Introduction To JavaScript – Part 5
The Event Object

Instructor: Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790

Department of Electrical Engineering and Computer Science
University of Central Florida
JavaScript – Part 5 – The Event Object

• When a W3C event listener’s event occurs and it calls its associated event handler (function), it also passes a single argument to the function – a reference to the event object.

• The event object contains a number of properties that describe the event that occurred.

• The table on the next page lists the names of the most commonly used properties of the event object, which of course usually differ between the W3C and Microsoft models.
### The Event Object

<table>
<thead>
<tr>
<th>W3C Name</th>
<th>Microsoft Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>window.event</td>
<td>The object containing the event properties</td>
</tr>
<tr>
<td>type</td>
<td>type</td>
<td>The event that occurred (click, focus, blur, etc.)</td>
</tr>
<tr>
<td>target</td>
<td>srcElement</td>
<td>The element to which the event occurred</td>
</tr>
<tr>
<td>keyCode</td>
<td>keyCode</td>
<td>The numerical ASCII value of the pressed key</td>
</tr>
<tr>
<td>shiftKey</td>
<td></td>
<td>Returns 1 if pressed, 0 if not</td>
</tr>
<tr>
<td>altKey</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cntlKey</td>
<td></td>
<td></td>
</tr>
<tr>
<td>currentTarget</td>
<td>fromElement</td>
<td>The element the mouse came from on mouseover</td>
</tr>
<tr>
<td>relatedTarget</td>
<td>toElement</td>
<td>The element the mouse went to on mouseout</td>
</tr>
</tbody>
</table>
By convention, the parameter name `e` is used in event-triggered functions to receive the event object argument.

If you wanted to determine the type of event that occurred, you’d write the following function:

```javascript
function myEvent(e) {
    var eventType = e.type
    alert("The following event has occurred: ", eventType);
    //the alert will display click, or whatever the event
    //type was
}
```
Note that this code would not work on Microsoft browsers, because the Microsoft model does not pass an event object reference like the W3C model; instead, it uses a central global object that contains the properties of the most recent event.

We’ll start by looking at the W3C model before we look at the Microsoft model.

To demonstrate how the W3C event model works, we’ll go back and modify the highlighting example from the previous set of notes.

We’ll modify the JavaScript functions `addHighlight()` and `removeHighlight()` so that they no longer have to get the target element before modifying it.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
    var emailField=document.getElementById("email"); // get the field
    addEvent(emailField, 'focus', addHighlight); // add focus event
    addEvent(emailField, 'blur', removeHighlight); // add blur event
}

function addHighlight() {
    var emailField=document.getElementById("email");
    emailField.style.backgroundColor="#6F3";
}

function removeHighlight() {
    var emailField=document.getElementById("email");
    emailField.style.backgroundColor=""; // field now goes back to default settings
}

</script>
</head>

<body>
<div id="sign_up"

<h3>Sign up for our newsletter</h3>
<form id="email_form" action="#" method="get">
    <label for="email">Email</label>
    <input id="email" name="email" type="text" size="24" />
</form>
window.onload = setUpFieldEvents;

function setUpFieldEvents() {
    var emailField = document.getElementById("email"); // get the field
    addEvent(emailField, 'focus', addHighlight); // add focus event
    addEvent(emailField, 'blur', removeHighlight); // add blur event
}

function addHighlight(e) {
    var emailField = e.target;
    emailField.style.backgroundColor = ";6F3;";
}

function removeHighlight(e) {
    var emailField = e.target;
    emailField.style.backgroundColor = ""; // field now goes back to default settings
}

<body>
    <div id="sign_up">
        <h3>Sign up for our newsletter</h3>
        <form id="email_form" action="#" method="get">
            <label for="email">Email</label>
        </form>
    </div>
</body>
Page operates as before.
As you can see, there is no visual change in how the page behaves when using the event object when compared to how the page behaved when accessing the target object directly via the DOM element.

