24

2005

24 WAYS

Credits

24 ways is the advent calendar for web geeks. For twenty-four days each December we publish a daily dose of web design and development goodness to bring you all a little Christmas cheer.

- *24 ways* is brought to you by Perch CMS
- Produced by Drew McLellan, Brian Suda, Anna Debenham and Owen Gregory.
- Designed by Paul Robert Lloyd.
- eBook published by edgeofmyseat.com and produced by Rachel Andrew.
- Possible only with the help and dedication of our authors.

2005

It all started here, in the heady days of Web 2.0. Ajax was the first new browser technology we'd seen in years, and combined with a new breed of libraries such as Prototype, it kick-started the JavaScript renaissance.

1. Easy Ajax with Prototype

Drew McLellan 24ways.org/200501

There's little more impressive on the web today than a appropriate touch of Ajax. Used well, Ajax brings a web interface much closer to the experience of a desktop app, and can turn a bear of an task into a pleasurable activity.

But it's really hard, right? It involves all the nasty JavaScript that no one ever does often enough to get really good at, and the browser support is patchy, and urgh it's just so much damn effort. Well, the good news is that – ta-da – it doesn't have to be a headache. But *man* does it still look impressive. Here's how to amaze your friends.

INTRODUCING PROTOTYPE.JS

Prototype is a JavaScript framework by Sam Stephenson designed to help make developing dynamic web apps a whole lot easier. In basic terms, it's a JavaScript file which you link into your page that then enables you to do cool stuff.

There's loads of capability built in, a portion of which covers our beloved Ajax. The whole thing is freely distributable under an MIT-style license, so it's good to go. What a nice man that Mr Stephenson is – friends, let us raise a hearty cup of mulled wine to his good name. Cheers! **sluurrrrp**.

First step is to download the latest Prototype and put it somewhere safe. I suggest underneath the Christmas tree.

CUTTING TO THE CHASE

Before I go on and set up an example of how to use this, let's just get to the crux. Here's how Prototype enables you to make a simple Ajax call and dump the results back to the page:

```
var url = 'myscript.php';
var pars = 'foo=bar';
var target = 'output-div';
var myAjax = new Ajax.Updater(target, url, {method:
'get', parameters: pars});
```

This snippet of JavaScript does a GET to `myscript.php`, with the parameter `foo=bar`, and when a result is returned, it places it inside the element with the ID `output-div` on your page.

KNOCKING UP A BASIC EXAMPLE

So to get this show on the road, there are three files we need to set up in our site alongside `prototype.js`. Obviously we need a basic HTML page with `prototype.js` linked in. This is the page the user interacts with. Secondly, we need our own JavaScript file for the glue between the interface and the stuff Prototype is doing. Lastly, we need the page (a PHP script in my case) that the Ajax is going to make its call too.

So, to that basic HTML page for the user to interact with. Here's one I found whilst out carol singing:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8"/>
    <title>Easy Ajax</title>
    <script type="text/javascript"
src="prototype.js"></script>
    <script type="text/javascript"
src="ajax.js"></script>
```

```
</head>
<body>
    <form method="get" action="greeting.php"
id="greeting-form">
        <div>
            <label for="greeting-name">Enter your
name:</label>
            <input id="greeting-name" type="text" />
            <input id="greeting-submit" type="submit"
value="Greet me!" />
        </div>
        <div id="greeting"></div>
    </form>
</body>
</html>
```

As you can see, I've linked in `prototype.js`, and also a file called `ajax.js`, which is where we'll be putting our glue. (Careful where you leave your glue, kids.)

Our basic example is just going to take a name and then echo it back in the form of a seasonal greeting. There's a form with an input field for a name, and crucially a DIV (`greeting`) for the result of our call. You'll also notice that the form has a submit button – this is so that it can function as a regular form when no JavaScript is available. It's important not to get carried away and forget the basics of accessibility.

MEANWHILE, BACK AT THE SERVER

So we need a script at the server which is going to take input from the Ajax call and return some output. This is normally where you'd hook into a database and do whatever transaction you need to before returning a result. To keep this as simple as possible, all this example here will do is take the name the user has given and add it to a greeting message. Not exactly Web 2-point-HoHoHo, but there you have it.

Here's a quick PHP script – `greeting.php` – that Santa brought me early.

```
<?php
     $the_name =
htmlspecialchars($_GET['greeting-name']);
     echo "<p>Season's Greetings, $the_name!</p>";
?>
```

You'll perhaps want to do something a little more complex within your own projects. Just sayin'.

GLUING IT ALL TOGETHER

Inside our `ajax.js` file, we need to hook this all together. We're going to take advantage of some of the handy listener routines and such that Prototype also makes available. The first task is to attach a listener to set the scene once the window has loaded. He's how we attach an `onload` event to the `window` object and get it to call a function named `init()`:

```
Event.observe(window, 'load', init, false);
```

Now we create our `init()` function to do our evil bidding. Its first job of the day is to hide the submit button for those with JavaScript enabled. After that, it attaches a listener to watch for the user typing in the name field.

```
function init(){
     $('greeting-submit').style.display = 'none';
     Event.observe('greeting-name', 'keyup', greet,
false);
}
```

As you can see, this is going to make a call to a function called `greet()` `onkeyup` in the `greeting-name` field. That function looks like this:

```
function greet(){
     var url = 'greeting.php';
     var pars =
'greeting-name='+escape($F('greeting-name'));
     var target = 'greeting';
     var myAjax = new Ajax.Updater(target, url, {method:
'get', parameters: pars});
}
```

The key points to note here are that any user input needs to be escaped before putting into the parameters so that it's URL-ready. The target is the ID of the element on the page (a DIV in our case) which will be the recipient of the output from the Ajax call.

THAT'S IT

No, seriously. That's everything. Try the example. Amaze your friends with your 1337 Ajax sk1llz.

ABOUT THE AUTHOR

Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a personal site covering web development issues and themes, takes photos and tweets a lot.

2. An Explanation of Ems

Richard Rutter 24ways.org/200502

Ems are so-called because they are thought to approximate the size of an uppercase letter M (and so are pronounced “emm”), although 1em is actually significantly larger than this. The typographer Robert Bringhurst describes the em thus:

> The em is a sliding measure. One em is a distance equal to the type size. In 6 point type, an em is 6 points; in 12 point type an em is 12 points and in 60 point type an em is 60 points. Thus a one em space is proportionately the same in any size.

To illustrate this principle in terms of CSS, consider these styles:

```
#box1 {
     font-size: 12px;
     width: 1em;
     height: 1em;
     border:1px solid black;
}

#box2 {
     font-size: 60px;
     width: 1em;
     height: 1em;
     border: 1px solid black;
}
```

These styles will render like:

and

Note that both boxes have a height and width of 1em but because they have different font sizes, one box is bigger than the other. Box 1 has a `font-size` of 12px so its width and height is also 12px; similarly the text of box 2 is set to 60px and so its width and height are also 60px.

ABOUT THE AUTHOR

Richard Rutter is a user experience consultant and director of Clearleft. In 2009 he cofounded the webfont service, Fontdeck. He runs an ongoing project called The Elements of Typographic Style Applied to the Web, where he extols the virtues of good web typography. Richard occasionally blogs at Clagnut, where he writes about design, accessibility and web standards issues, as well as his passion for music and mountain biking.

3. Improving Form Accessibility with DOM Scripting

Ian Lloyd   24ways.org/200503

The form `label` element is an incredibly useful little element – it lets you link the form field unquestionably with the descriptive label text that sits alongside or above it. This is a very useful feature for people using screen readers, but there are some problems with this element.

What happens if you have one piece of data that, for various reasons (validation, the way your data is collected/stored etc), needs to be collected using several form elements?

The classic example is date of birth – ideally, you'll ask for the date of birth once but you may have three inputs, one each for day, month and year, that you also need to provide hints about the format required. The problem is that to be truly accessible you need to label each field. So you end up needing something to say "this is a date of birth", "this is the day field", "this is the month field" and "this is the day field". Seems like overkill, doesn't it? And it can uglify a form no end.

There are various ways that you can approach it (and I think I've seen them all). Some people omit the `label` and rely on the `title` attribute to help the user through; others put text in a `label` but make the text 1 pixel high and merging in to the background so that screen readers can still get that information. The most common method, though, is simply to set the `label` to not display at all using the CSS `display:none` property/value pairing (a technique which, for the time being, seems to work on most screen readers). But perhaps we can do more with this?

The technique I am suggesting as another alternative is as follows (here comes the pseudo-code):

- Start with a totally valid and accessible form
- Ensure that each form input has a `label` that is linked to its related form control
- Apply a `class` to any `label` that you don't want to be visible (for example `superfluous`)

Then, through the magic of unobtrusive JavaScript/the DOM, manipulate the page as follows once the page has loaded:

- Find all the `label` elements that are marked as superfluous and hide them
- Find out what `input` element each of these `label` elements is related to
- Then apply a hint about formatting required for input (gleaned from the original, now-hidden label text) – add it to the form input as default text
- Finally, add in a behaviour that clears or selects the default text (as you choose)

So, here's the theory put into practice – a date of birth, grouped using a `fieldset`, and with the behaviours added in using DOM, and here's the JavaScript that does the heavy lifting.

But why not just use `display:none`? As demonstrated at Juicy Studio, `display:none` seems to work quite well for hiding `label` elements. So why use a sledge hammer to crack a nut? In all honesty, this is something of an experiment, but consider the following:

- Using the DOM, you can add extra levels of help, potentially across a whole form – or even range of forms – without necessarily increasing your markup (it goes beyond simply hiding labels)
- Screen readers today may identify a `label` that is set not to display, but they may not in the future – this might provide a way around

- By expanding this technique above, it might be possible to *visually* change the parent container that groups these items – in this case, a `fieldset` and `legend`, which are notoriously difficult to style *consistently* across different browsers – while still retaining the underlying semantic/logical structure

Well, it's an idea to think about at least. How is it for you? How else might you use DOM scripting to improve the accessiblity or usability of your forms?

ABOUT THE AUTHOR

Ian Lloyd founded Accessify.com, a web accessibility site, back in 2002 and has been a member of the Web Standards Project since 2003, where he is part of the Accessibility Task Force. He has written or co-authored a number of books on the topic of standards-based web design/development, most recently co-authoring on Pro CSS for Apress. He lives in Swindon, UK, a place best known for its ‘Magic Roundabout‘ and Doctor Who’s Billie Piper. (It’s not all bad, though.)

4. CSS Layout Starting Points

Rachel Andrew 24ways.org/200504

I build a lot of CSS layouts, some incredibly simple, others that cause sleepless nights and remind me of the torturous puzzle books that were given to me at Christmas by aunties concerned for my education. However, most of the time these layouts fit quite comfortably into one of a very few standard formats. For example:

- Liquid, multiple column with no footer
- Liquid, multiple column with footer
- Fixed width, centred

Rather than starting out with blank CSS and (X)HTML documents every time you need to build a layout, you can fairly quickly create a bunch of layout starting points, that will give you a solid basis for creating the rest of the design and mean that you don’t have to remember how a three column layout with a footer is best achieved every time you come across one!

These starting points can be really basic, in fact that’s exactly what you want as the final design, the fonts, the colours and so on will be different every time. It’s just the main sections we want to be able to quickly get into place. For example, here is a basic starting point CSS and XHTML document for a fixed width, centred layout with a footer.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <title>Fixed Width and Centred starting point
document</title>
     <link rel="stylesheet" type="text/css"
href="fixed-width-centred.css" />
     <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
</head>
<body>
<div id="wrapper">
     <div id="side">
          <div class="inner">
               <p>Sidebar content here</p>
          </div>
     </div>
     <div id="content">
          <div class="inner">
```

```
                <p>Your main content goes here.</p>
            </div>
        </div>
        <div id="footer">
            <div class="inner">
                <p>Ho Ho Ho!</p>
            </div>
        </div>
</div>
</body>
</html>
```

```
body {
        text-align: center;
        min-width: 740px;
        padding: 0;
        margin: 0;
 }

 #wrapper {
        text-align: left;
        width: 740px;
        margin-left: auto;
        margin-right: auto;
        padding: 0;
 }

 #content {
        margin: 0 200px 0 0;
 }

 #content .inner {
        padding-top: 1px;
        margin: 0 10px 10px 10px;
}

#side {
    float: right;
    width: 180px;
    margin: 0;
}

#side .inner {
    padding-top: 1px;
    margin: 0 10px 10px 10px;
}

#footer {
    margin-top: 10px;
    clear: both;
}

#footer .inner {
    margin: 10px;
}
```

9 times out of 10, after figuring out exactly what main elements I have in a layout, I can quickly grab the 'one I prepared earlier', mark-up the relevant sections within the ready-made divs, and from that point on, I only need to worry about the contents of those different areas. The actual layout is tried and tested, one that I know works well in different browsers and that is unlikely to throw me any nasty surprises later on. In addition, considering how the layout is going to work first prevents the problem of

developing a layout, then realising you need to put a footer on it, and needing to redevelop the layout as the method you have chosen won't work well with a footer.

While enjoying your mince pies and mulled wine during the 'quiet time' between Christmas and New Year, why not create some starting point layouts of your own? The css-discuss Wiki, CSS layouts section is a great place to find examples that you can try out and find your favourite method of creating the various layout types.

ABOUT THE AUTHOR

Rachel Andrew is a Director of edgeofmyseat.com, a UK web development consultancy and creators of the small content management system, Perch. She is the author of a number of books, most recently The Profitable Side Project Handbook and CSS3 Layout Modules, and is a regular columnist for A List Apart.

When not writing about business and technology on her blog at rachelandrew.co.uk or speaking at conferences, you will usually find Rachel running up and down one of the giant hills in Bristol.

5. DOM Scripting Your Way to Better Blockquotes

Jeremy Keith 24ways.org/200505

Block quotes are great. I don't mean they're great for indenting content – that would be an abuse of the browser's default styling. I mean they're great for semantically marking up a chunk of text that is being quoted verbatim. They're especially useful in blog posts.

```
<blockquote>
     <p>Progressive Enhancement, as a label for a
strategy for Web design,
     was coined by Steven Champeon in a series of
articles and presentations
     for Webmonkey and the SxSW Interactive
conference.</p>
 </blockquote>
```

Notice that you can't just put the quoted text directly between the `<blockquote>` tags. In order for your markup to be valid, block quotes may only contain block-level elements such as paragraphs.

There is an optional `cite` attribute that you can place in the opening `<blockquote>` tag. This should contain a URL containing the original text you are quoting:

```
<blockquote cite="http://en.wikipedia.org/wiki/
Progressive_Enhancement">
     <p>Progressive Enhancement, as a label for a
strategy for Web design,
     was coined by Steven Champeon in a series of
articles and presentations
     for Webmonkey and the SxSW Interactive
conference.</p>
 </blockquote>
```

Great! Except… the default behavior in most browsers is to completely ignore the `cite` attribute. Even though it contains important and useful information, the URL in the `cite` attribute is hidden.

