

VOODOO'S INTRODUCTION TO JAVASCRIPT

© 1996, 1997 by Stefan Koch

Part 12: Drag & Drop

What is drag & drop?

With the help of the new event model of JavaScript 1.2 and layers we can implement drag & drop on our web-page. You'll need at least Netscape Navigator 4.0 for this as we use JavaScript 1.2 features.

What is drag & drop? Some operating systems (like Win95/NT or MacOS) let you for example erase files through dropping icons on a trash bin. What you do is you click on the icon of a file, drag (i.e. you hold the mouse button down while moving the mouse) the icon to the trash bin and drop it there.

The drag & drop we want to implement here is restricted to the web-page. So you cannot use this code shown here in order to drag objects inside a HTML-page to your hard disk or something like this. (Since Netscape Navigator 4.0 your script can react to an event called DragDrop when somebody drops a file on your browser window - but this is not what we are going to talk about in this lesson)

(The online version lets you test the example described in this lesson immediately. It shows three boxes which can be moved with the help of the mouse)

You might also want to check out the example provided by Netscape. You can find it at this address: http://home.netscape.com/comprod/products/communicator/user_agent_vacation.html

JavaScript does not support drag & drop directly. This means we cannot just specify a property `draggable` (or whatever) in an image object. We have to write the code for this on our own. You'll see that this isn't too difficult.

So what do we need? We need two things. First we have to register certain mouse events, i.e. how do we know which object shall be moved to which position? Then we need to make up our minds on how we can display the moving objects on the screen. Of course we will use the new layer feature for defining different objects and moving them around on the screen. Every object is represented through its own layer.

Mouse events with JavaScript 1.2

Which mouse events do we have to use? We don't have a `MouseDown` event - but we can achieve the same through the events `MouseDown`, `MouseMove` and `MouseUp`. JavaScript 1.2 uses a new event model. Without this event model we could not solve our task. I have talked about the new event model in the last lesson. But let's have a look at the important parts once again.

The user pushes the mouse button somewhere inside the browser window. Our script has to react on this event and calculate which object (i.e. layer) was hit. We need to know the coordinates

of the mouse event. JavaScript 1.2 implements a new Event object which stores the coordinates of a mouse event (besides other information).

Another important thing is called event capturing. If a user for example clicks on a button the mouse event is sent directly to the button object. But in our case we want the window to handle our event. So we let the window capture the mouse event, i.e. that the window object gets this event and can react upon it. The following example demonstrates this (using the event *Click*). You can click somewhere inside the browser window. An alert window pops up and displays the coordinates of the mouse event.

(The online version lets you test this script immediately)

This is the code for this example:

```
<html>

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

    window.captureEvents(Event.CLICK);

    window.onclick= displayCoords;

    function displayCoords(e) {
        alert("x: " + e.pageX + " y: " + e.pageY);
    }

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

Click somewhere inside the browser window.

```
</html>
```

First we tell the window object to capture the *Click* event. We use the method *captureEvent()* for this. The line

```
    window.onclick= displayCoords;
```

defines what happens when a *Click* event occurs. It tells the browser to call *displayCoords()* as a reaction to a *Click* event (Please note that you must not use brackets behind *displayCoords* in this case). *displayCoords()* is a function which is defined like this:

```
function displayCoords(e) {
    alert("x: " + e.pageX + " y: " + e.pageY);
}
```

You can see that this function takes one argument (we call it *e*). This is the Event object which is being passed to the *displayCoords()* function. The Event object has got the properties *pageX* and *pageY* (besides others) which represent the coordinates of the mouse event. The alert window displays these values.

MouseDown, MouseMove and MouseUp

As I already told you JavaScript does not know a *MouseDown* event. Therefore we have to use the events *MouseDown*, *MouseMove* and *MouseUp* in order to implement drag & drop. The following example demonstrates the use of *MouseMove*. The actual coordinates of the mouse cursor are displayed on the statusbar.

(The online version lets you test this script immediately)

You can see that the code is almost the same as in the last example:

```
<html>

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

    window.captureEvents(Event.MOUSEMOVE);

    window.onmousemove= displayCoords;

    function displayCoords(e) {
        status= "x: " + e.pageX + " y: " + e.pageY;
    }

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

Mouse coordinates are displayed on the statusbar.

```
</html>
```

Please note that you have to write *Event.MOUSEMOVE*, where *MOUSEMOVE* must be in upper case. When defining which function to call when the *MouseMove* event occurs you have to use lower case: *window.onmousemove=...*

Now we can combine the last two examples. We want the coordinates of the mouse pointer to be displayed when the mouse is being moved with pushed mouse button.

(The online version lets you test this script immediately)

The code for this example looks like this:

```
<html>

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

    window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

```

window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;

function startDrag(e) {
  window.captureEvents(Event.MOUSEMOVE);
}

function moveIt(e) {
  // display coordinates
  status= "x: " + e.pageX + " y: " + e.pageY;
}

function endDrag(e) {
  window.releaseEvents(Event.MOUSEMOVE);
}

// -->
</script>

```

Push the mouse button and move the mouse. The coordinates are being displayed on the statusbar.

