

Working with Events

Dr. Patrick Young, Stanford University

CS193C

Working with events can be very confusing. There are a great many approaches which are available to you, and unfortunately many of them are browser-specific. IE9 goes a long way to eliminating browser-specific issues, however, professional web developers will need to support earlier versions of Internet Explorer for quite some time.

In this handout, I look at three aspects of working with events: assigning event handlers, accessing the event object, and attempting to get the W3C-compliant event model and the pre-IE9 Microsoft-IE event model to cooperate together.

Assigning Event Handlers

In order to get our JavaScript-enhanced webpage up and running, we'll not only need to write our JavaScript functions, we'll also need to get the web browser to call our JavaScript code. We do this by assigning JavaScript functions as event handlers. When a particular event occurs—such as when the user clicks on a particular item, types at the keyboard, or moves the mouse on top of an element—we can tell the web browser to run our JavaScript function.

There are a number of different ways to assign a JavaScript function to execute when an event occurs. Let's take a look:

Assign using HTML Attribute-Value Pair

The traditional approach to assigning an event handler is to include an attribute value-pair on one of your HTML tags. For example, in order to get the JavaScript function `myHandler` to run when the user clicks on a button, I could use the following HTML:

```
<input type="button" value="Press Here"
       onclick="myHandler();" />
```

Because I've included an `onclick` attribute in my HTML code, and because that `onclick` attribute has the value `"myHandler();"` the function `myHandler` will be called when this button is clicked.

There are several advantages to this approach. First, this approach works in both W3C-compliant web browsers such as Firefox and in all versions of Microsoft's Internet Explorer. In addition, using this approach, we can pass parameters to our event handler. In this next example, we pass the function `myHandler` the values `2`, `"hello"`, and `true`:

```
<input type="button" value="Press Here"
       onclick="myHandler(2, 'hello', true);" />
```

We'll discover that the other approaches don't allow us to pass parameters into a function.

Assign a Function Object directly to Element's Property

In order to understand this next method, you need to be aware of several JavaScript language characteristics. JavaScript Functions are first-class objects. That means that Functions are on par with other datatypes like numbers, booleans, strings, Arrays, or other objects. We can assign a Function to a variable, pass it in as a parameter, or do any of the other things we normally do with data values. When we define a function, we are implicitly creating a Function object, declaring a variable based on the function's name, and are assigning the Function object as the initial value of the variable. In other words, when I write code like this:

```
function myHandler() {  
    alert("item was clicked on");  
}
```

I am defining a new variable named `myHandler`, I am create a new Function object, and the new Function object is the initial value for the variable `myHandler`.

Now, suppose we have the following HTML code:

```

```

If we execute the following JavaScript code:

```
document.getElementById("photo").onclick=myHandler;
```

The function `myHandler` will now be executed if the user clicks on the image.

This method works on both W3C-compliant web browsers and on all versions of Microsoft Internet Explorer. Notice that in contrast to the previous method, we cannot pass parameters into the function which is called. One advantage to this approach is that assignment of the handler can be changed programmatically. We might have our JavaScript code setup to assign `document.photo.onclick` different values, for example, thus calling different event handling functions depending on current conditions. In contrast, when using the HTML attribute-value method described previously, the value of the `onclick` is always the same; it cannot be changed programmatically on the fly and it's specific value is always determined at the time the file is written.

One last precaution, this method as with the next two methods depends upon the HTML element having been seen by the web browser prior to assigning an event handler. I'll explore this issue in a few minutes in the section entitled: "Assigning Handlers using OnLoad".

Use the W3C's `addEventListener`

The proper way to assign an event handler on a W3C-compliant web browser is to call `addEventListener` on the element. This method also works in IE9, if IE9 interprets the webpage in IE9 standards mode.

Here is an example, given the HTML code:

```

```

we can assign the `` an event handler as follows:

```
document.getElementById("photo").addEventListener("click",myHandler,false);
```

`addEventListener` takes three parameters

- The type of event we want to script. Notice that the word “on” is not included with this parameter, thus we pass in “click” instead of “onclick”.
- The function to call when the event occurs. Notice that we pass in the actual Function object, thus there are no quotes around `myHandler`; we are passing in a Function object, not a string with the name of the function.
- A boolean which determines whether the handler is called during the W3C DOM capture phase or the bubble phase. In almost all cases, you’ll want this parameter to be `false`. Set it to `true` to have the handler called during the special capture phase.