You could simply duplicate the information with a hyperlink at the end of the quoted text:

```
<blockquote cite="http://en.wikipedia.org/wiki/
Progressive_Enhancement">
     <p>Progressive Enhancement, as a label for a
strategy for Web design,
     was coined by Steven Champeon in a series of
```

```
articles and presentations
     for Webmonkey and the SxSW Interactive
conference.</p>
     <p class="attribution">
          <a href="http://en.wikipedia.org/wiki/
Progressive_Enhancement">source</a>
     </p>
 </blockquote>
```

But somehow it feels wrong to have to write out the same URL twice every time you want to quote something. It could also get very tedious if you have a lot of quotes.

Well, “tedious” is no problem to a programming language, so why not use a sprinkling of DOM Scripting? Here’s a plan for generating an attribution link for every block quote with a `cite` attribute:

1. Write a function called prepareBlockquotes.
2. Begin by making sure the browser understands the methods you will be using.
3. Get all the `blockquote` elements in the document.
4. Start looping through each one.
5. Get the value of the `cite` attribute.
6. If the value is empty, continue on to the next iteration of the loop.
7. Create a paragraph.
8. Create a link.
9. Give the paragraph a class of “attribution”.

10. Give the link an `href` attribute with the value from the `cite` attribute.
11. Place the text “source” inside the link.
12. Place the link inside the paragraph.
13. Place the paragraph inside the block quote.
14. Close the for loop.
15. Close the function.

Here’s how that translates to JavaScript:

```
function prepareBlockquotes() {
     if (!document.getElementsByTagName ||
!document.createElement || !document.appendChild) return;
     var quotes =
document.getElementsByTagName("blockquote");
     for (var i=0; i<quotes.length; i++) {
          var source = quotes[i].getAttribute("cite");
          if (!source) continue;
          var para = document.createElement("p");
          var link = document.createElement("a");
          para.className = "attribution";
          link.setAttribute("href",source);

link.appendChild(document.createTextNode("source"));
          para.appendChild(link);
          quotes[i].appendChild(para);
     }
 }
```

Now all you need to do is trigger that function when the document has loaded:

```
window.onload = prepareBlockquotes;
```

Better yet, use Simon Willison's handy `addLoadEvent` function to queue this function up with any others you might want to execute when the page loads.

That's it. All you need to do is save this function in a JavaScript file and reference that file from the head of your document using `<script>` tags.

You can style the attribution link using CSS. It might look good aligned to the right with a smaller font size.

If you're looking for something to do to keep you busy this Christmas, I'm sure that this function could be greatly improved. Here are a few ideas to get you started:

- Should the text inside the generated link be the URL itself?
- If the block quote has a `title` attribute, how would you take its value and use it as the text inside the generated link?
- Should the attribution paragraph be placed outside the block quote? If so, how would you that (remember, there is an `insertBefore` method but no `insertAfter`)?
- Can you think of other instances of useful information that's locked away inside attributes? Access keys? Abbreviations?

ABOUT THE AUTHOR

Jeremy Keith is an Irish web developer living in Brighton, England where he works with the web consultancy firm Clearleft. He wrote the books, DOM Scripting, Bulletproof Ajax, and most recently HTML5 For Web Designers.

His latest project is Huffduffer, a service for creating podcasts of found sounds. When he's not making websites, Jeremy plays bouzouki in the band Salter Cane. His loony bun is fine benny lava.

6. Practical Microformats with hCard

Drew McLellan 24ways.org/200506

You've probably heard about microformats over the last few months. You may have even read the easily digestible introduction at Digital Web Magazine, but perhaps you've not found time to actually implement much yet. That's understandable, as it can sometimes be difficult to see exactly what you're *adding* by applying a microformat to a page. Sure, you're semantically enhancing the information you're marking up, and the Semantic Web is a great idea and all, but what benefit is it right now, today?

Well, the answer to that question is simple: you're adding *lots* of information that can be and is being used on the web here and now. The big ongoing battle amongst the big web companies if one of territory over information. Everyone's grasping for as much data as possible. Some of that information many of us are cautious to give away, but a lot of is happy to be freely available. Of the data you're giving away, it makes sense to give it as much meaning as possible, thus enabling anyone from your friends and family to the giant search company down the road to make the most of it.

Ok, enough of the waffle, let's get working.

INTRODUCING HCARD

You may have come across hCard. It's a microformat for describing contact information (or really address book information) from within your HTML. It's based on the vCard format, which is the format the contacts/address book program on your computer uses. All the usual fields are available – name, address, town, website, email, you name it.

If you're running Firefox and Greasemonkey (or if you can, just to try this out), install this user script. What it does is look for instances of the hCard microformat in a page, and then add in a link to pass any hCards it finds to a web service which will convert it to a vCard. Take a look at the *About the author* box at the bottom of this article. It's a hCard, so you should be able to click the icon the user script inserts and add me to your Outlook contacts or OS X Address Book with just a click.

So microformats *are* useful after all. Free microformats all round!

IMPLEMENTING HCARD

This is the really easy bit. All the hCard microformat is, is a bunch of predefined class names that you apply to the markup you've probably already got around your contact information. Let's take the example of the *About the author* box from this article. Here's how the markup looks without hCard:

```
<div class="bio">
     <h3>About the author</h3>
     <p>Drew McLellan is a web developer, author and
no-good swindler from
     just outside London, England. At the
     <a href="http://www.webstandards.org/">Web
Standards Project</a> he works
     on press, strategy and tools. Drew keeps a
     <a href="http://www.allinthehead.com/">personal
weblog</a> covering web
     development issues and themes.</p>
</div>
```

This is a really simple example because there's only two key bits of address book information here:- my name and my website address. Let's push it a little and say that the Web Standards Project is the organisation I work for – that gives us Name, Company and URL.

To kick off an hCard, you need a containing object with a class of `vcard`. The div I already have with a class of `bio` is perfect for this – all it needs to do is contain the rest of the contact information.

The next thing to identify is my name. hCard uses a class of `fn` (meaning Full Name) to identify a name. As is this case there's no element surrounding my name, we can just use a `span`. These changes give us:

```
<div class="bio vcard">
     <h3>About the author</h3>
     <p><span class="fn">Drew McLellan</span> is a web
developer...
```

The two remaining items are my URL and the organisation I belong to. The class names designated for those are `url` and `org` respectively. As both of those items are links in this case, I can apply the classes to those links. So here's the finished hCard.

```
<div class="bio vcard">
     <h3>About the author</h3>
     <p><span class="fn">Drew McLellan</span> is a web
developer, author and
     no-good swindler from just outside London, England.
     At the <a class="org"
href="http://www.webstandards.org/">Web Standards
Project</a>
     he works on press, strategy and tools. Drew keeps a
     <a class="url"
href="http://www.allinthehead.com/">personal weblog</a>
covering web
     development issues and themes.</p>
</div>
```

OK, that was easy. By just applying a few easy class names to the HTML I was already publishing, I've implemented an hCard that right now anyone with Greasemonkey can click to add to their address book, that Google and Yahoo! and whoever else can index and work out important things like which websites are associated with my name if they so choose (and boy, will they so choose), and in the future who knows what. In terms of effort, practically nil.

WHERE NEXT?

So that was a trivial example, but to be honest it doesn't really get much more complex even with the most pernickety permutations. Because hCard is based on vCard (a mature and well thought-out standard), it's all tried and tested. Here's some good next steps.

- Play with the hCard Creator
- Take a deep breath and read the spec
- Start implementing hCard as you go on your own projects – it takes very little time

hCard is just one of an ever-increasing number of microformats. If this tickled your fancy, I suggest subscribing to the microformats site in your RSS reader to keep in touch with new developments.

WHAT’S THE TAKE-AWAY?

The take-away is this. They may sound like just more Web 2-point-HoHoHo hype, but microformats are a well thought-out, and easy to implement way of adding greater depth to the information you publish online. They have some nice benefits right away – certainly at geek-level – but in the longer term they become much more significant. We’ve been at this long enough to know that the web has a long, long memory and that what you publish today will likely be around for years. But putting the extra depth of meaning into your documents *now* you can help guard that they’ll continue to be useful in the future, and not just a bunch of flat ASCII.

ABOUT THE AUTHOR

Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a personal site covering web development issues and themes, takes photos and tweets a lot.

7. Don't be eval()

Simon Willison 24ways.org/200507

JavaScript is an interpreted language, and like so many of its peers it includes the all powerful `eval()` function. `eval()` takes a string and executes it as if it were regular JavaScript code. It's incredibly powerful and incredibly easy to abuse in ways that make your code slower and harder to maintain. As a general rule, if you're using `eval()` there's probably something wrong with your design.

COMMON MISTAKES

Here's the classic misuse of `eval()`. You have a JavaScript object, `foo`, and you want to access a property on it – but you don't know the name of the property until runtime. Here's how **NOT** to do it:

```
var property = 'bar';
var value = eval('foo.' + property);
```

Yes it will work, but every time that piece of code runs JavaScript will have to kick back in to interpreter mode, slowing down your app. It's also dirt ugly.

Here's the right way of doing the above:

```
var property = 'bar';
var value = foo[property];
```

In JavaScript, square brackets act as an alternative to lookups using a dot. The only difference is that square bracket syntax expects a string.

SECURITY ISSUES

In any programming language you should be extremely cautious of executing code from an untrusted source. The same is true for JavaScript – you should be extremely cautious of running `eval()` against any code that may have been tampered with – for example, strings taken from the page query string. Executing untrusted code can leave you vulnerable to cross-site scripting attacks.

WHAT'S IT GOOD FOR?

Some programmers say that `eval()` is B.A.D. – Broken As Designed – and should be removed from the language. However, there are some places in which it can dramatically simplify your code. A great example is for use with `XMLHttpRequest`, a component of the set of tools more popularly known as Ajax. `XMLHttpRequest` lets you make a call back to the server from JavaScript without refreshing the whole page. A simple way of using this is to have the server return JavaScript code which is then passed to `eval()`. Here is a simple function for doing exactly that – it takes the URL to some JavaScript code (or a server-side script that produces JavaScript) and loads and executes that code using `XMLHttpRequest` and `eval()`.

```
function evalRequest(url) {
     var xmlhttp = new XMLHttpRequest();
     xmlhttp.onreadystatechange = function() {
          if (xmlhttp.readyState==4 &&
xmlhttp.status==200) {
               eval(xmlhttp.responseText);
          }
     }
     xmlhttp.open("GET", url, true);
     xmlhttp.send(null);
 }
```

If you want this to work with Internet Explorer you'll need to include this compatibility patch.

ABOUT THE AUTHOR

Simon Willison is a freelance client- and server-side Web developer and the co-creator of the Django Web framework. Simon's interests include OpenID and decentralised systems, unobtrusive JavaScript, rapid application development and RESTful Web Service APIs. Before going frelance Simon worked on Yahoo!'s Technology Development team, and prior to that at the Lawrence Journal-World, an award winning local newspaper in Kansas. Simon maintains a popular Web development weblog at simonwillison.net

Photo: Tom Coates

8. Centered Tabs with CSS

Ethan Marcotte 24ways.org/200508

Doug Bowman's Sliding Doors is pretty much the *de facto* way to build tabbed navigation with CSS, and rightfully so – it is, as they say, rockin' like Dokken. But since it relies heavily on floats for the positioning of its tabs, we're constrained to either left- or right-hand navigation. But what if we need a bit more flexibility? What if we need to place our navigation in the center?

Styling the `li` as a floated block does give us a great deal of control over margin, padding, and other presentational styles. However, we should learn to love the inline box – with it, we can create a flexible, *centered* alternative to floated navigation lists.

HUMBLE BEGINNINGS

Do an extra shot of 'nog, because you know what's coming next. That's right, a simple unordered list:

```
<div id="navigation">
     <ul>
          <li><a href="#"><span>Home</span></a></li>
          <li><a href="#"><span>About</span></a></li>
          <li><a href="#"><span>Our Work</span></a></li>
          <li><a href="#"><span>Products</span></a></li>
          <li class="last"><a href="#"><span>Contact
Us</span></a></li>
     </ul>
</div>
```

If we were wedded to using floats to style our list, we could easily fix the width of our `ul`, and trick it out with some `margin: 0 auto;` love to center it accordingly. But this wouldn't net us much flexibility: if we ever changed the number of navigation items, or if the user increased her browser's font size, our design could easily break.

Instead of worrying about floats, let's take the most basic approach possible: let's turn our list items into inline elements, and simply use `text-align` to center them within the `ul`:

```
#navigation ul, #navigation ul li {
     list-style: none;
     margin: 0;
     padding: 0;
}

#navigation ul {
     text-align: center;
}
```

```
#navigation ul li {
     display: inline;
     margin-right: .75em;
}

#navigation ul li.last {
     margin-right: 0;
}
```

Our first step is sexy, no? Well, okay, not really – but it gives us a good starting point. We've tamed our list by removing its default styles, set the list items to `display: inline`, and centered the lot. Adding a background color to the links shows us exactly how the different elements are positioned.

Now the fun stuff.

INLINE ELEMENTS, PADDING, AND YOU

So how do we give our links some dimensions? Well, as the CSS specification tells us, the `height` property isn't an option for inline elements such as our anchors. However, what if we add some padding to them?

```
#navigation li a {
     padding: 5px 1em;
}
```

I just love leading questions. Things are looking good, but something's amiss: as you can see, the padded anchors seem to be escaping their containing list.

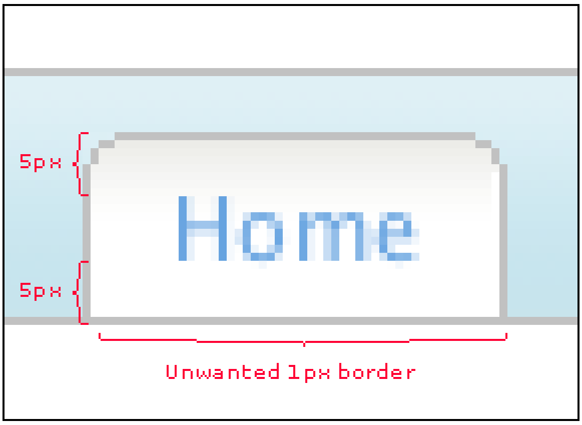
Thankfully, it's easy to get things back in line. Our anchors have 5 pixels of padding on their top and bottom edges, right? Well, by applying the same vertical padding to the list, our list will finally "contain" its child elements once again.