However, you could now assign this same event listener to multiple input fields, and those fields would all display the same highlight behavior.

Instead of saying “highlight this field”, the code could state “highlight the field to which the event occurred.”

This gives us the ability to streamline the JavaScript to some extent. The next example will illustrate this.
The Event Object’s Type Property

- With access to the event object, you can also determine the type of event that occurred, e.g. `focus`, `blur`, `click`, etc., so you can use a single function to detect both the focus and the blur events.

- To do this, we’ll again modify the previous example.

- We’ll modify the event listeners to call the same function, we’ll call this function `checkHighlight`, which will make more sense because this function will both add and remove the highlighting.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
    var emailField=document.getElementById("email"); // get the field
    addEvent(emailField, 'focus', checkHighlight); // add focus event
    addEvent(emailField, 'blur', checkHighlight); // add blur event
}

function checkHighlight(e) {
    switch (e.type) {
    case "focus":
        e.target.style.backgroundColor="#6F3";
        break;
    case "blur":
        e.target.style.backgroundColor="";
        break;
    }
}</script>
</head>
<body>
Page operates as before – no visual change
The Event Object In Microsoft Browsers

• When an event triggers a function in W3C-compliant browsers, a reference to an object containing properties that describe the triggering event is passed to the function. This event object can be accessed through the `e` parameter.

• In Microsoft browsers, the model is slightly different. There is one global object, `window.event`, that holds the last event that occurred.

• Because its global, it doesn’t have to be passed to the function like the W3C event object; it’s always available to your code.
The Event Object In Microsoft Browsers

- The following two lines are equivalent in their respective browsers:

  ```javascript
  e.type
  window.event.type
  ```

  *W3C – e must be stated as a function parameter*
  *Microsoft – direct access of global event object*

- The simplest way to write cross-browser event object code is like this:

  ```javascript
  function eventType(e) {
      if (e) { alert(e.type) } //W3C
      else { alert(window.event.type) } //Microsoft
      //displays the triggering event
  }
  ```
The Event Object In Microsoft Browsers

• While the technique on the previous page works fine, branching your code for every event object property you want to use gets old fast.

• A better solution is to get the object, whichever kind it is, and give it a new name and use the OR operator to distinguish:

\[
\text{var evt} = e \text{ || window.event};
\]

• If \( e \) evaluates to true (a W3C object exists, the \( \text{evt} \) variable is set to \( e \) – the W3C event object with all its properties. If not, \( \text{evt} \) is set to the Microsoft object instead. Now you would have:

\[
\text{var evt} = e \text{ || window.event};
\text{alert(evt.type);}
\]
The Event Object In Microsoft Browsers

• The preceding code works because, unlike many event object properties, the property name for the type of event that occurred is the same – `type` – in both W3C and Microsoft browsers.

• If you want to get the event target, which is `target` for W3C-compliant browsers and `srcElement` for Microsoft browsers, you can expand on the previous example and again use an OR statement to create a common cross-browser name for the event target as well:

```javascript
var evt = e || window.event;
var evtTarget = evt.target || evt.srcElement;
alert(evtTarget);
```
The Event Object In Microsoft Browsers

• The next example applies this concept to create a cross-browser compatible version of the highlighted form example that we’ve been dealing with in the few previous examples.

• Notice again, that there are no visual changes to the way the page behaves with the exception that it now will also work in a Microsoft browser.

• The JavaScript is shown on the next page.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
    var emailField=document.getElementById("email"); // get the field
    addEvent(emailField, 'focus', checkHighlight); // add focus event
    addEvent(emailField, 'blur', checkHighlight); // add blur event
}

function checkHighlight(e) {
    var evt = e || window.event; // sets evt variable to either W3C or Microsoft event obj
    var evtTarget = evt.target || evt.srcElement; // gets the target of the event
    switch (evt.type) { // type is the same name in both objs
    case "focus":
        evtTarget.style.backgroundColor="#6F3";
        break;
    case "blur":
        evtTarget.style.backgroundColor="";
        break;
    }
}

</script>
</head>
<body>

Cross-browser compatible version of the single form field highlighting example
Cross-browser compatible version – Opera rendering
Cross-browser compatible version – IE rendering
The running example that we’ve been using has included a single field in the form to which we’ve been attaching event handlers and responding to events that happened in that particular field.