This method has many of the same advantages and disadvantages of the previous method (the event handler is determined programmatically, but you have no way to send parameters to the function). One additional advantage, calling `addEventListener` is the only way to take advantage of the W3C’s special capture phase. In addition, using this method we can assign as many functions as we want as event handlers for the exact same event. For example, we might assign both the functions `myHandler` and `myNewHandler` to the same `onclick` event by executing both the following lines of code.

```
document.getElementById("photo").addEventListener("click",myHandler,false);
document.getElementById("photo").addEventListener("click",myNewHandler,false);
```

As with the previous approach, be sure to read the section: “Assigning Handlers using `OnLoad`”.

Use Microsoft’s `attachEvent`

Microsoft has their own equivalent to `addEventListener` called `attachEvent`.

```
document.getElementById("photo").attachEvent("onclick",myHandler);
```

is equivalent to our previous

```
document.getElementById("photo").addEventListener("click",myHandler,false);
```

`attachEvent` takes only two parameters:

- The type of event. In this case, we pass in the name of the event including the prefix “on”, thus “onclick” instead of the W3C’s “click”.
- The function to call when the event occurs, exactly the same as with the W3C’s `addEventListener`.

`attachEvent` only works on Microsoft Internet Explorer. Advantages and disadvantages are similar to those with `addEventListener`. However, pre-IE9 Microsoft’s event model does not have a capture phase. You must use `addEventListener` on IE9 if you want to take advantage of the capture phase—using `attachEvent` mode will cause IE9 to use a backward-compatibility mode for legacy support.

As with the last two methods, please pay attention to the next section “Assigning Handlers using `OnLoad`” when using this approach.

Assigning Handlers using `OnLoad`

All these assignment methods, except placing an attribute-value pair on the HTML tag itself, requires that we access the JavaScript object corresponding to the HTML element for which we wish to assign an event handler. This means that the HTML element must exist when we try to set the event handler. Consider the following HTML file:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
<script type="text/javascript">

function myHandler() {
    alert("hello");
}

document.getElementById("photo").onclick=myHandler;

</script>
</head>
<body>

</body>
</html>

```

Does this correctly set the function `myHandler` to execute when the user clicks on the photo? No, it does not. The code:

```
document.getElementById("photo").onclick=myHandler;
```

tries to execute before the web browser sees the `` tag. This occurs because the `<script>` tag in the head is listed before the `` tag in the body. There are several possible ways to handle this.

One straightforward method is to simply put your JavaScript code at the bottom of the body element. As we discussed in class, this also allows your webpage to render more quickly.

As an alternative, you can add a `onload` attribute-value pair to my `<body>` tag which calls the initialization function. The code would look like this:

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
<script type="text/javascript">

function myHandler() {
    alert("hello");
}

function init() {
    document.getElementById("photo").onclick=myHandler;

    ... // other event handler assignments go here
}

</script>
</head>
<body onload="init();">

</body>
</html>

```

Because the `init` function doesn't execute until the entire webpage is loaded, the event handlers are only assigned after each of the HTML elements has been read. This method may be useful if your code also depends on having all external files loaded.

Accessing the Event Object

Our next lesson is to learn how to access the special `Event` object. This object is automatically created for us, and can be used to find information such as the exact coordinates where the user clicked the mouse or which key was typed for a keyboard event. While this object is created for us by both IE and W3C-compliant web browsers, the method used to access it is different.¹

Accessing in IE

Accessing the `Event` object is very easy in IE. Microsoft has a global variable named `event`. You can access `event` from anywhere, as long as your code is executing in response to an event handler. For example, as long as `myHandler` is called in response to an event, this code should bring up an alert box telling us the mouse's x-location:

```

function myHandler() {
    alert(event.screenX);
}

```

Accessing in W3C-Compliant Web Browsers

Accessing in W3C-compliant Web Browsers is also easy, unless you've used an HTML tag's attribute-value pair to assign your event handler. If you assign your event handler using either of the two following formulas:

¹ Also as mentioned in lecture some of the properties on the `Event` object are also different on IE vs. W3C DOM-compliant browsers. Check the documentation Danny Goodman *Dynamic HTML* Online Supplement for details (see Lecture Links from the Events Lecture).

```
document.getElementById("photo").onclick=myHandler;
```

or

```
document.getElementById("photo").addEventListener("click",myHandler,false);
```

your event handler will receive the `Event` object as a parameter when it is called. So we can write our handler like this:

```
function myHandler(event) {  
    alert(event.screenX);  
}
```