'TIS THE SEASON FOR TABBING

Now, we're finally able to follow the "Sliding Doors" model, and tack on some graphics:

```
#navigation ul li a {
     background: url("tab-right.gif") no-repeat 100% 0;
     color: #06C;
     padding: 5px 0;
     text-decoration: none;
}

#navigation ul li a span {
     background: url("tab-left.gif") no-repeat;
     padding: 5px 1em;
}

#navigation ul li a:hover span {
     color: #69C;
     text-decoration: underline;
}
```

Finally, our navigation's looking appropriately sexy. By placing an equal amount of padding on the top and bottom of the `ul`, our tabs are properly "contained", and we can subsequently style the links within them.



But what if we want them to bleed over the bottom-most border? Easy: we can simply decrease the bottom padding on the list by one pixel, like so.

A SPECIAL NOTE FOR SPECIAL BROWSERS

The Mac IE5 users in the audience are likely hopping up and down by now: as they've discovered, our centered navigation behaves rather annoyingly in their browser. As Philippe Wittenbergh has reported, Mac IE5 is known to create "phantom links" in a block-level element when `text-align` is set to any value but the default value of `left`. Thankfully, Philippe has documented a workaround that gets that ~~[censored]~~ venerable browser to behave. Simply place the following code into your CSS, and the links will be restored to their appropriate width:

```
/**//*/
#navigation ul li a {
     display: inline-block;
     white-space: nowrap;
     width: 1px;
}
/**/
```

IE for Windows, however, displays an extra kind of crazy. The padding I've placed on my anchors is offsetting the spans that contain the left curve of my tabs; thankfully, these shenanigans are easily straightened out:

```
/**/
* html #navigation ul li a {
     padding: 0;
}
/**/
```

And with that, we're finally finished.

ALL SET.

And that's it. With your centered navigation in hand, you can finally enjoy those holiday toddies and uncomfortable conversations with your skeevy Uncle Eustace.

ABOUT THE AUTHOR

Ethan Marcotte is a web designer and developer who cares about beautiful design, elegant code, and how the two intersect. He is currently working on a book about responsive web design, and drinking entirely too much coffee.

He swears profusely on Twitter, and would like to be an unstoppable robot ninja when he grows up. Beep.

Photo: Brian Warren

9. Putting the World into "World Wide Web"

Molly Holzschlag 24ways.org/200509

Despite the fact that the Web has been international in scope from its inception, the predominant mass of Web sites are written in English or another left-to-right language. Sites are typically designed visually for Western culture, and rely on an enormous body of practices for usability, information architecture and interaction design that are by and large centric to the Western world.

There are certainly many reasons this is true, but as more and more Web sites realize the benefits of bringing their products and services to diverse, global markets, the more demand there will be on Web designers and developers to understand how to put the **World** into World Wide Web.

INTERNATIONALIZATION

According to the W3C, Internationalization is:

> "…the design and development of a product, application or document content that enables easy localization for target audiences that vary in culture, region, or language."

Many Web designers and developers have at least heard, if not read, about Internationalization. We understand that the Web is in fact worldwide, but many of us never have the opportunity to work with Internationalization. Or, when we do, think of it in purely technical terms, such as "which character set do I use?"

At first glance, it might seem to many that Internationalization is the act of making Web sites available to international audiences. And while that is in fact true, this isn't done by broad-stroking techniques and technologies. Instead, it involves a far more narrow understanding of geographical, cultural and linguistic differences in specific areas of the world. This is referred to as *localization* and is the act of making a Web site make sense in the context of the region, culture and language(s) the people using the site are most familiar with.

Internationalization itself includes the following technical tasks:

- **Ensuring no barrier exists to the localization of sites**. Of critical importance in the planning stages of a site for Internationalized audiences, the role of the developer is to ensure that no barrier exists. This means being able to perform such tasks as enabling Unicode and making sure legacy character encodings are properly handled.
- **Preparing markup and CSS with Internationalization in mind**. The earlier in the site development process this occurs, the better. Issues such as ensuring that you can support bidirectional text, identifying language, and using CSS to support non-Latin typographic features.
- **Enabling code to support local, regional, language or culturally related references**. Examples in this category would include time/date formats, localization of calendars, numbering systems, sorting of lists and managing international forms of addresses.
- **Empowering the user**. Sites must be architected so the user can easily choose or implement the localized alternative most appropriate to them.

Localization

According to the W3C, Localization is the:

> …adaptation of a product, application or document content to meet the language, cultural and other requirements of a specific target market (a "locale").

So here's where we get down to thinking about the more sociological and anthropological concerns. Some of the primary localization issues are:

- **Numeric formats**. Different languages and cultures use numbering systems unlike ours. So, any time we need to use numbers, such as in an ordered list, we have to have a means of representing the accurate numbering system for the locale in question.
- **Money, honey**! That's right. I've got a pocketful of ugly U.S. dollars (why is U.S. money so unimaginative?). But I also have a drawer full of Japanese Yen, Australian Dollars, and Great British Pounds. Currency, how it's calculated and how it's represented is always a consideration when dealing with localization.
- **Using symbols, icons and colors properly**. Using certain symbols or icons on sites where they might offend or confuse is certainly not in the best interest of a site that wants to sell or promote a product, service or information type. Moreover, the colors we use are surprisingly persuasive – or detrimental. Think about colors that represent death, for example. In many parts of Asia, white is the color of death. In most of the Western world, black represents death. For Catholic Europe, shades of purple (especially lavender) have represented Christ on the cross and mourning since at least Victorian times. When Walt Disney World Europe launched an ad campaign using a lot of purple and very glitzy imagery, millions of dollars were

lost as a result of this seeming subtle issue. Instead of experiencing joy and celebration at the ads, the European audience, particularly the French, found the marketing to be overly American, aggressive, depressing and basically unappealing. Along with this and other cultural blunders, Disney Europe has become a well-known case study for businesses wishing to become international. By failing to understand localization differences, and how powerful color and imagery act on the human psyche, designers and developers are put to more of a disadvantage when attempting to communicate with a given culture.

- **Choosing appropriate references to objects and ideas**. What seems perfectly natural in one culture in terms of visual objects and ideas can get confused in another environment. One of my favorite cases of this has to do with Gerber baby food. In the U.S., the baby food is marketed using a cute baby on the package. Most people in the U.S. culturally do not make an immediate association that what is being represented on the label is what is *inside the container*. However, when Gerber expanded to Africa, where many people don't read, and where visual associations are less abstract, people made the inference that a baby on the cover of a jar of food represented what is in fact *in the jar*. You can imagine how confused and even angry people became. Using such approaches as a marketing ploy in the wrong locale can and will render the marketing a failure.

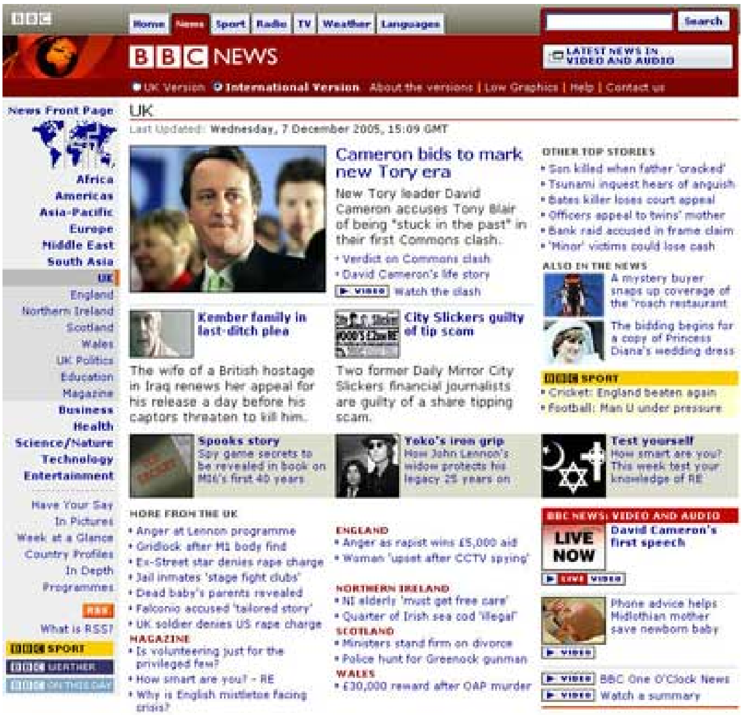
As you can see, the act of localization is one that can have profound impact on the success of a business or organization as it seeks to become available to more and more people across the globe.

RETHINKING DESIGN IN THE CONTEXT OF CULTURE

While well-educated designers and those individuals working specifically for companies that do a lot of localization understand these nuances, most of us don’t get exposed to these ideas. Yet, we begin to see how necessary it becomes to have an awareness of not just the technical aspects of Internationalization, but the socio-cultural ones within localization.

What’s more, the bulk of information we have when it comes to designing sites typically comes from studies and work done on sites built in English and promoted to Western culture at large. We’re making a critical mistake by not including diverse languages and cultural issues within our usability and information architecture studies.

Consider the following design from the BBC:



Home News Sport Radio TV Weather Languages

Search

BBC NEWS

LATEST NEWS IN VIDEO AND AUDIO

UK Version International Version About the versions | Low Graphics | Help | Contact us

News Front Page

Africa
Americas
Asia-Pacific
Europe
Middle East
South Asia
UK
England
Northern Ireland
Scotland
Wales
UK Politics
Education
Magazine
Business
Health
Science/Nature
Technology
Entertainment

Have Your Say
In Pictures
Week at a Glance
Country Profiles
In Depth
Programmes

RSS

What is RSS?

BBC SPORT
BBC WEATHER
BBC ON THIS DAY

UK

Last Updated: Wednesday, 7 December 2005, 15:09 GMT

Cameron bids to mark new Tory era

New Tory leader David Cameron accuses Tony Blair of being "stuck in the past" in their first Commons clash.

- Verdict on Commons clash
- David Cameron's life story

VIDEO Watch the clash

Kember family in last-ditch plea

The wife of a British hostage in Iraq renews her appeal for his release a day before his captors threaten to kill him.

City Slickers guilty of tip scam

Two former Daily Mirror City Slickers financial journalists are guilty of a share tipping scam.

Spooks story
Spy game secrets to be revealed in book on MI6's first 40 years

Yoko's iron grip
How John Lennon's widow protects his legacy 25 years on

Test yourself
How smart are you? This week test your knowledge of RE

OTHER TOP STORIES
- Son killed when father 'cracked'
- Tsunami inquest hears of anguish
- Bates killer loses court appeal
- Officers appeal to twins' mother
- Bank raid accused in frame claim
- 'Minor' victims could lose cash

ALSO IN THE NEWS
A mystery buyer snaps up coverage of the 'roach restaurant
The bidding begins for a copy of Princess Diana's wedding dress

BBC SPORT
- Cricket: England beaten again
- Football: Man U under pressure

MORE FROM THE UK
- Anger at Lennon programme
- Gridlock after M1 body find
- Ex-Street star denies rape charge
- Jail inmates 'stage fight clubs'
- Dead baby's parents revealed
- Falconio accused 'tailored story'
- UK soldier denies US rape charge

MAGAZINE
- Is volunteering just for the privileged few?
- How smart are you? - RE
- Why is English mistletoe facing crisis?

ENGLAND
- Anger as rapist wins £5,000 aid
- Woman 'upset after CCTV spying'

NORTHERN IRELAND
- NI elderly 'must get free care'
- Quarter of Irish sea cod 'illegal'

SCOTLAND
- Ministers stand firm on divorce
- Police hunt for Greenock gunman

WALES
- £30,000 reward after OAP murder

BBC NEWS: VIDEO AND AUDIO

LIVE NOW

David Cameron's first speech

LIVE VIDEO

Phone advice helps Midlothian mother save newborn baby

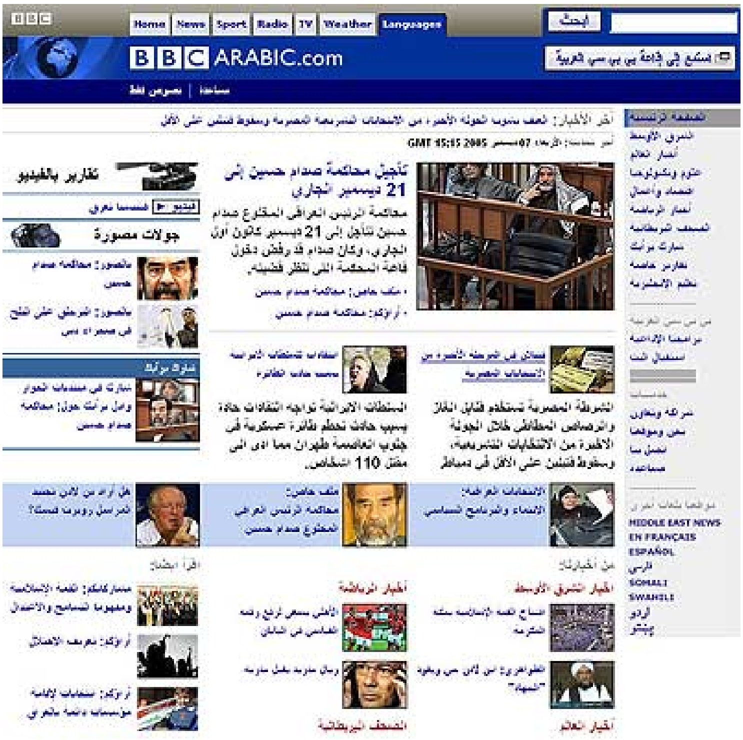
VIDEO

VIDEO BBC One O'Clock News
VIDEO Watch a summary

In this case, we're dealing with English, which is read left to right. We are also dealing with U.K. cultural norms. Notice the following:

- Location of of navigation
- Use of the color red
- Use of diverse symbols
- Mix of symbols, icons and photos
- Location of Search

Now look at this design, which is the Arabic version of the BBC News, read right to left, and dealing with cultural norms within the Arabic-speaking world.



Notice the following:

- Location of of navigation (location switches to the right)
- Use of the color blue (blue is considered the "safest" global color)
- No use of symbols and icons whatsoever

- Limitation of imagery to photos
- In most cases, the photos show people, not objects
- Location of Search

Admittedly, some choices here are more obvious than others in terms of why they were made. But one thing that stands out is that the placement of search is the same for both versions. Is this the result of a specific localization decision, or based on what we believe about usability at large? This is exactly the kind of question that designers working on localization have to seek answers to, instead of relying on popular best practices and belief systems that exist for English-only Web sites.

IT'S A WIDE WORLD WEB AFTER ALL

From this brief article on Internationalization, it becomes apparent that the art and science of creating sites for global audiences requires a lot more preparation and planning than one might think at first glance. Developers and designers not working to address these issues specifically due to time or awareness will do well to at least understand the basic process of making sites more culturally savvy, and better prepared for any future global expansion.

