

Unobtrusive Event Handling with Prototype

JavaScript

is commonly used to add functionality to HTML documents.

There are two ways to do this, with level one event handlers or using unobtrusive JavaScript. One allows you only to assign one event handler to an event at a time and is deeply coupled to your HTML markup. The other allows you to assign as many event handlers to an event as you wish and completely removes JavaScript from your markup.

Do you know which is which? This tutorial will cover what the difference

is, why using unobtrusive JavaScript is a better approach and how to implement unobtrusive event handlers using the Prototype JavaScript library.

Event Handler Types

There are two main types of event handlers in JavaScript, DOM level 0 and DOM level 2. DOM level 0 is the type of event handling that we all learned when we first were exposed to JavaScript: ``

```
Click Me
</a>
```

```
or<html>
  <head>
    <script>
      function helloWorld() {
        alert('Hello, world!');
      }
      window.onload = helloWorld;
    </script>
  </head>
  <body>
    <p>
      Some HTML here
    </p>
  </body>
</html>
```

So what is wrong with that? It has always worked in the past, right? There are two issues with the above: coupling and event overriding.

Coupling and Event Overriding

We all know coupling is bad. We decouple our data access from our views, we decouple our services from each other, we try to keep coupling to a minimum in every piece of code we write...except our JavaScript. Coupling of our JavaScript to markup prevents you from changing your markup without addressing your JavaScript as well. For example, let's say we didn't want to have the Click Me be a link anymore. Let's say we wanted it to be a cell on a table. How would you do that? You would change the markup, but then you would have to change the JavaScript as well. If you were using unobtrusive JavaScript, all you would have to do is change the markup. The JavaScript would still reference the same id so nothing would have to change.

Event overriding is the second reason to avoid the above way of doing JavaScript. What do I mean by event overriding? The way the event handlers in the previous example are associated are with what is called DOM Level 0 events. The issue with DOM Level 0 events is that you can only assign one event handler to a DOM Level 0. With unobtrusive JavaScript (using Prototype), you actually assign event handlers to a higher level (DOM Level 2 to be exact). This level allows for multiple event handlers to be assigned to one event. For example `<html>`

```
<head>
  <script>
    function helloWorld() {
      alert('Hello, world!');
    }
    window.onload = helloWorld;
  </script>
</head>
```

```

<body onload="alert('I beat you');">
  <p>
    Some HTML here
  </p>
</body>
</html>

```

The alert "I beat you" would be displayed, but the "Hello, world!" would not. If you used unobtrusive JavaScript, both could be triggered. Enter Prototype

So to alleviate the issues talked about before, we would rather use Prototype and develop unobtrusive JavaScript. At the core of Prototype's event handling is the Event.observe method. This method is used to associate an event handler with an element and its related event. So to recreate the above hello world example, we would do the following: <html>

```

<html>
  <head>
    <script>
      Event.observe(window, 'load', function() {
        alert('Hello, world!');
      });
    </script>
  </head>
  <body>
    <p>
      Some HTML here
    </p>
  </body>
</html>

```

To associate a handler to an event we call Event.observe with three parameters: an element (window in this case), the event to trigger the handler (in this case load), and finally a function to be called. In this case we defined the function inline, however you can define the handler elsewhere as well.

Now let's take a look at the second scenario we presented earlier. The one where the onload in the body tag would override the window.onload assignment in the JavaScript. <html>

```

<html>
  <head>
    <script>
      function helloWorld() {
        alert('Hello, world!');
      }
      function iBeatYou() {
        alert ('I beat you');
      }
      Event.observe(window, 'load', helloWorld);
      Event.observe(window, 'load', iBeatYou);
    </script>
  </head>
  <body>
    <p>
      Some HTML here
    </p>
  </body>
</html>

```

In the redone version above, both alert boxes will fire once the page loads instead of whoever was loaded last like in the original version. Finally, let's look at the click here example. Below, we have the original link but now the JavaScript is associated with the id. So if we want to change the markup, all we need to do is associate the same id to the new element and the JavaScript will still work (fyi, \$() is Prototype shorthand for document.getElementById()).<script>

```

Event.observe($('clickMeLink'), 'click', function() {
  alert('Hello, world!');
});
</script>
<a href="javascript:void(0);" id="clickMeLink">Click Me</a>

```

So that's all there is to it. I hope this helps you take a step towards better JavaScript writing. Good luck!