The versatility of the event object is in allowing you to attach event handlers to as many fields as you would like.

To illustrate, we’ll add a couple more fields to the basic form we’ve been using. In addition to the email field, I’ve added fields for the user to enter their first and last names.

The markup is shown on the next page.
</script>
</head>

<body>

<div id="sign_up">

<h3>Sign up for our newsletter</h3>
<form id="email_form" action="#" method="get">
  <label for="first_name">First Name</label>
  <input id="first_name" name="first_name" type="text" size="18"/>
  
  <label for="last_name">Last Name</label>
  <input id="last_name" name="last_name" type="text" size="18"/>
  
  <label for="email">Email</label>
  <input id="email" name="email" type="text" size="18"/>
  
  <input id="submit" type="submit" value="Go!"/>
</form>

</div>

</body>
</html>
Attaching Event Handlers To Multiple Fields

• Since there are now several different form elements, the “hook” into the DOM will need to be higher up at the form element’s id, which is email_form.

• Once the parent element is established, the you can get at all the form’s child elements within.

• To do this we’ll need to modify the setUpFieldEvents function.

• In keeping with our technique of testing that what we’re doing is correct, we’ll first modify this function to simply tell us how many input tags (children) are present in the form.

• This part of the modified JavaScript is shown on the next page.
function setUpFieldEvents() {
    var emailField=document.getElementById("email"); // get the field
    addEvent(emailField, 'focus', checkHighlight); // add focus event
    addEvent(emailField, 'blur', checkHighlight); // add blur event
}

function setUpFieldEvents() {
    var emailField=document.getElementById("email_form"); // get the field
    var theInputs=document.getElementsByTagName("input");
    var inputCount=theInputs.length;
    alert(inputCount);
}
ATTACHING EVENT HANDLERS TO MULTIPLE FIELDS

• Notice that the alert window informs us that there are 4 input elements in this form. Does this surprise you?

• There are four input elements because the submit button is also an input element, it simply has a different type (“submit”) than the other input elements (“text”).

• I don’t want to apply the highlighting styles to the button element. Without step-by-step testing such as this it would be easy to overlook this sort of case and this might have induced a weird “bug” into the code that would have changed the background color of the button every time it was clicked.

• Now, I’m aware of this and will worry about filtering out the button a bit later. For now, we’ll loop through all of the input fields and apply the event listeners to each of them.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
    var emailField=document.getElementById("email_form"); // get the field
    var theInputs=document.getElementsByTagName("input");
    var inputCount=theInputs.length;
    for (i=0; i < inputCount; i++) {
        addEvent(theInputs[i], 'focus', checkHighlight); // add focus event
        addEvent(theInputs[i], 'blur', checkHighlight); // add blur event
    }
}

function checkHighlight(e) {
    var evt = e || window.event; // sets evt variable to either W3C or
    var evtTarget = evt.target || evt.srcElement; // gets the target of the
    event
    switch (evt.type) { // type is the same name in both objs
        case "focus":
            evtTarget.style.backgroundColor="#6F3";
            break;
        case "blur":
            evtTarget.style.backgroundColor="";
            break;
    }
}
An Aside – JavaScript Processing

• I noticed several of you in your last program did something like the following (a modified version of the script on the previous page).

```javascript
function setUpFieldEvents() {
  var emailField=document.getElementById("email_form"); // get the field
  var theInputs=document.getElementsByTagName("input");
  for (i=0; i < theInputs.length; i++) {
    addEvent(theInputs[i], 'focus', checkHighlight); // add focus event
    addEvent(theInputs[i], 'blur', checkHighlight); // add blur event
  }
}
```

• This version is less efficient than the one on the previous page because JavaScript must recalculate the length of the `theInputs` node list every time the loop executes. Counting items in arrays and node lists is a relatively slow process in JavaScript. It isn’t such a big deal with a few items, but if you’re looping over a big data set or hundreds of table rows, the wasted time can add up.