One thing is certain: We not only are on a dramatic learning curve for designing and developing Web sites as it is, the need to localize sites is going to become more and more a part of the day to day work. Understanding aspects of what makes a site international and local will not only help you expand your skill set and make you more marketable, but it will also expand your understanding of the world and the people within it, how they relate to and use the Web, and how you can help make their experience the best one possible.

ABOUT THE AUTHOR

Molly E. Holzschlag works to educate designers and developers on using Web technologies in practical ways to create highly sustainable, maintainable, accessible, interactive and beautiful

Web sites for the global community. A popular and colorful individual, Molly has a particular passion for people, blogs, and the use of technology for social progress.

Photo: Pete LePage

10. Auto-Selecting Navigation

Drew McLellan 24ways.org/200510

In the article Centered Tabs with CSS Ethan laid out a tabbed navigation system which can be centred on the page. A frequent requirement for any tab-based navigation is to be able to visually represent the currently selected tab in some way.

If you're using a server-side language such as PHP, it's quite easy to write something like `class="selected"` into your markup, but it can be even simpler than that.

Let's take the navigation div from Ethan's article as an example.

```
<div id="navigation">
     <ul>
          <li><a href="#"><span>Home</span></a></li>
          <li><a href="#"><span>About</span></a></li>
          <li><a href="#"><span>Our Work</span></a></li>
          <li><a href="#"><span>Products</span></a></li>
          <li class="last"><a href="#"><span>Contact
```

```
Us</span></a></li>
     </ul>
 </div>
```

As you can see we have a standard unordered list which is then styled with CSS to look like tabs. By giving each tab a class which describes it's logical section of the site, if we were to then apply a class to the body tag of each page showing the same, we could write a clever CSS selector to highlight the correct tab on any given page.

Sound complicated? Well, it's not a trivial concept, but actually applying it is dead simple.

MODIFYING THE MARKUP

First thing is to place a class name on each `li` in the list:

```
<div id="navigation">
     <ul>
          <li class="home"><a
href="#"><span>Home</span></a></li>
          <li class="about"><a
href="#"><span>About</span></a></li>
          <li class="work"><a href="#"><span>Our
Work</span></a></li>
          <li class="products"><a
href="#"><span>Products</span></a></li>
          <li class="last contact"><a
href="#"><span>Contact Us</span></a></li>
     </ul>
 </div>
```

Then, on each page of your site, apply the a matching class to the `body` tag to indicate which section of the site that page is in. For example, on your About page:

```
<body class="about">...</body>
```

WRITING THE CSS SELECTOR

You can now write a single CSS rule to match the selected tab on any given page. The logic is that you want to match the 'about' tab on the 'about' page and the 'products' tab on the 'products' page, so the selector looks like this:

```
body.home #navigation li.home,
 body.about #navigation li.about,
 body.work #navigation li.work,
 body.products #navigation li.products,
 body.contact #navigation li.contact{
      ... whatever styles you need to show the tab
selected ...
 }
```

So all you need to do when you create a new page in your site is to apply a class to the body tag to say which section it's in. The CSS will do the rest for you – without any server-side help.

ABOUT THE AUTHOR

Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a personal site covering web development issues and themes, takes photos and tweets a lot.

11. The Attribute Selector for Fun and (no ad) Profit

Andy Budd 24ways.org/200511

If I had a favourite CSS selector, it would undoubtedly be the attribute selector (Ed: You really need to get out more). For those of you not familiar with the attribute selector, it allows you to style an element based on the existence, value or partial value of a specific attribute.

At it's very basic level you could use this selector to style an element with particular attribute, such as a title attribute.

```
<abbr title="Cascading Style Sheets">CSS</abbr>
```

In this example I'm going to make all `<abbr>` elements with a title attribute grey. I am also going to give them a dotted bottom border that changes to a solid border on hover. Finally, for that extra bit of feedback, I will change the cursor to a question mark on hover as well.

```
abbr[title] {
     color: #666;
     border-bottom: 1px dotted #666;
 }

 abbr[title]:hover {
     border-bottom-style: solid;
     cursor: help;
 }
```

This provides a nice way to show your site users that `<abbr>` elements with title tags are special, as they contain extra, hidden information.

Most modern browsers such as Firefox, Safari and Opera support the attribute selector. Unfortunately Internet Explorer 6 and below does not support the attribute selector, but that shouldn't stop you from adding nice usability embellishments to more modern browsers.

Internet Explorer 7 looks set to implement this CSS2.1 selector, so expect to see it become more common over the next few years.

Styling an element based on the existence of an attribute is all well and good, but it is still pretty limited. Where attribute selectors come into their own is their ability to target the value of an attribute. You can use this for a variety of interesting effects such as styling VoteLinks.

VOTEWHATS?

If you haven't heard of VoteLinks, it is a microformat that allows people to show their approval or disapproval of a links destination by adding a pre-defined keyword to the rev attribute.

For instance, if you had a particularly bad meal at a restaurant, you could signify your dissaproval by adding a rev attribute with a value of vote-against.

```
<a href="http://www.mommacherri.co.uk/"
rev="vote-against">Momma Cherri's</a>
```

You could then highlight these links by adding an image to the right of these links.

```
a[rev="vote-against"]{
     padding-right: 20px;
     background: url(images/vote-against.png) no-repeat
right top;
}
```

This is a useful technique, but it will only highlight VoteLinks on sites you control. This is where user stylesheets come into effect. If you create a user stylesheet containing this rule, every site you visit that uses VoteLinks will receive your new style.

COOL HUH?

However my absolute favourite use for attribute selectors is as a lightweight form of ad blocking. Most online adverts conform to industry-defined sizes. So if you wanted to block all banner-ad sized images, you could simply add this line of code to your user stylesheet.

```
img[width="468"][height="60"],
img[width="468px"][height="60px"]  {
     display: none !important;
}
```

To hide any banner-ad sized element, such as flash movies, applets or iFrames, simply apply the above rule to every element using the universal selector.

```
*[width="468"][height="60"],
*[width="468px"][height="60px"]  {
     display: none !important;
}
```

Just bare in mind when using this technique that you may accidentally hide something that isn't actually an advert; it just happens to be the same size.

The Interactive Advertising Bureau lists a number of common ad sizes. Using these dimensions, you can create stylesheet that blocks all the popular ad formats. Apply this as a user stylesheet and you never need to suffer another advert again.

Here's wishing you a Merry, ad-free Christmas.

ABOUT THE AUTHOR

Andy Budd is an internationally renowned web designer, developer and weblog author based in Brighton, England. He specialises in building attractive, accessible, and standards complaint web solutions as a Director of Clearleft. Andy enjoys writing about web techniques for sites such as digital-web.com and his work has been featured in numerous magazines, books, and websites around the world. He is the author of CSS Mastery: Advanced Web Standards Solutions.

12. Introduction to Scriptaculous Effects

Michael Heilemann 24ways.org/200512

Gather around kids, because this year, much like in that James Bond movie with Denise Richards, Christmas is coming early… in the shape of scrumptuous smooth javascript driven effects at your every whim.

Now what I’m going to do, is take things down a notch. Which is to say, you don’t need to know much beyond how to open a text file and edit it to follow this article. Personally, I for instance can’t code to save my life.

Well, strictly speaking, that’s not entirely true. If my life was on the line, and the code needed was really simple and I wasn’t under any time constraints, then yeah maybe I could hack my way out of it

But my point is this: I'm not a programmer in the traditional sense of the word. In fact, what I do best, is scrounge code off of other people, take it apart and then put it back together with duct tape, chewing gum and dumb blind luck.

No, don't run! That happens to be a good thing in this case. You see, we're going to be implementing some really snazzy effects (which are considerably more relevant than most people are willing to admit) on your site, and we're going to do it with the aid of Thomas Fuchs' amazing Script.aculo.us library. And it will be like stealing candy from a child.

WHAT ARE WE DOING?

I'm going to show you the very basics of implementing the Script.aculo.us javascript library's Combination Effects. These allow you to fade elements on your site in or out, slide them up and down and so on.

WHY USE EFFECTS AT ALL?

Before get started though, let me just take a moment to explain how I came to see smooth transitions as something more than smoke and mirror-like effects included for with little more motive than to dazzle and make parents go 'uuh, snazzy'.

Earlier this year, I had the good fortune of meeting the kind, gentle and quite knowledgable Matt Webb at a conference here in Copenhagen where we were both speaking (though I will be the first to admit my little talk on Open Source Design was vastly inferior to Matt's talk). Matt held a talk called Fixing Broken Windows (based on the Broken Windows theory), which really made an impression on me, and which I have since then referred back to several times.

You can listen to it yourself, as it's available from Archive.org. Though since Matt's session uses many visual examples, you'll have to rely on your imagination for some of the examples he runs through during it. Also, I think it looses audio for a few seconds every once in a while.

Anyway, one of the things Matt talked a lot about, was how our eyes are wired to react to movement. The world doesn't flickr. It doesn't disappear or suddenly change and force us to look for the change. Things *move smoothly* in the real world. They do not pop up.

HOW IT WORKS

Once the necessary files have been included, you trigger an effect by pointing it at the ID of an element. Simple as that.

IMPLEMENTING THE EFFECTS

So now you know why I believe these effects have a place in your site, and that's half the battle. Because you see, actually getting these effects up and running, is deceptively simple.

First, go and download the latest version of the library (as of this writing, it's version 1.5 rc5). Unzip itand open it up.

Now we're going to bypass the instructions in the readme file. Script.aculo.us can do a bunch of quite advanced things, but all we really want from it is its effects. And by sidestepping the rest of the features, we can shave off roughly 80KB of unnecessary javascript, which is well worth it if you ask me.

As with Drew's article on Easy Ajax with Prototype, script.aculo.us also uses the Prototype framework by Sam Stephenson. But contrary to Drew's article, you don't have to download Prototype, as a version comes bundled with script.aculo.us (though feel free to upgrade to the latest version if you so please).

So in the unzipped folder, containing the script.aculo.us files and folder, go into 'lib' and grab the 'prototype.js' file. Move it to whereever you want to store the javascript files. Then fetch the 'effects.js' file from the 'src' folder and put it in the same place.

To make things even easier for you to get this up and running, I have prepared a small javascript snippet which does some checking to see what you're trying to do. The script.aculo.us effects are all either 'turn this off' or 'turn this on'. What this snippet does, is check to see what state the target currently has (is it on or off?) and then use the necessary effect.

You can either skip to the end and download the example code, or copy and paste this code into a file manually (I'll refer to that file as combo.js):

```
Effect.OpenUp = function(element) {
     element = $(element);
     new Effect.BlindDown(element, arguments[1] || {});
 }

 Effect.CloseDown = function(element) {
     element = $(element);
     new Effect.BlindUp(element, arguments[1] || {});
 }

 Effect.Combo = function(element) {
     element = $(element);
     if(element.style.display == 'none') {
          new Effect.OpenUp(element, arguments[1] ||
{});
     }else {
          new Effect.CloseDown(element, arguments[1] ||
{});
     }
 }
```

Currently, this code uses the BlindUp and BlindDown code, which I personally like, but there's nothing wrong with you changing the effect-type into one of the other effects available.

Now, include the three files in the header of your code, like so:

```
<script src="prototype.js" type="text/
javascript"></script>
<script src="effects.js" type="text/javascript"></script>
<script src="combo.js" type="text/javascript"></script>
```

Now insert the element you want to use the effect on, like so:

```
<div id="content" style="display: none;">Lorem ipsum
dolor sit amet.</div>
```

The above element will start out invisible, and when triggered will be revealed. If you want it to start visible, simply remove the style parameter.

And now for the trigger

```
<a href="javascript:Effect.Combo('content');">Click
Here</a>
```

And that, is pretty much it. Clicking the link should unfold the DIV targeted by the effect, in this case 'content'.

EFFECT OPTIONS

Now, it gets a bit long-haired though. The documentation for script.aculo.us is next to non-existing, and because of that you'll have to do some digging yourself to appreciate the full potentialof these effects.

First of all, what kind of effects are available? Well you can go to the demo page and check them out, or you can open the 'effects.js' file and have a look around, something I recommend doing regardlessly, to gain an overview of what exactly you're dealing with.

If you dissect it for long enough, you can even distill some of the options available for the various effects. In the case of the BlindUp and BlindDown effect, which we're using in our example (as triggered from combo.js), one of the things that would be interesting to play with would be the duration of the effect. If it's too long, it will feel slow and unresponsive. Too fast and it will be imperceptible.

You set the options like so:

```
<a href="javascript:Effect.Combo('content', {duration:
.2});">Click Here</a>
```

The change from the previous link being the inclusion of `, {duration: .2}`. In this case, I have lowered the duration to 0.2 second, to really make it feel snappy.

You can also go all-out and turn on all the bells and whistles of the Blind effect like so (slowed down to a duration of three seconds so you can see what's going on):

```
<a href="javascript:Effect.Combo('content', {duration:
3, scaleX: true, scaleContent: true});">Click Here</a>
```

CONCLUSION

And that's pretty much it. The rest is a matter of getting to know the rest of the effects and their options as well as finding out just when and where to use them. Remember the ancient Chinese saying: Less is more.

DOWNLOAD EXAMPLE

I have prepared a very basic example, which you can download and use as a reference point.

ABOUT THE AUTHOR

Michael Heilemann is a 30-year-old Computer Game Developer and Interface Design Enthusiast from Copenhagen, Denmark.

13. Transitional vs. Strict Markup

Roger Johansson 24ways.org/200513

When promoting web standards, standardistas often talk about XHTML as being more strict than HTML. In a sense it is, since it requires that all elements are properly closed and that attribute values are quoted. But there are two flavours of XHTML 1.0 (three if you count the Frameset DOCTYPE, which is outside the scope of this article), defined by the Transitional and Strict DOCTYPEs. And HTML 4.01 also comes in those flavours.

The names reveal what they are about: Transitional DOCTYPEs are meant for those making the transition from older markup to modern ways. Strict DOCTYPEs are actually the default – the way HTML 4.01 and XHTML 1.0 were constructed to be used.

A Transitional DOCTYPE may be used when you have a lot of legacy markup that cannot easily be converted to comply with a Strict DOCTYPE. But Strict is what you should be aiming for. It encourages, and in some cases enforces, the separation of structure and presentation, moving the presentational aspects from markup to CSS. From the HTML 4 Document Type Definition:

> This is HTML 4.01 Strict DTD, which excludes the presentation attributes and elements that W3C expects to phase out as support for style sheets matures. Authors should use the Strict DTD when possible, but may use the Transitional DTD when support for presentation attribute and elements is required.

An additional benefit of using a Strict DOCTYPE is that doing so will ensure that browsers use their strictest, most standards compliant rendering modes.

Tommy Olsson provides a good summary of the benefits of using Strict over Transitional in Ten questions for Tommy Olsson at Web Standards Group:

> In my opinion, using a Strict DTD, either HTML 4.01 Strict or XHTML 1.0 Strict, is far more important for the quality of the future web than whether or not there is an X in front of the name. The Strict DTD promotes a separation of structure and presentation, which makes a site so much easier to maintain.