```
</html>
```

First we tell the window object to capture the events *MouseDown* and *MouseUp*:

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

You can see that we use the sign | (*or*) in order to define several events which shall be captured by the window object. The next two lines define what happens when these events occur:

```

window.onmousedown= startDrag;
window.onmouseup= endDrag;

```

The next line of code defines what happens when the window object gets a *MouseMove* event:

```
window.onmousemove= moveIt;
```

But wait, we didn't define *Event.MOUSEMOVE* in *window.captureEvents()*! This means that this event isn't captured by the window object. So why do we tell the window object to call *moveIt()* although this event never reaches the window object? The answer to this question can be found in the function *startDrag()* which is being called as soon as a *MouseDown* event occurs:

```

function startDrag(e) {
  window.captureEvents(Event.MOUSEMOVE);
}

```

This means the window object captures the *MouseMove* event as soon as the mouse button is

pushed down. We have to stop capturing the *MouseMove* event when the *MouseUp* event occurs. This does the function *endDrag()* with the help of the method *releaseEvents()*:

```
function endDrag(e) {  
    window.releaseEvents(Event.MOUSEMOVE);  
}
```

The function *moveIt()* writes the mouse coordinates to the statusbar.

Now we have all elements for registering the events needed to implement drag & drop. We can move forward to displaying the objects on the screen.

Displaying moving objects

We have seen in previous lessons that we can create moving objects with the help of layers. All we have to do now is to register which object the user clicked on. Then this object has to follow the mouse movements. Here is the code for the example shown at the beginning of this lesson:

```
<html>  
<head>  
  
<script language="JavaScript">  
<!--  
  
var dragObj= new Array();  
var dx, dy;  
  
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);  
  
window.onmousedown= startDrag;  
window.onmouseup= endDrag;  
window.onmousemove= moveIt;  
  
function startDrag(e) {  
    currentObj= whichObj(e);  
    window.captureEvents(Event.MOUSEMOVE);  
}  
  
function moveIt(e) {  
    if (currentObj != null) {  
        dragObj[currentObj].left= e.pageX - dx;  
        dragObj[currentObj].top= e.pageY - dy;  
    }  
}  
  
function endDrag(e) {  
    currentObj= null;  
    window.releaseEvents(Event.MOUSEMOVE);  
}
```

```

function init() {
  // define the 'dragable' layers
  dragObj[0]= document.layers["layer0"];
  dragObj[1]= document.layers["layer1"];
  dragObj[2]= document.layers["layer2"];
}

function whichObj(e) {

  // check which object has been hit

  var hit= null;
  for (var i= 0; i < dragObj.length; i++) {
    if ((dragObj[i].left < e.pageX) &&
        (dragObj[i].left + dragObj[i].clip.width > e.pageX) &&
        (dragObj[i].top < e.pageY) &&
        (dragObj[i].top + dragObj[i].clip.height > e.pageY)) {
      hit= i;
      dx= e.pageX- dragObj[i].left;
      dy= e.pageY- dragObj[i].top;
      break;
    }
  }
  return hit;
}

```

```

// -->
</script>
</head>
<body onLoad="init()">

<layer name="layer0" left=100 top=200 clip="100,100" bgcolor="#0000ff">
<font size=+1>Object 0</font>
</layer>

<layer name="layer1" left=300 top=200 clip="100,100" bgcolor="#00ff00">
<font size=+1>Object 1</font>
</layer>

<layer name="layer2" left=500 top=200 clip="100,100" bgcolor="#ff0000">
<font size=+1>Object 2</font>
</layer>

</body>
</html>

```

You can see that we define three layers in the `<body>` part of this HTML-page. After the whole page is loaded the function `init()` is called through the `onLoad` event handler in the `<body>` tag:

```

function init() {

```

```
// define the 'dragable' layers
dragObj[0]= document.layers["layer0"];
dragObj[1]= document.layers["layer1"];
dragObj[2]= document.layers["layer2"];
}
```

The *dragObj* array takes all layers which can be moved by the user. Every layer gets a number in the *dragObj* array. We will need this number later on.

You can see that we use the same code as shown above in order to capture the mouse events:

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

```
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
```

I have added the following line to the *startDrag()* function:

```
currentObj= whichObj(e);
```

The function *whichObj()* determines which object the user clicked on. It returns the number of the layer. If no layer has been hit it returns the value *null*. The variable *currentObj* stores this value. This means that *currentObj* represents the number of the layer which is being moved at the moment (or it is *null* if no layer is being moved).

In the function *whichObj()* we check the properties *left*, *top*, *width* and *height* for each layer. With the help of these values we can check which object the user clicked on.

Dropping objects

We have now everything we need in order to implement drag & drop. With our script the user can drag around objects on our web-page. But we haven't talked about dropping objects yet. Let's suppose that you want to create an online shop. You have several items which can be put into a shopping basket. The user has to drag these items to the shopping basket and drop them there. This means we have to register when the user drops an object on the shopping basket - which means that he wants to buy it.

Which part of the code do we have to change in order to implement this? We have to check which position the object has after a *MouseUp* event - i.e. we have to add some code to the function *endDrag()*. We could for example check if the coordinates of the mouse event lie inside a certain rectangle. If this is true you call a function which registers all items to buy (you might want to put them inside an array). Then you could show the item inside the shopping basket.

Improvements

There are several ways for improving our script. First we might want to change the order of the layers as soon as the user clicks on one object. Otherwise it might look a bit strange if you move an object and it disappears behind another object. You can solve this problem by changing the order of the layers in the *startDrag()* function.

I don't think that you'll be satisfied by putting up red, green and blue boxes on your web page. Add some cool graphics and the users will remember your page. You can place anything inside

the layer objects. So place a single `` tag there if you want your object to appear as an image.

©1996,1997 by Stefan Koch

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

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

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