This PDF file contains pages extracted from HTML5 and CSS3, Second Edition, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2013 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.
HTML5 and CSS3
Second Edition
Level Up with Today’s Web Technologies

Brian P. Hogan
Edited by Susannah Davidson Pfalzer
Creating Pop-Up Windows with Custom Data Attributes

If you’ve built any web applications that use JavaScript to grab information out of a document, you know that it can sometimes involve a bit of hackery and parsing to make things work. You’ll end up inserting extra information into event handlers or abusing the rel or class attributes to inject behavior. Those days are now over thanks to the introduction of custom data attributes.

Custom data attributes all start with the prefix data- and are ignored by the validator for HTML5 documents. You can attach a custom data attribute to any element you’d like, whether it be metadata about a photograph, latitude and longitude coordinates, or, as you’ll see in this tip, dimensions for a pop-up window. Best of all, you can use custom data attributes right now in nearly every web browser, since they can be easily grabbed with JavaScript.

Separating Behavior from Content, or Why onclick() Is Bad

Over the years, pop-up windows have gotten a bad reputation, and rightly so. They’re often used to get you to look at an ad, to convince unsuspecting web surfers to install spyware or viruses, or, worse, to give away personal information that is then resold. It’s no wonder most browsers have some type of pop-up blocker available.

Pop-ups aren’t all bad, though. Web-application developers often rely on pop-up windows to display online help, additional options, or other important user-interface features. To make pop-ups less annoying, we need to implement them in an unobtrusive manner. When you look at AwesomeCo’s human-resources page, you see several links that display policies in pop-up windows. Most of them look like this:

```html
<a href='#' onclick="window.open('help/holiday_pay.html',WinName,'width=300,height=300');">Holiday pay</a>
```

This is a pretty common way to build links that spawn pop-ups. In fact, this is the way JavaScript newbies often learn to make pop-up windows. We should address a couple of problems with this approach before moving on, though.
**Improve Accessibility**

The very first problem you might notice is that the link’s destination isn’t set! If JavaScript is disabled, the link won’t take the user to the page. That’s a huge problem we need to address immediately. Don’t ever omit the `href` attribute or give it a value like this under *any* circumstances. Give it the address of the resource that would normally pop up.

```html
<a href='help/holiday_pay.html'
   onclick="window.open(this.href,WinName,'width=300,height=300');">
   Holiday pay
</a>
```

The JavaScript code can then read the attached element’s `href` attribute for the link’s location.

The first step toward building accessible pages is to ensure that all the functionality works *without* JavaScript. Writing the interactive JavaScript on top of this foundation can be much easier.

**Abolish the `onclick()`**

Keep the behavior separate from the content, just like you keep the presentation information separate by using linked style sheets. Using `onclick()` is easy at first, but imagine a page with fifty links, and you’ll see how the `onclick()` method gets out of hand. First, you’re repeating that JavaScript over and over again.

And if you generate this code from some server-side code, you’re making the resulting HTML much bigger than it needs to be.

Instead, we can give each of the anchors on the page a class that identifies it as a link.

```html
<a href='help/holiday_pay.html' class='popup'>Holiday Pay</a>
```

To make handling events work smoothly across browsers, we’ll use jQuery. At the bottom of the page, right above the closing `<body>` tag, bring in the jQuery library.

```html
<script
   src='http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js'>
</script>
```

Then right below that, add a new `<script>` tag that contains this code:
We use a jQuery selector to grab all elements with the class of `popup`, and then add an observer to each element’s click event.

The function we pass to the `click()` method will be executed when someone clicks the link. The `preventDefault` method prevents the default click-event behavior. In this case, it prevents the browser from following the link and displaying a new page.

However, we’ve lost the information on how to size and position the window. We want a page designer who isn’t very familiar with JavaScript to still be able to set the dimensions of a window on a per-link basis.

**Custom Data Attributes to the Rescue!**

Situations like this are common when building any JavaScript-enabled application. As we’ve seen, storing the window’s desired height and width with the code is desirable, but the `onclick()` approach has lots of drawbacks. Instead we can embed these attributes on the element. All we have to do is construct the link like this:

```html
<a href="help/holiday_pay.html" data-width="600" data-height="400" title="Holiday Pay" class="popup">Holiday pay</a>
```

Now we modify the `click()` event we wrote to grab the options from the link’s custom data attributes and pass them to the `window.open()` method.

```javascript
$("a.popup").click(function(event){
  event.preventDefault();
  var link = this;
  var href = link.getAttribute("href");
  var height = link.getAttribute("data-height");
  var width = link.getAttribute("data-width");

  window.open (href,"popup",
    "height=" + height +",width=" + width + ");
});
```
We use jQuery strictly to handle the `click()` event. The element we clicked on is represented by the keyword `this` within the click-handler function. Using `getAttribute()`, we retrieve the attributes we need from the elements to construct our pop-up window.

That’s all there is to it! The link now opens in a new window.

**Falling Back**

These attributes work in older browsers right now as long as they support JavaScript. The custom data attributes won’t trip up the browser, and your document will be valid. Since you’re using the HTML5 doctype, the attributes that start with `data-` will all be ignored.

We can access custom data attributes another way, using `dataset`. This converts the custom data attributes into properties, which we access like this:

```html
var height = link.dataset.height;
var width = link.dataset.width;
```

This is convenient, but you should be aware of a couple of issues. First and foremost, it doesn’t work in Internet Explorer 10 or previous, so it’s not a great fit for general-purpose solutions yet. Second, if you had a custom data attribute like `data-mobile-image-size`, we’d have to access it as `dataset.mobileImageSize`. The properties on `dataset` are converted to camel case.

**A Word of Caution**

In this example, we used custom data attributes to provide additional information to a client-side script. It’s a clever approach to a specific problem and illustrates one way to use these attributes. It does tend to mix presentation information with markup, but it’s a simple way to see how easy it is to use JavaScript to read values you embed in your page.