For those looking to start using web standards and valid, semantic markup, it is important to understand the difference between Transitional and Strict DOCTYPEs. For complete listings of the differences between Transitional and Strict DOCTYPEs, see XHTML: Differences between Strict & Transitional, Comparison of Strict and Transitional XHTML, and XHTML1.0 Element Attributes by DTD.

Some of the differences are more likely than others to cause problems for developers moving from a Transitional DOCTYPE to a Strict one, and I'd like to mention a few of those.

ELEMENTS THAT ARE NOT ALLOWED IN STRICT DOCTYPES

- `center`
- `font`
- `iframe`
- `strike`
- `u`

ATTRIBUTES NOT ALLOWED IN STRICT DOCTYPES

- `align` (allowed on elements related to tables: `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead`, and `tr`)
- `language`

- `background`
- `bgcolor`
- `border` (allowed on `table`)
- `height` (allowed on `img` and `object`)
- `hspace`
- `name` (allowed in HTML 4.01 Strict, not allowed on `form` and `img` in XHTML 1.0 Strict)
- `noshade`
- `nowrap`
- `target`
- `text`, `link`, `vlink`, and `alink`
- `vspace`
- `width` (allowed on `img`, `object`, `table`, `col`, and `colgroup`)

CONTENT MODEL DIFFERENCES

An element type's content model describes what may be contained by an instance of the element type. The most important difference in content models between Transitional and Strict is that `blockquote`, `body`, and `form` elements may only contain block level elements. A few examples:

- text and images are not allowed immediately inside the body element, and need to be contained in a block level element like `p` or `div`

- `input` elements must not be direct descendants of a `form` element
- text in `blockquote` elements must be wrapped in a block level element like `p` or `div`

GO STRICT AND MOVE ALL PRESENTATION TO CSS

Something that can be helpful when doing the transition from Transitional to Strict DOCTYPEs is to focus on what each element of the page you are working on **is** instead of how you want it to **look**.

Worry about looks later and get the structure and semantics right first.

ABOUT THE AUTHOR

Roger Johansson is a Swedish web professional who has been working with the web and other interactive media since 1994.

Photo: Paul Hammond

14. Broader Border Corners

Patrick Griffiths

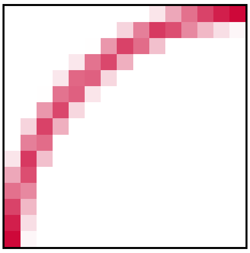
A note from the editors: Since this article was written the CSS border-radius property has become widely supported in browsers. It should be preferred to this image technique.

24ways.org/200514

A quick and easy recipe for turning those single-pixel borders that the kids love so much into into something a little less right-angled.

Here's the principle: We have a box with a one-pixel wide border around it. Inside that box is another box that has a little rounded-corner background image sitting snugly in one of its corners. The inner-box is then nudged out a bit so that it's actually sitting on top of the outer box. If it's all done properly, that little background image can mask the hard right angle of the default border of the outer-box, giving the impression that it actually has a rounded corner.

TAKE AN IMAGE, FINELY CHOPPED



ADD A SPRINKLE OF MARKUP

```
<div id="content">
    <p>Lorem ipsum etc. etc. etc.</p>
</div>
```

THROW IN A DOLLOP OF CSS

```
#content {
     border: 1px solid #c03;
}

#content p {
     background: url(corner.gif) top left no-repeat;
     position: relative;
     left: -1px;
     top: -1px;
     padding: 1em;
     margin: 0;
}
```

BUBBLIN' HOT

- The content div has a one-pixel wide red border around it.
- The paragraph is given a single instance of the background image, created to look like a one-pixel wide arc.
- The paragraph is shunted outside of the box – back one pixel and up one pixel – so that it is sitting over the div's border. The white area of the image covers up that part of the border's corner, and the arc meets up with the top and left border.
- Because, in this example, we're applying a background image to a paragraph, its top margin needs to be zeroed so that it starts at the top of its container.

Et voilà. Bon appétit.

EXTRA TOPPINGS

- If you want to apply a curve to each one of the corners and you run out of meaningful markup to hook the background images on to, throw some spans or divs in the mix (there's *nothing* wrong with this if that's the effect you truly want – they don't hurt anybody) or use some nifty DOM Scripting to put the scaffolding in for you.

- Note that if you've got more than one of these relative corners, you will need to compensate for the starting position of each box which is nested in an already nudged parent.
- You're not limited to one pixel wide, rounded corners – the same principles apply to thicker borders, or corners with different shapes.

ABOUT THE AUTHOR

Patrick Griffiths has been doing the professional web developer thing since 1999, and HTML and CSS has pretty much always been his specialty. He has worked for the likes of

Vodafone, Wiley, and on various UK Government projects, and has contributed a number of articles and projects to well respected web design resources.

15. Splintered Striper

Patrick Lauke 24ways.org/200515

Back in March 2004, David F. Miller demonstrated a little bit of DOM scripting magic in his A List Apart article *Zebra Tables*.

His script programmatically adds two alternating CSS background colours to table rows, making them more readable and visually pleasing, while saving the document author the tedious task of manually assigning the styling to large static data tables.

Although David's original script performs its duty well, it is nonetheless very specific and limited in its application. It only:

- works on a single `table`, identified by its `id`, with at least a single `tbody` section
- assigns a background colour
- allows two colours for odd and even rows
- acts on data cells, rather than rows, and then only if they have no class or background colour already defined

TAKING IT FURTHER

In a recent project I found myself needing to apply a striped effect to a medium sized unordered list. Instead of simply modifying the *Zebra Tables* code for this particular case, I decided to completely recode the script to make it more generic.

Being more general purpose, the function in my splintered striper experiment is necessarily more complex. Where the original script only expected a single parameter (the `id` of the target `table`), the new function is called as follows:

```
striper('[parent element tag]','[parent element class or null]','[child element tag]','[comma separated list of classes]')
```

This new, fairly self-explanatory function:

- targets any type of parent element (and, if specified, only those with a certain class)
- assigns two or more classes (rather than just two background colours) to the child elements inside the parent
- preserves any existing classes already assigned to the child elements

SEE IT IN ACTION

View the demonstration page for three usage examples. For simplicity's sake, we're making the calls to the striper function from the `body`'s `onload` attribute. In a real deployment situation, we would look at attaching a behaviour to the `onload` programmatically — just remember that, as we need to pass variables to the *striper* function, this would involve creating a wrapper function which would then be attached…something like:

```
function stripe() {
    striper('tbody','splintered','tr','odd,even');
}

window.onload=stripe;
```

A FINAL THOUGHT

Just because the function is called *striper* does not mean that it's limited to purely applying a striped look; as it's more of a general purpose “alternating class assignment” script, you can achieve a whole variety of effects with it.

ABOUT THE AUTHOR

Patrick H. Lauke works as Web Evangelist in the Developer Relations team at Opera Software. He has been engaged in the discourse on standards and accessibility since early 2001 – regularly speaking at conferences and contributing to a variety of web development and accessibility related mailing lists and initiatives such as the Web Standards Project and the Webkrauts. For more of his ruminations and weird experiments you can visit Patrick's personal site.

16. "Z's not dead baby, Z's not dead"

Andrew Clarke 24ways.org/200516

While Mr. Moll and Mr. Budd have pipped me to the post with their predictions for 2006, I'm sure they won't mind if I sneak in another. "The use of positioning together with z-index will be one of next year's *hot* techniques"

Both has been a little out of favour recently. For many, positioned layouts made way for the flexibility of floats. Developers I speak to often associate z-index with Dreamweaver's *layers* feature. But in combination with alpha transparency support for PNG images in IE7 and full implementation of position property values, the stacking of elements with z-index is going to be big. I'm going to cover the basics of z-index and how it can be used to create designs which 'break out of the box'.

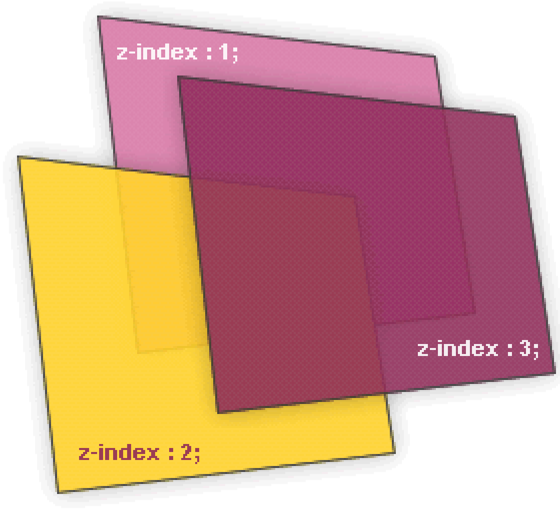
NO POSITIONING? NO Z!

Remember geometry? The x axis represents the horizontal, the y axis represents the vertical. The z axis, which is where we get the z-index, represents /depth/. Elements which are stacked using z-index are stacked from front to back and z-index is only applied to elements which have their position property set to `relative` or `absolute`. No positioning, no z-index. Z-index values can be either negative or positive and it is the element with the highest z-index value appears *closest* to the viewer, regardless of its order in the source. Furthermore, if more than one element are given the same z-index, the element which comes last in source order comes out top of the pile.

Let’s take three `<div>`s.

```
<div id="one"></div>
 <div id="two"></div>
 <div id="three"></div>

 #one {
     position: relative;
     z-index: 3;
 }

 #two {
     position: relative;
     z-index: 1;
 }
```

```
#three {
    position: relative;
    z-index: 2;
}
```



As you can see, the `<div>` with the z-index of 3 will appear closest, even though it comes before its siblings in the source order. As these three `<div>`s have no defined positioning context in the form of a positioned parent such as a `<div>`, their stacking order is defined from the root element `<html>`. Simple stuff, but these building blocks are the basis on which we can create interesting interfaces (particularly when used in combination with image replacement and transparent PNGs).

BRAND BUILDING

Now let's take three more basic elements, an `<h1>`, `<blockquote>` and `<p>`, all inside a *branding* `<div>` which acts a new positioning context. By enclosing them inside a positioned parent, we establish a *new* stacking order which is independent of either the root element or other positioning contexts on the page.

```
<div id="branding">
     <h1>Worrysome.com</h1>
     <blockquote><p>Don' worry 'bout a
thing...</p></blockquote>
     <p>Take the weight of the world off your
shoulders.</p>
 </div>
```

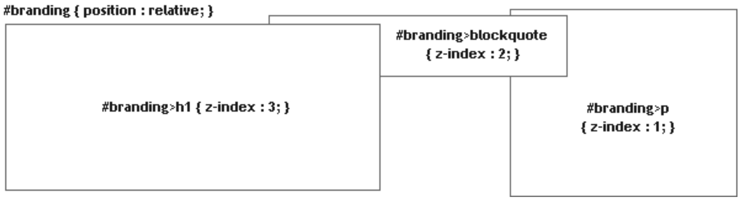
Applying a little positioning and z-index magic we can both set the position of these elements inside their positioning context and their stacking order. As we are going to use background images made from transparent PNGs, each element will allow another further down the stacking order to show through. This makes for some novel effects, particularly in liquid layouts.

(**Ed:** We're using *n* below to represent whatever values you require for your specific design.)

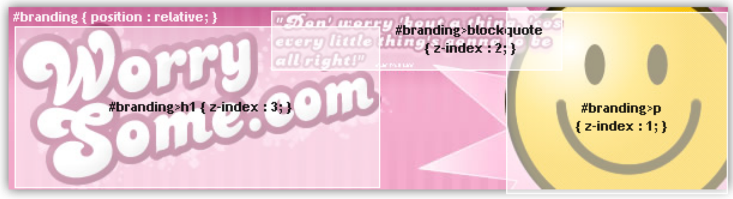
```
#branding {
    position: relative;
    width: n;
    height: n;
```

```
   background-image: url(n);
}

#branding>h1 {
   position: absolute;
   left: n;
   top: n;
   width: n;
   height: n;
   background-image: url(h1.png);
   text-indent: n;
}

#branding>blockquote {
   position: absolute;
   left: n;
   top: n;
   width: n;
   height: n;
   background-image: url(bq.png);
   text-indent: n;

}

#branding>p {
   position: absolute;
   right: n;
   top: n;
   width: n;
   height: n;
   background-image: url(p.png);
   text-indent: n;
}
```

Next we can begin to see how the three elements build upon each other.


#branding { position : relative; }
#branding>blockquote { z-index : 2; }
#branding>h1 { z-index : 3; }
#branding>p { z-index : 1; }

1. Elements outlined


#branding { position : relative; }
#branding>blockquote { z-index : 2; }
#branding>h1 { z-index : 3; }
#branding>p { z-index : 1; }
WorrySome.com

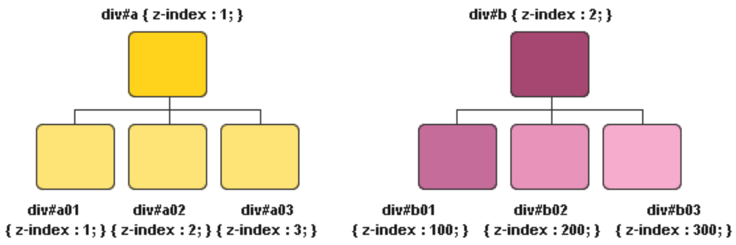
2. Positioned elements overlayed to show context

"Don' worry 'bout a thing, 'cos every little thing's gonna to be all right!"
WorrySome.com

3. Our final result

MULTIPLE STACKING ORDERS

Not only can elements *within* a positioning context be given a z-index, but those positioning contexts themselves can also be stacked.



Two positioning contexts, each with their own stacking order

Interestingly each stacking order is independent of that of either the root element or its siblings and we can exploit this to make complex layouts from just a few semantic elements. This technique was used heavily on my recent redesign of Karova.com.