Note that the handler will receive the `Event` object as a parameter, regardless of what we call the formal parameter, thus the following two definitions will work exactly the same as the previous definition:

```
function myHandler(evt) {  
    alert(evt.screenX);  
}  
  
function myHandler(q) {  
    alert(q.screenX);  
}
```

On the other hand, if we assign an event handler using an HTML tag's attribute-value pair, we'll receive the parameters we explicitly pass in instead of the `Event` object. For example, if we write:

```

```

as you might expect, the parameter passed into `myHandler` will be the value 3, not our `Event` object. We can explicitly tell the web browser we want it to pass in the `Event` object, by using the special word "event" as one of our actual parameters in the function call:

```

```

tells the web browser that we want to pass in two parameters to `myHandler`, the first one being the special `Event` object, the second being the number 3. While this looks like `event` is being treated as a global variable, it's not. For W3C-compliant web browsers, the word `event` only has meaning in the event handler attribute value in the HTML tag. If you were to try to access it elsewhere, you will receive an undefined variable error. For example, if we use the handler

```
function myHandler() {  
    alert(event.screenX);  
}
```

which works fine in IE, we will get an error in a W3C-compliant browser, as the word `event` only has meaning within the value for an `onclick`, `onload`, `onchange`, etc. HTML attribute-value pair.

Working with both Microsoft and W3C

What happens if you want to support both Microsoft and W3C-Compliant Browsers? You have several different approaches available, depending on whether or not you need to distinguish between the two browser types.

Explicitly Pass Event into Handlers

If you don't need to distinguish the browser types, one approach is to use the traditional HTML tag attribute-value pair to assign handlers, and then explicitly use the word event as a parameter:

```
function myHandler(evt) {
    alert(evt.screenX);
}

...

```

Here a Microsoft web browser sees that we are using the global variable `event` and passes it in as a parameter to the event handler. The W3C web browser sees that we're using the special word "event" and passes it in as a parameter to our event handler. In either case, our event handler receives the `Event` object as a parameter.

Write Code in the Handler

If we don't want to use the traditional HTML tag attribute-value pair to assign handlers, then we'll need a different approach to put the two models together. If we take any other approach to assign our event handlers, W3C-compliant web browsers will automatically receive the `Event` object as a parameter, whereas IE will not. Here's is some sample code which allows both IE and W3C to ultimately have the parameter `evt` assigned to the `Event` object (?: code copied from Danny Goodman's *Dynamic HTML* book).

```
function myHandler(evt) {
    evt = (evt) ? evt : ((event) ? event : null);
}

... document.getElementById("photo").onclick=myHandler;
    // assume handler is assigned something like this
```

Here the W3C-compliant web browsers pass the `Event` object in as a parameter, IE does not. In IE there is no initial value to the `evt` parameter (it's undefined) but through careful use of the `?:` operator we assign the `evt` parameter the value of the `event` global variable.²

Write Code in the Handler Distinguishing IE and W3C

We can take advantage of a similar technique to actually distinguish which type of web browser we are working with:

² The `?:` operator is a ternary operator which returns either the second or third operand based on the value of the first operand. Consider the following:

```
var max = (a > b) ? a : b;
```

If `a` is greater than `b`, then the `?:` returns `a`, otherwise it returns `b`, therefore `max` is always set to the larger of the two values.

```

function myHandler(evt) {
    if (evt) {
        alert("We are working with W3C-Compliant Browser");
        // do W3C stuff here
    }
    else if (window.event) {
        alert("We are working with Microsoft Browser");
        // do IE stuff here
    }
    else {
        alert("Not working with IE or W3C Browser");
        // do problematic stuff here
    }
}

... document.getElementById("photo").onclick=myHandler;
// assume handler is assigned something like this

```

The W3C browser passes in the Event object as a parameter, so that `evt` is defined but `window.event` is not. The Microsoft browser doesn't pass in any parameters, so that `evt` is undefined. However in IE the event global variable is set. I use the formula, `window.event` instead of accessing `event` directly, because it is illegal to access a non-existent global variable, but it is not illegal to access a non-existent property. Remember in the web browser all global variables are actually properties of the window, the `window.event` does exactly what we want—it accesses the event global variable if it exists, otherwise it evaluates to undefined. If we were to try to access IE's global variable `event` directly, a non-Microsoft web browser would see us attempting to access a non-existent variable and would flag this as an error.