• It’s always good practice to get the number of items once and store it in a variable that you can use as the loop count.
Attaching Event Handlers To Multiple Fields

• Returning to our original problem. Notice in the screen shot below that the problem we identified with the submit button is indeed a problem. We don’t want the submit button to be highlighted when the cursor moves over the button. We want that behavior to apply only to the form elements that the user is entering data.
Attaching Event Handlers To Multiple Fields

• What makes the submit button different from the text inputs is that its type attribute is submit and not text.

• First we’ll create a simple if statement filter based on this difference to identify and exclude the type submit from our highlighting.

• As before, we’ll add alert pop-up windows to help us test the code.

• The code is shown on the next page and an illustration of it running on the following page.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
  var emailField=document.getElementById("email_form"); // get the field
  var theInputs=document.getElementsByTagName("input");
  var inputCount=theInputs.length;
  for (i=0; i < inputCount; i++) {
    alert("This is a: " + theInputs[i].getAttribute("type")); // testing alerts
    addEvent(theInputs[i], 'focus', checkHighlight); // add focus event
    addEvent(theInputs[i], 'blur', checkHighlight); // add blur event
  }
}

function checkHighlight(e) {
  var evt = e || window.event; // sets evt variable to either W3C or Microsoft event
  var evtTarget = evt.target || evt.srcElement; // gets the target of the event
  switch (evt.type) {
    case "focus":
      evtTarget.style.backgroundColor="#6F3";
      break;
    case "blur":
      evtTarget.style.backgroundColor="";
      // other cases
  }
}
Attaching Event Handlers To Multiple Fields

• Now that we know that we can distinguish between the submit input type and the text input type, all that is left to do is work this test code into the JavaScript to be inside the for loop driving through the input fields.

• This will produce the final version of this markup/JavaScript into a complete working example that will attach event handlers to multiple elements. The markup is also cross-browser compatible due to our technique of being able to distinguish between the W3C event object model and the Microsoft event object model.

• The complete markup/JavaScript file is available on the course webpage for you to download and experiment with. The following page illustrates the final version of the relevant portion of the JavaScript and the subsequent pages illustrate its rendering.
window.onload=setUpFieldEvents;

function setUpFieldEvents() {
    var emailField=document.getElementById("email_form"); // get the field
    var theInputs=document.getElementsByTagName("input");
    var inputCount=theInputs.length;
    //alert(theInputs[0].getAttribute("type"));
    for (i=0; i < inputCount; i++) {
        var theInputType=theInputs[i].getAttribute("type");
        if (theInputType=="text") {
            addEvent(theInputs[i], 'focus', checkHighlight); // add focus event
            addEvent(theInputs[i], 'blur', checkHighlight); // add blur event
        }
    }
}

function checkHighlight(e) {
    var evt = e || window.event; // picks either W3C or Microsoft event obj
    var evtTarget = evt.target || evt.srcElement; // gets the target of the event
    switch (evt.type) { // type is the same name in both objs
        case "focus":
            evtTarget.style.backgroundColor="#6F3";
            break;
        case "blur":
            evtTarget.style.backgroundColor="";
            break;
    }
}
Sign up for our newsletter

First Name    Last Name    Email    Go!
Notice that the submit button does not have the highlighting applied to it. It now renders properly by the JavaScript ignoring it.
Sign up for our newsletter

First Name [ ] Last Name [ ] Email [ ] Go!

Sign up for our newsletter

First Name [ ] Last Name [ ] Email [ ] Go!
Sign up for our newsletter

First Name  Last Name  Email

Go!