DISSECTING PART OF KAROVA.COM

First the XHTML. The default template markup used for the site places `<div id="nav_main">` and `<div id="content">` as siblings inside their container.

```
<div id="container">
     <div id="content">
          <h2></h2>
          <p></p>
     </div>
     <div id="nav_main"></div>
 </div>
```

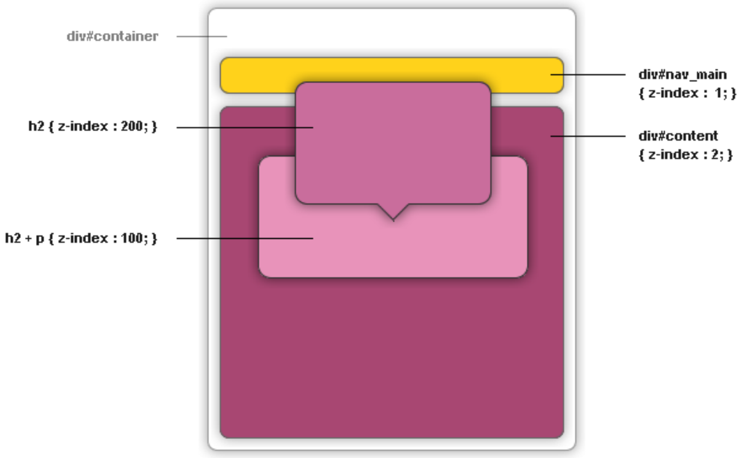
By giving the navigation `<div>` a lower z-index than the content `<div>` we can ensure that the positioned content elements will always appear closest to the viewer, despite the fact that the navigation comes after the content in the source.

```
#content {
     position: relative;
     z-index: 2;
 }

 #nav_main {
     position: absolute;
     z-index: 1;
 }
```

Now applying absolute positioning with a negative top value to the `<h2>` and a higher z-index value than the following `<p>` ensures that the header sits not only on top of the navigation but also the styled paragraph which follows it.

```
h2 {
     position: absolute;
     z-index: 200;
```

```
    top: -n;
}

h2+p {
    position: absolute;
    z-index: 100;
    margin-top: -n;
    padding-top: n;
}
```


div#container
div#nav_main { z-index : 1; }
h2 { z-index : 200; }
div#content { z-index : 2; }
h2 + p { z-index : 100; }

Dissecting part of Karova.com

You can see the full effect in the wild on the Karova.com site.

Have a great holiday season!

ABOUT THE AUTHOR

Andrew Clarke runs Stuff and Nonsense, a tiny web design company where they make fashionably flexible websites. Andrew's the author of Transcending CSS and Hardboiled Web Design and hosts the popular weekly podcast Unfinished Business where he discusses the business side of web, design and creative industries with his guests. He tweets as @malarkey.

17. Avoiding CSS Hacks for Internet Explorer

Kimberly Blessing 24ways.org/200517

Back in October, IEBlog issued a call to action, asking developers to clean up their CSS hacks for IE7 testing. Needless to say, a lot of hubbub ensued… both on IEBlog and elsewhere. My contribution to all of the noise was to suggest that developers review their code and use good CSS hacks. But what makes a good hack?

Tantek Çelik, the Godfather of CSS hacks, gave us the answer by explaining how CSS hacks should be designed. In short, they should (1) be valid, (2) target only old/frozen/abandoned user-agents/browsers, and (3) be ugly. Tantek also went on to explain that using a feature of CSS is not a hack.

Now, I'm not a frequent user of CSS hacks, but Tantek's post made sense to me. In particular, I felt it gave developers direction on how we should be coding to accommodate that sometimes troublesome browser, Internet Explorer. But what I've found, through my work with other developers, is that there is still much confusion over the use of CSS hacks and IE. Using examples from the code I've seen recently, allow me to demonstrate how to clean up some IE-specific CSS hacks.

The two hacks that I've found most often in the code I've seen and worked with are the star html bug and the underscore hack. We know these are both IE-specific by checking Kevin Smith's CSS Filters chart. Let's look at each of these hacks and see how we can replace them with the same CSS feature-based solution.

THE STAR HTML BUG

This hack violates Tantek's second rule as it targets current (and future) UAs. I've seen this both as a stand alone rule, as well as an override to some other rule in a style sheet. Here are some code samples:

```
* html div#header {margin-top:-3px;}
.promo h3 {min-height:21px;}
* html .promo h3 {height:21px;}
```

THE UNDERSCORE HACK

This hack violates Tantek's first two rules: it's invalid (according to the W3C CSS Validator) and it targets current UAs. Here's an example:

```
ol {padding:0; _padding-left:5px;}
```

USING CHILD SELECTORS

We can use the child selector to replace both the star html bug and underscore hack. Here's how:

1. Write rules with selectors that would be successfully applied to all browsers. This may mean starting with no declarations in your rule!

```
div#header {}
.promo h3 {}
ol {padding:0;}
```

2. To these rules, add the IE-specific declarations.

```
div#header {margin-top:-3px;}
.promo h3 {height:21px;}
ol {padding:0 0 0 5px;}
```

3. After each rule, add a second rule. The selector of the second rule must use a child selector. In this new rule, correct any IE-specific declarations previously made.

```
div#header {margin-top:-3px;}
body > div#header {margin-top:0;}

.promo h3 {height:21px;}
.promo > h3 {height:auto; min-height:21px;}

ol {padding:0 0 0 5px;}
html > body ol {padding:0;}
```

Voilà – no more hacks! There are a few caveats to this that I won't go into… but assuming you're operating in strict mode and barring any really complicated stuff you're doing in your code, your CSS will still render perfectly across browsers. And while this may make your CSS slightly heftier in size, it should future-proof it for IE7 (or so I hope). Happy holidays!

ABOUT THE AUTHOR

Kimberly Blessing has been developing Web sites since 1994 and has been a professional standards evangelist since 2000. She has worked for large companies like AOL and PayPal, leading their transitions to Web standards. She has also consulted for institutions large and small, helping them migrate to Web standards. She is a member and former Group Lead of the Web Standards Project and is active in other local, grass-roots Web standards efforts. (Geez, can we say “Web standards” any more in this bio?) An instructor in and a graduate of Bryn Mawr College‘s Computer Science program, Kimberly is also passionate about increasing the number of women in technology.

18. Introducing UDASSS!

Dustin Diaz 24ways.org/200518

Okay. What's that mean?

UNOBTRUSIVE DEGRADABLE AJAX STYLE SHEET SWITCHER!

Boy are you in for treat today 'cause we're gonna have a whole lotta Ajaxifida Unobtrucitosity CSS swappin' Fun!

Okay are you really kidding? Nope. I've even impressed myself on this one. Unfortunately, I don't have much time to tell you the ins and outs of what I actually did to get this to work. We're talking JavaScript, CSS, PHP…Ajax. But don't worry about that. I've always believed that a good A.P.I. is an invisible A.P.I… and this I felt I achieved. The only thing you need to know is **how it works and what to do**.

A QUICK INTRODUCTION ANYWAY…

First of all, the idea is very simple. I wanted something *just like* what Paul Sowden put together in Alternative Style: Working With Alternate Style Sheets from Alistapart Magazine **EXCEPT** a few minor (not-so-minor actually) differences which I've listed briefly below:

- Allow users to switch styles without JavaScript enabled (degradable)
- Preventing the F.O.U.C. before the window 'load' when getting preferred styles
- Keep the JavaScript entirely off our markup (no onclick's or onload's)
- Make it very very easy to implement (ok, Paul did that too)

What I did to achieve this was used server-side cookies instead of JavaScript cookies. Hence, PHP. However this isn't a "PHP style switcher" – which is where Ajax comes in. For the extreme technical folks, no, there is no xml involved here, or even a callback response. I only say Ajax because everyone knows what 'it' means. With that said, it's the Ajax that sets the cookies 'on the fly'. Got it? Awesome!

WHAT YOU NEED

Luckily, I've done the work for you. It's all packaged up in a nice zip file (at the end…keep reading for now) – so from here on out,
just follow these instructions

As I've mentioned, one of the things we'll be working with is PHP. So, first things first, open up a file called index and save it with a '.php' extension.

Next, place the following text at the top of your document (even above your `DOCTYPE`)

```
<?php
 require_once('utils/style-switcher.php');
 // style sheet path[, media, title, bool(set as
alternate)]
 $styleSheet = new AlternateStyles();
 $styleSheet->add('css/global.css','screen,projection');
// [Global Styles]
 $styleSheet->add('css/
preferred.css','screen,projection','Wog Standard'); //
[Preferred Styles]
 $styleSheet->add('css/
alternate.css','screen,projection','Tiny Fonts',true);
// [Alternate Styles]
 $styleSheet->add('css/
alternate2.css','screen,projection','Big O Fonts',true);
// // [Alternate Styles]
 $styleSheet->getPreferredStyles();
 ?>
```

The way this works is **REALLY EASY**. Pay attention closely.

Notice in the first line we've included our style-switcher.php file.

Next we instantiate a PHP class called `AlternateStyles()` which will allow us to configure our style sheets.
So for kicks, let's just call our object `$styleSheet`

As part of the AlternateStyles object, there lies a public method called `add`. So naturally with our `$styleSheet` object, we can call it to (da – da-da-da!) Add Style Sheets!

HOW THE `ADD()` METHOD WORKS

The `add` method takes in a possible **four arguments**, only one is required. However, you'll want to add some… since the whole point is working with alternate style sheets.

`$path` can simply be a uri, absolute, or relative path to your style sheet. `$media` adds a media attribute to your style sheets. `$title` gives a name to your style sheets (via title attribute).`$alternate` (which shows boolean) simply tells us that these are the **alternate style sheets**.

`ADD()` TIPS

For all global style sheets (meaning the ones that will always be seen and will not be swapped out), simply use the `add` method as shown next to `// [Global Styles]`.

To add preferred styles, do the same, but add a 'title'.

To add the alternate styles, do the same as what we've done to add preferred styles, but add the extra boolean and set it to true.

Note following when adding style sheets

- Multiple *global* style sheets are allowed
- **You can only have *one* preferred style sheet (That's a browser rule)**
- Feel free to add as many alternate style sheets as you like

MOVING ON

Simply add the following snippet to the `<head>` of your web document:

```
<script type="text/javascript" src="js/
prototype.js"></script>
 <script type="text/javascript" src="js/
common.js"></script>
 <script type="text/javascript" src="js/
alternateStyles.js"></script>
```

```
 <?php
 $styleSheet->drop();
 ?>
```

Nothing much to explain here. Just use your copy & paste powers.

HOW TO SWITCH STYLES

Whether you knew it or not, this baby already has the built in ‘ubobtrusive’ functionality that lets you switch styles by the drop of any link with a class name of ‘**altCss**‘. Just drop them where ever you like in your document as follows:

```
<a class="altCss" href="index.php?css=Bog_Standard">Bog
Standard</a>
 <a class="altCss"
href="index.php?css=Really_Small_Fonts">Small Fonts</a>
 <a class="altCss"
href="index.php?css=Large_Fonts">Large Fonts</a>
```

Take special note where the file is linking to. Yep. Just linking right back to the page we’re on. The only extra parameters we pass in is a variable called ‘css’ – and within that we append the names of our style sheets.

Also take very special note on the names of the style sheets have an under_score to take place of any spaces we might have.

Go ahead… play around and change the style sheet on the example page. Try disabling JavaScript and refreshing your browser. Still works!

COOL EH?

Well, I put this together in one night so it's still a work in progress and very beta. If you'd like to hear more about it and its future development, be sure stop on by my site where I'll definitely be maintaining it.

DOWNLOAD THE BETA ANYWAY

Well this wouldn't be fun if there was nothing to download. So we're hooking you up so you don't go home (or logoff) unhappy

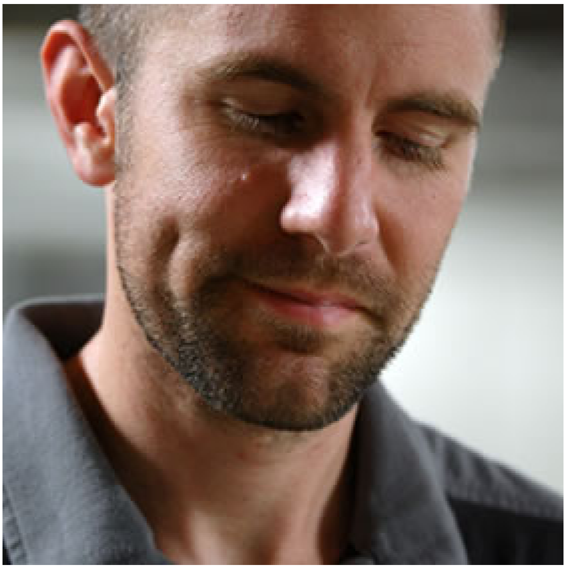
Download U.D.A.S.S.S | V0.8

MERRY CHRISTMAS!

Thanks for listening and I hope U.D.A.S.S.S. has been well worth your time and will bring many years of Ajaxy Style Switchin' Fun!

Many Blessings, Merry Christmas and have a great new year!

ABOUT THE AUTHOR



Dustin Diaz is a User Interface Engineer at Goooooogle who enjoys writing JavaScript, CSS, and HTML as well as making interactive and usable interfaces to create passionate users (real people (not fake ones)).

19. Tables with Style

Jonathan Snook 24ways.org/200519

It might not seem like it but styling tabular data can be a lot of fun. From a semantic point of view, there are plenty of elements to tie some style into. You have cells, rows, row groups and, of course, the table element itself. Adding CSS to a paragraph just isn't as exciting.

WHERE DO I START?

First, if you have some tabular data (you know, like a spreadsheet with rows and columns) that you'd like to spiffy up, pop it into a table — it's rightful place!

To add more semantics to your table — and coincidentally to add more hooks for CSS — break up your table into row groups. There are three types of row groups: the header (`thead`), the body (`tbody`) and the footer (`tfoot`). You can only have one header and one footer but you can have as many table bodies as is appropriate.

Sample table example

INSPIRATION

Table Striping

To improve scanning information within a table, a common technique is to style alternating rows. Also known as zebra tables. Whether you apply it using a class on every other row or turn to JavaScript to accomplish the task, a handy-dandy trick is to use a semi-transparent PNG as your background image. This is especially useful over patterned backgrounds.

```
tbody tr.odd td {
     background:transparent url(background.png) repeat
top left;
 }

 * html tbody tr.odd td {
     background:#C00;
     filter:
progid:DXImageTransform.Microsoft.AlphaImageLoader(
          src='background.png', sizingMethod='scale');
 }
```

We turn off the default background and apply our PNG hack to have this work in Internet Explorer.

Styling Columns

Did you know you could style a column? That's right. You can add special column (`col`) or column group (`colgroup`) elements. With that you can add border or background styles to the column.

```
<table>
     <col id="ingredients">
     <col id="serve12">
     <col id="serve24">
 ...
```

Check out the example.

Fun with Backgrounds

Pop in a tiled background to give your table some character! Internet Explorer's PNG hack unfortunately only works well when applied to a cell.

To figure out which background will appear over another, just remember the hierarchy:

(bottom) Table → Column → Row Group → Row → Cell (top)

THE FUTURE IS BRIGHT

Once browser-makers start implementing CSS3, we'll have more power at our disposal. Just with `:first-child` and `:last-child`, you can pull off a scalable version of our previous table with rounded corners and all — unfortunately, only Firefox manages to pull this one off successfully. And the selector the masses are clamouring for, nth-child, will make zebra tables easy as eggnog.

