining event handlers. But this is nothing new in JavaScript 1.2. If you for example have a button object you can define the event handler like this:

```
<form name="myForm">  
<input type="button" name="myButton" onClick="alert('Click event occurred!')">  
</form>
```

But you could also write it like this:

```
<form name="myForm">  
<input type="button" name="myButton">  
</form>
```

...

```
<script language="JavaScript">  
  
document.myForm.myButton.onclick= message;  
  
function message() {  
    alert('Click event occurred!');  
}  
  
</script>
```

You might think that the second alternative is a bit complicated. So why are we using it in the first script? The problem is that the window object isn't defined through a certain tag - so we'll have to use the second possibility.

Two important things: First you must not write `window.onResize` - i.e. you must use lower case. Second you must not write any brackets after `message`. If you write `window.onresize= message()` the browser interprets `message()` as a function call. But in this case we do not want to call the function directly - we just want to define the event handler.

## The Event object

A new Event object has been added to JavaScript 1.2. It contains properties which describe an event. Every time an event occurs an Event object is passed to the event handler.

The following example shows an image. You can click it somewhere. An alert window will come up and display the coordinates of the mouse event.

*(The online version lets you test this script immediately)*

Here is the source code:

```

<layer>
<a href="#" onClick="alert('x: ' + event.x + 'y: ' + event.y); return false;">
</a>
</layer>

```

You can see that we are using the event handler `onClick` inside the `<a>` tag as we would have done with prior JavaScript versions. What is new is that we use `event.x` and `event.y` for creating the output in the alert window. This is the Event object which we need in order to get to know the mouse coordinates of the event.

I have put everything inside a `<layer>` tag. Like this we will get the coordinates relative to this layer, i.e. in our case the image. Otherwise we would get the coordinates relative to the browser window. (`return false;` is used here so that the browser does not follow the link)

The Event object has got the following properties (we will see some of these properties in the next examples):

Property	Description
<i>data</i>	Array of URLs of the dropped objects when a <i>DragDrop</i> event occurs.
<i>layerX</i>	Horizontal position of cursor in pixel relative to layer. In combination with the <i>Resize</i> event this property represents the width of the browser window.
<i>layerY</i>	Vertical position of cursor in pixel relative to layer. In combination with the <i>Resize</i> event this property represents the height of the browser window.
<i>modifiers</i>	String specifying the modifier keys - <i>ALT_MASK</i> , <i>CONTROL_MASK</i> , <i>META_MASK</i> or <i>SHIFT_MASK</i>
<i>pageX</i>	Horizontal position of cursor in pixel relative to browser window.
<i>pageY</i>	Vertical position of cursor in pixel relative to browser window.
<i>screenX</i>	Horizontal position of cursor in pixel relative to screen.
<i>screenY</i>	Vertical position of cursor in pixel relative to screen.
<i>target</i>	String representing the object to which the event was originally sent.
<i>type</i>	String representing event type.
<i>which</i>	ASCII-value of a pressed key or number of mouse button.
<i>x</i>	Synonymous to <i>layerX</i> .
<i>y</i>	Synonymous to <i>layerY</i> .

## Event capturing

One important feature is called event capturing. If someone for example clicks on a button the `onClick` event handler of this button is being called. With the help of event capturing you can achieve that your window, document or layer object captures the event before it is being handled by the button object. Like this your window, document or layer object can handle the event before it reaches its intended target.

Let's have a look at an example in order to see what this is good for:

```

<html>
<head>
<script language="JavaScript">

```

```
window.captureEvents(Event.CLICK);
```

```
window.onclick= handle;
```

```
function handle(e) {  
    alert("The window object captured this event!");  
    return true; // i.e. follow the link  
}
```

```
</script>  
</head>  
<body>  
<a href="test.htm">Click on this link</a>  
</body>  
</html>
```

(The online version lets you test this script immediately)

You can see that we do not define an event handler inside the `<a>` tag. Instead we use

```
window.captureEvents(Event.CLICK);
```

in order to capture the *Click* event through the window object. Normally the window object does not know the *Click* event. But through capturing the event we can redirect it to the window object.

Please note the writing of *Event.CLICK*. *CLICK* has to be in upper case. If you want to capture several events you'll have to separate them through a `|` - for example:

```
window.captureEvents(Event.CLICK | Event.MOVE);
```

You can see that we use *return true;* inside the function *handle()* which we defined as event handling function. This means that the browser is going to follow the link after the *handle()* function is being executed. If you write *return false;* instead, all following actions are being suppressed.

If you define an *onClick* event handler inside the `<a>` tag you'll realize that this event handler isn't called. This is obvious as the window object captures the event before it reaches the link object. If you define the *handle()* function like this

```
function handle(e) {  
    alert("The window object captured this event!");  
    window.routeEvent(e);  
    return true;  
}
```

the computer checks if there are other event handlers defined for this object. The variable *e* is our Event object which is being passed to the event handling function.

You can also send an event directly to a certain object. For this purpose you can use the *handleEvent()* method. This looks like this:

```

<html>
<script language="JavaScript">

window.captureEvents(Event.CLICK);

window.onclick= handle;

function handle(e) {
  document.links[1].handleEvent(e);
}

</script>
<a href="test.htm">Click on this link</a><br>
<a href="test.htm"
  onClick="alert('Event handler of second link!');">Second link</a>
</html>

```

*(The online version lets you test this script immediately)*

All Click events are being sent to the second link - even if you do not click directly on the links!

The following script demonstrates that your script can react to key events. Just push a key in order to see the script in action.

```

<html>
<script language="JavaScript">

window.captureEvents(Event.KEYPRESS);

window.onkeypress= pressed;

function pressed(e) {
  alert("Key pressed! ASCII-value: " + e.which);
}

</script>
</html>

```

©1996,1997 by Stefan Koch  
e-mail:skoch@rumms.uni-mannheim.de  
http://rummelplatz.uni-mannheim.de/~skoch/  
My german JavaScript-book: http://www.dpunkt.de/javascript/