ABOUT THE AUTHOR

Jonathan Snook writes about tips, tricks, and bookmarks on his blog at Snook.ca. He has also written for A List Apart and .net magazine, and has co-authored two books, The Art and Science of CSS and Accelerated DOM Scripting. He has also authored and received world-wide acclaim for the self-published book, Scalable and Modular Architecture for CSS sharing his experience and best practices on CSS architecture.

Photo: Patrick H. Lauke

20. Naughty or Nice? CSS Background Images

Derek Featherstone 24ways.org/200520

Web Standards based development involves many things – using semantically sound HTML to provide structure to our documents or web applications, using CSS for presentation and layout, using JavaScript responsibly, and of course, ensuring that all that we do is accessible and interoperable to as many people and user agents as we can.

This we understand to be good.

And it is good.

Except when we don't clearly think through the full implications of using those techniques.

Which often happens when time is short and we need to get things done.

Here are some naughty examples of CSS background images with their nicer, more accessible counterparts.

TRANSACTION RELATED MESSAGES

I'm as guilty of this as others (or, perhaps, I'm the only one that has done this, in which case this can serve as my holiday season confessional) We use lovely little icons to show status messages for a transaction to indicate if the action was successful, or was there a warning or error? For example:

"Your postal/zip code was not in the correct format."

Notice that we place a nice little icon there, and use background colours and borders to convey a specific message: there was a problem that needs to be fixed. Notice that all of this visual information is now contained in the CSS rules for that div:

```
<div class="error">
     <p>Your postal/zip code was not in the correct
format.</p>
 </div>

 div.error {
     background: #ffcccc url(../images/error_small.png)
no-repeat 5px 4px;
     color: #900;
     border-top: 1px solid #c00;
     border-bottom: 1px solid #c00;
     padding: 0.25em 0.5em 0.25em 2.5em;
     font-weight: bold;
 }
```

Using this approach also makes it very easy to create a div.success and div.warning CSS rules meaning we have less to change in our HTML.

Nice, right?

No. Naughty.

VISUAL DESIGN COMMUNICATES

The CSS is being used to convey very specific information. The choice of icon, the choice of background colour and borders tell us visually that there is something wrong.

With the icon as a background image – there is no way to specify any alt text for the icon, and significant meaning is lost. A screen reader user, for example, misses the fact that it is an “error.”

The solution? Ask yourself: what is the bare minimum needed to indicate there was an error? Currently in the absence of CSS there will be no icon – which (I’m hoping you agree) is critical to communicating there was an error.

The icon should be considered content and not simply presentational.

The borders and background colour are certainly much less critical – they belong in the CSS.

Lets change the code to place the image directly in the HTML and using appropriate alt text to better communicate the meaning of the icon to *all* users:

```
<div class="bettererror">
     <img src="images/error_small.png" alt="Error" />
     <p>Your postal/zip code was not in the correct
format.</p>
 </div>

 div.bettererror {
     background-color: #ffcccc;
     color: #900;
     border-top: 1px solid #c00;
     border-bottom: 1px solid #c00;
     padding: 0.25em 0.5em 0.25em 2.5em;
     font-weight: bold;
     position: relative;
     min-height: 1.25em;
 }

 div.bettererror img {
     display: block;
     position: absolute;
     left: 0.25em;
     top: 0.25em;
     padding: 0;
     margin: 0;
 }

 div.bettererror p {
     position: absolute;
     left: 2.5em;
```

```
    padding: 0;
    margin: 0;
}
```

Compare these two examples of transactional messages

STATUS OF A RECORD

This example is pretty straightforward. Consider the following: a real estate listing on a web site. There are three "states" for a listing: new, normal, and sold. Here's how they look:

Example of a New Listing

Example of A Sold Listing

If we (forgive the pun) blindly apply the "use a CSS background image" technique we clearly run into problems with the new and sold images – they actually contain content with no way to specify an alternative when placed in the CSS.

In this case of the "new" image, we can use the same strategy as we used in the first example (the transaction result). The "new" image should be considered content and is placed in the HTML as part of the `<h2>...</h2>` that identifies the listing.

However when considering the “sold” listing, there are less changes to be made to keep the same look by leaving the “SOLD” image as a background image and providing the equivalent information elsewhere in the listing – namely, right in the heading.

For those that can’t see the background image, the status is communicated clearly and right away. A screen reader user that is navigating by heading or viewing a listing will know right away that a particular property is sold.

Of note here is that in both cases (new and sold) placing the status near the beginning of the record helps with a zoom layout as well.

Better Example of A Sold Listing

SUMMARY

Remember: in the holiday season, its what you give that counts!! Using CSS background images is easy and saves time for you but *think of the children*. And everyone else for that matter…

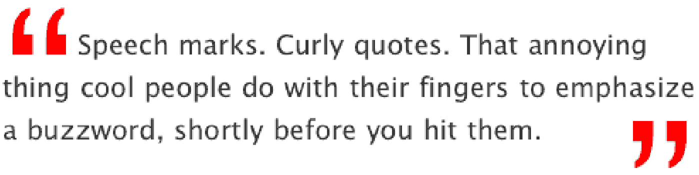
CSS background images should only be used for presentational images, not for those that contain content (unless that content is already represented and readily available elsewhere).

ABOUT THE AUTHOR

Derek Featherstone is a web developer and experienced accessibility consultant based in Ottawa, Canada where he runs Further Ahead. He serves as the Lead for the WaSP Accessibility Task Force. He is insane and thinks that somehow he'll manage to find time to train for an IronMan triathlon amidst work and family life with wife and three children. Insane.

21. Swooshy Curly Quotes Without Images

Simon Collison 24ways.org/200521


“ Speech marks. Curly quotes. That annoying thing cool people do with their fingers to emphasize a buzzword, shortly before you hit them. ”

The problem

Take a quote and render it within blockquote tags, applying big, funky and stylish curly quotes both at the beginning and the end without using any images – at all.

The traditional way

Feint background images under the text, or an image in the markup housed in a little float. Often designers only use the opening curly quote as it’s just too difficult to float a closing one.

Why is the traditional way bad?

Well, for a start there are no actual curly quotes in the text (unless you're doing some nifty image replacement). Thus with CSS disabled you'll only have default blockquote styling to fall back on. Secondly, images don't resize, so scaling text will have no affect on your graphic curlies.

The solution

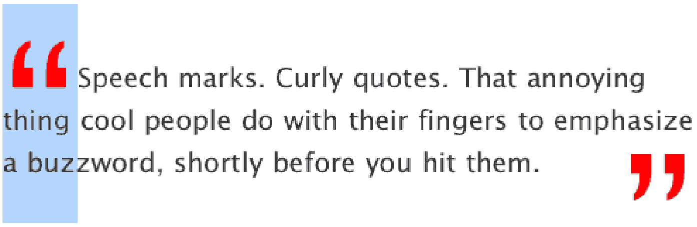
Use really big text. Then it can be resized by the browser, resized using CSS, and even be restyled with a new font style if you fancy it. It'll also make sense when CSS is unavailable.

The problem

Creating "Drop Caps" with CSS has been around for a while (Big Dan Cederholm discusses a neat solution in that first book of his), but drop caps are normal characters – the A to Z or 1 to 10 – and these can all be pulled into a set space and do not serve up a ton of whitespace, unlike punctuation characters.

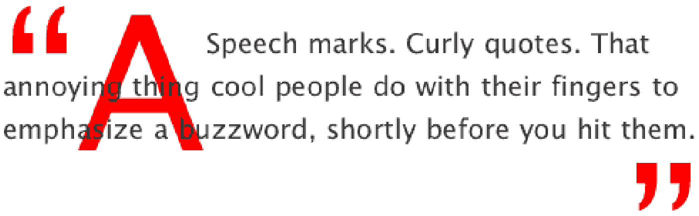
Curly quotes aren't like traditional characters. Like full stops, commas and hashes they float within the character space and leave lots of dead white space, making it bloody difficult to manipulate them with CSS. Styles generally fit around text, so cutting into that character is tricky indeed.

Also, all that extra white space is going to push into the quote text and make it look pretty uneven. This grab highlights the actual character space:



“Speech marks. Curly quotes. That annoying thing cool people do with their fingers to emphasize a buzzword, shortly before you hit them.”

See how this is emphasized when we add a normal alphabetical character within the span. This is what we’re dealing with here:



“A Speech marks. Curly quotes. That annoying thing cool people do with their fingers to emphasize a buzzword, shortly before you hit them.”

Then, there’s size. Call in a curly quote at less than 300% font-size and it ain’t gonna look very big. The white space it creates will be big enough, but the curlies will be way too small. We need more like 700% (as in this example) to make an impression, but that sure makes for a big character space.

Prepare the curlies

Firstly, remove the opening " from the quote. Replace it with the opening curly quote character entity “. Then replace the closing " with the entity reference for that, which is ”. Now at least the curlies will look nice and swooshy.

Add the hooks

Two reasons why we aren't using `:first-letter` pseudo class to manipulate the curlies. Firstly, only CSS2-friendly browsers would get what we're doing, and secondly we need to affect the last "letter" of our text also – the closing curly quote.

So, add a span around the opening curly, and a second span around the closing curly, giving complete control of the characters:

```
<blockquote><span class="bqstart">“</span>Speech marks.
Curly quotes. That annoying thing cool people do with
their fingers to emphasize a buzzword, shortly before
you hit them.<span class="bqend">”</span></blockquote>
```

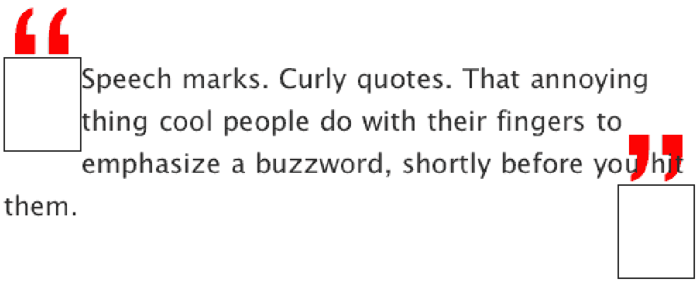
So far nothing will look any different, aside form the curlies looking a bit nicer. I know we've just added extra markup, but the benefits as far as accessibility are concerned are good enough for me, and of course there are no images to download.

The CSS

OK, easy stuff first. Our first rule `.bqstart` floats the span left, changes the color, and whacks the font-size up to an exuberant 700%. Our second rule `.bqend` does the same tricks aside from floating the curly to the right.

```
.bqstart {
     float: left;
     font-size: 700%;
     color: #FF0000;
 }

 .bqend {
     float: right;
     font-size: 700%;
     color: #FF0000;
 }
```

That gives us this, which is rubbish. I’ve highlighted the actual span area with outlines:


“Speech marks. Curly quotes. That annoying thing cool people do with their fingers to emphasize a buzzword, shortly before you hit ” them.

Note that the curlies don't even fit inside the span! At this stage on IE 6 PC you won't even see the quotes, as it only places focus on what it thinks is in the div. Also, the quote text is getting all spangled.

Fiddle with margin and padding

Think of that span outline box as a window, and that you need to position the curlies within that window in order to see them. By adding some small adjustments to the margin and padding it's possible to position the curlies exactly where you want them, and remove the excess white space by defining a height:

```
.bqstart {
     float: left;
     height: 45px;
     margin-top: -20px;
     padding-top: 45px;
     margin-bottom: -50px;
     font-size: 700%;
     color: #FF0000;
 }

 .bqend {
     float: right;
     height: 25px;
     margin-top: 0px;
     padding-top: 45px;
     font-size: 700%;
     color: #FF0000;
 }
```

I wanted the blocks of my curlies to align with the quote text, whereas you may want them to dig in or stick out more. Be aware however that my positioning works for IE PC and Mac, Firefox and Safari. Too much tweaking seems to break the magic in various browsers at various times. Now things are fitting beautifully:

I must admit that the heights, margins and spacing don't make a lot of sense if you analyze them. This was a real trial and error job. Get it working on Safari, and IE would fail. Sort IE, and Firefox would go weird.

Finished

The final thing looks ace, can be resized, looks cool without styles, and can be edited with CSS at any time. Here's a real example (note that I'm specifying Lucida Grande and then Verdana for my curlies):

> "Speech marks. Curly quotes. That annoying thing cool people do with their fingers to emphasize a buzzword, shortly before you hit them."

Browsers happy

As I said, too much tweaking of margins and padding can break the effect in some browsers. Even now, Firefox insists on dropping the closing curly by approximately 6 or 7 pixels, and if I adjust the padding for that, it'll crush it into the text on Safari or IE. Weird. Still, as I close now it seems solid through resizing tests on Safari, Firefox, Camino, Opera and IE PC and Mac. Lovely.

It's probably not perfect, but together we can beat the evil typographic limitations of the web and walk together towards a brighter, more aligned world. Merry Christmas.

ABOUT THE AUTHOR

Simon Collison is a designer, author and speaker with a decade of experience at the sharp end. He co-founded Erskine Design back in 2006, but left in early 2010 to pursue new and exciting challenges, including writing an ambitious new book, and organising the New Adventures in Web Design event. Simon has lived in London and Reykjavik, but now lives back in his hometown of Nottingham, where he is owned by a cat.

Photo: Lachlan Hardy

22. Debugging CSS with the DOM Inspector

Jon Hicks 24ways.org/200522

AN INSPECTOR CALLS

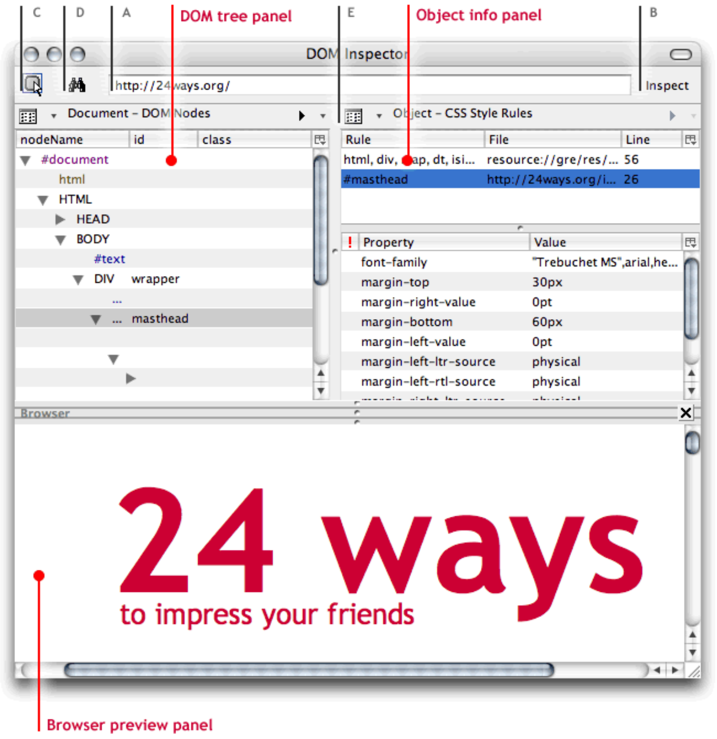
The larger your site and your CSS becomes, the more likely that you will run into bizarre, inexplicable problems. Why does that heading have all that extra padding? Why is my text the wrong colour? Why does my navigation have a large moose dressed as Noel Coward on top of all the links?

Perhaps you work in a collaborative environment, where developers and other designers are adding code? In which case, the likelihood of CSS strangeness is higher.

You need to debug. You need Firefox's wise-guy know-it-all, the DOM Inspector.

The DOM Inspector knows where *everything* is in your layout, and more importantly, what causes it to look the way it does. So without further ado, load up any css based site in your copy of Firefox (or Flock for that matter), and launch the DOM Inspector from the Tools menu.

The inspector uses two main panels – the left to show the DOM tree of the page, and the right to show you detail:



nodeName	id	class
#document		
html		
HTML		
HEAD		
BODY		
#text		
DIV	wrapper	
...		
...	masthead	

Rule	File	Line
html, div, map, dt, isi...	resource://gre/res/...	56
#masthead	http://24ways.org/i...	26

!	Property	Value
	font-family	"Trebuchet MS",arial,he...
	margin-top	30px
	margin-right-value	0pt
	margin-bottom	60px
	margin-left-value	0pt
	margin-left-ltr-source	physical
	margin-left-rtl-source	physical

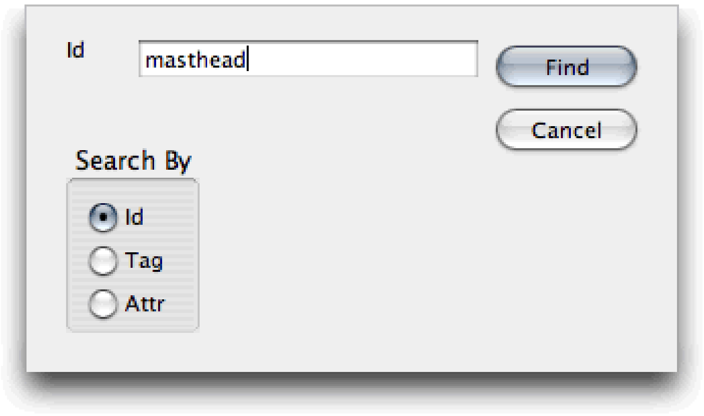
The Inspector will look at whatever site is in the front-most window or tab, but you can also use it without another window. Type in a URL at the top (A), press 'Inspect' (B) and a third panel appears at the bottom, with the browser view. I find this layout handier than looking at a window behind the DOM Inspector.

STEP 1 – FIND YOUR NODE!

Each element on your page – be it a HTML tag or a piece of text, is called a ‘node’ of the DOM tree. These nodes are all listed in the left hand panel, with any ID or CLASS attribute values next to them. When you first look at a page, you won’t see all those yet. Nested HTML elements (such as a link inside a paragraph) have a reveal triangle next to their name, clicking this takes you one level further down.

This can be fine for finding the node you want to look at, but there are easier ways. Say you have a complex rounded box technique that involves 6 nested DIVs? You’d soon get tired of clicking all those triangles to find the element you want to inspect. Click the top left icon © – “Find a node to inspect by clicking on it” and then select the area you want to inspect. Boom! All that drilling down the DOM tree has been done for you! Huzzah!

If you’re looking for an element that you know has an ID (such as `<ul id="navigation">`), or a specific HTML tag or attribute, click the binoculars icon (D) for “Finds a node to inspect by ID, tag or attribute” (You can also use Ctrl-F or Apple-F to do this if the DOM Inspector is the frontmost window) :

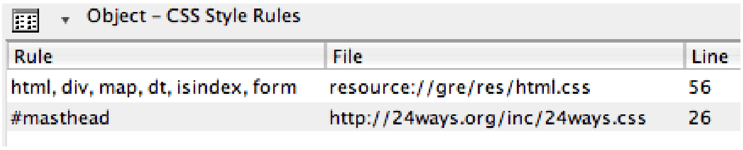


Type in the name and Bam! You're there! Pressing F3 will take you to any other instances. Notice also, that when you click on a node in the inspector, it highlights where it is in the browser view with a flashing red border!

Now that we've found the troublesome node on the page, we can find out what's up with it…

STEP 2 – DEBUG THAT NODE!

Once the node is selected, we move over to the right hand panel. Choose 'CSS Style Rules' from the document menu (E), and all the CSS rules that apply to that node are revealed, in the order that they load:



Object – CSS Style Rules

Rule	File	Line
html, div, map, dt, isindex, form	resource://gre/res/html.css	56
#masthead	http://24ways.org/inc/24ways.css	26

You'll notice that right at the top, there is a CSS file you might not recognise, with a file path beginning with "resource://". This is the browsers default CSS, that creates the basic rendering. You can mostly ignore this, especially if use the star selector method of resetting default browser styles.

Your style sheets come next. See how helpful it is? It even tells you the line number where to find the related CSS rules! If you use CSS shorthand, you'll notice that the values are split up (e.g margin-left, margin-right etc.).

Now that you can see all the style rules affecting that node, the rest is up to you! Happy debugging!

ABOUT THE AUTHOR

Jon Hicks is one half of the creative partnership Hicksdesign, designing for a variety of mediums, but with a particular fondness for icon and logo design. In fact he's written a book, about it called The Icon Handbook, released in January 2012. His recent clients include Skype, Mailchimp, Shopify and Opera Software, but is best known for his uncanny impression of Lucius Malfoy singing "I only want to be with you".

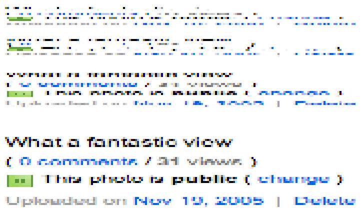
He blogs about design and personal interests (mainly Dr Who and Cycling) at hicksdesign.co.uk/journal

23. Edit-in-Place with Ajax

Drew McLellan 24ways.org/200523

Back on day one we looked at using the Prototype library to take all the hard work out of making a simple Ajax call. While that was fun and all, it didn't go that far towards implementing something really practical. We dipped our toes in, but haven't learned to swim yet.

So here is swimming lesson number one. Anyone who's used Flickr to publish their photos will be familiar with the edit-in-place system used for quickly amending titles and descriptions on photographs. Hovering over an item turns its background yellow to indicate it is editable. A simple click loads the text into an edit box, right there on the page.



[illegible]

What a fantastic view
(0 comments / 31 views)
This photo is public (change)
Uploaded on Nov 19, 2005 | Delete

Prototype includes all sorts of useful methods to help reproduce something like this for our own projects. As well as the simple Ajax GETs we learned how to do last time, we can also do POSTs (which we'll need here) and a whole bunch of manipulations to the user interface – all through simple library calls. Here's what we're building, so let's do it.

GETTING STARTED

There are two major components to this process; the user interface manipulation and the Ajax call itself. Our set-up is much the same as last time (you may wish to read the first article if you've not already done so). We have a basic HTML page which links in the `prototype.js` file and our own `editinplace.js`. Here's what Santa dropped down my chimney:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en" lang="en">
 <head>
```

```
     <meta http-equiv="Content-Type" content="text/html;
charset=utf-8"/>
    <title>Edit-in-Place with Ajax</title>
     <link href="editinplace.css" rel="Stylesheet"
type="text/css" />
     <script src="prototype.js" type="text/
javascript"></script>
     <script src="editinplace.js" type="text/
javascript"></script>
</head>
<body>
     <h1>Edit-in-place</h1>
     <p id="desc">Dashing through the snow on a one
horse open sleigh.</p>
 </body>
 </html>
```

So that's our page. The editable item is going to be the `<p>` called `desc`. The process goes something like this:

1. Highlight the area `onMouseOver`
2. Clear the highlight `onMouseOut`
3. If the user clicks, hide the area and replace with a `<textarea>` and buttons
4. Remove all of the above if the user cancels the operation
5. When the Save button is clicked, make an Ajax POST and show that something's happening
6. When the Ajax call comes back, update the page with the new content

EVENTS AND HIGHLIGHTING

The first step is to offer feedback to the user that the item is editable. This is done by shading the background colour when the user mouses over. Of course, the CSS `:hover` pseudo class is a straightforward way to do this, but for three reasons, I'm using JavaScript to switch class names.

1. `:hover` isn't supported on many elements in Internet Explorer for Windows
2. I want to keep control over when the highlight switches off after an update, regardless of mouse position
3. If JavaScript isn't available we don't want to end up with the CSS suggesting it might be

With this in mind, here's how `editinplace.js` starts:

```
Event.observe(window, 'load', init, false);

 function init(){
     makeEditable('desc');
 }

 function makeEditable(id){
     Event.observe(id, 'click', function(){edit($(id))},
false);
     Event.observe(id, 'mouseover',
function(){showAsEditable($(id))}, false);
     Event.observe(id, 'mouseout',
function(){showAsEditable($(id), true)}, false);
 }
```

```
function showAsEditable(obj, clear){
    if (!clear){
        Element.addClassName(obj, 'editable');
    }else{
        Element.removeClassName(obj, 'editable');
    }
}
```

The first line attaches an `onLoad` event to the window, so that the function `init()` gets called once the page has loaded. In turn, `init()` sets up all the items on the page that we want to make editable. In this example I've just got one, but you can add as many as you like.

The function `madeEditable()` attaches the mouseover, mouseout and click events to the item we're making editable. All `showAsEditable` does is add and remove the class name `editable` from the object. This uses the particularly cunning methods `Element.addClassName()` and `Element.removeClassName()` which enable you to cleanly add and remove effects without affecting any styling the object may otherwise have.

Oh, remember to add a rule for `.editable` to your style sheet:

```
.editable{
    color: #000;
    background-color: #ffffd3;
}
```

THE SWITCH

As you can see above, when the user clicks on an editable item, a call is made to the function `edit()`. This is where we switch out the static item for a nice editable textarea. Here’s how that function looks.

```
function edit(obj){
     Element.hide(obj);

     var textarea ='<div id="' + obj.id + '_editor">
          <textarea id="' + obj.id + '_edit" name="' +
obj.id + '" rows="4" cols="60">'
          + obj.innerHTML + '</textarea>';

     var button = '<input id="' + obj.id + '_save"
type="button" value="SAVE" /> OR
          <input id="' + obj.id + '_cancel"
type="button" value="CANCEL" /></div>';

     new Insertion.After(obj, textarea+button);

     Event.observe(obj.id+'_save', 'click',
function(){saveChanges(obj)}, false);
     Event.observe(obj.id+'_cancel', 'click',
function(){cleanUp(obj)}, false);

 }
```

The first thing to do is to hide the object. Prototype comes to the rescue with `Element.hide()` (and of course, `Element.show()` too). Following that, we build up the textarea and buttons as a string, and then use `Insertion.After()` to place our new editor underneath the (now hidden) editable object.

The last thing to do before we leave the user to edit is it attach listeners to the Save and Cancel buttons to call either the `saveChanges()` function, or to `cleanUp()` after a cancel.

In the event of a cancel, we can clean up behind ourselves like so:

```
function cleanUp(obj, keepEditable){
     Element.remove(obj.id+'_editor');
     Element.show(obj);
     if (!keepEditable) showAsEditable(obj, true);
 }
```

SAVING THE CHANGES

This is where all the Ajax fun occurs. Whilst the previous article introduced `Ajax.Updater()` for simple Ajax calls, in this case we need a little bit more control over what happens once the response is received. For this purpose, `Ajax.Request()` is perfect. We can use the `onSuccess` and `onFailure` parameters to register functions to handle the response.

```
function saveChanges(obj){
     var new_content = escape($F(obj.id+'_edit'));
     obj.innerHTML = "Saving...";
     cleanUp(obj, true);

     var success = function(t){editComplete(t, obj);}
     var failure = function(t){editFailed(t, obj);}

     var url = 'edit.php';
     var pars = 'id=' + obj.id + '&content=' +
new_content;
     var myAjax = new Ajax.Request(url, {method:'post',
          postBody:pars, onSuccess:success,
onFailure:failure});
 }

 function editComplete(t, obj){
     obj.innerHTML = t.responseText;
     showAsEditable(obj, true);
 }

 function editFailed(t, obj){
     obj.innerHTML = 'Sorry, the update failed.';
     cleanUp(obj);
 }
```

As you can see, we first grab in the contents of the textarea into the variable `new_content`. We then remove the editor, set the content of the original object to "Saving…" to show that an update is occurring, and make the Ajax POST.

If the Ajax fails, `editFailed()` sets the contents of the object to "Sorry, the update failed." Admittedly, that's not a very helpful way to handle the error but I have to limit the scope of this article somewhere. It might be a good idea to stow away the original contents of the object (`obj.preUpdate = obj.innerHTML`) for later retrieval *before* setting the content to "Saving…". No one likes a failure – especially a messy one.

If the Ajax call is successful, the server-side script returns the edited content, which we then place back inside the object from `editComplete`, and tidy up.

MEANWHILE, BACK AT THE SERVER

The missing piece of the puzzle is the server-side script for committing the changes to your database. Obviously, any solution I provide here is not going to fit your particular application. For the purposes of getting a functional demo going, here's what I have in PHP.

```
<?php
     $id = $_POST['id'];
     $content = $_POST['content'];
     echo htmlspecialchars($content);
 ?>
```

Not exactly rocket science is it? I'm just catching the `content` item from the POST and echoing it back. For your application to be useful, however, you'll need to know exactly which record you should be updating. I'm passing in the ID of my `<div>`, which is not a fat lot of use. You can modify `saveChanges()` to post back whatever information your app needs to know in order to process the update.

You should also check the user's credentials to make sure they have permission to edit whatever it is they're editing. Basically the same rules apply as with any script in your application.

LIMITATIONS

There are a few bits and bobs that in an ideal world I would tidy up. The first is the error handling, as I've already mentioned. The second is that from an idealistic standpoint, I'd rather not be using `innerHTML`. However, the reality is that it's presently the most efficient way of making large changes to the document. If you're serving as XML, remember that you'll need to replace these with proper DOM nodes.

It's also important to note that it's quite difficult to make something like this universally accessible. Whenever you start updating large chunks of a document based on user interaction, a lot of non-traditional devices don't cope well. The benefit of this technique, though, is that if JavaScript is unavailable none of the functionality gets implemented at all – it fails silently. It is for this reason that **this shouldn't be used as a complete replacement**

for a traditional, universally accessible edit form. It's a great time-saver for those with the ability to use it, but it's no replacement.

SEE IT IN ACTION

I've put together an example page using the inert PHP script above. That is to say, your edits aren't committed to a database, so the example is reset when the page is reloaded.

ABOUT THE AUTHOR

Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency edgeofmyseat.com, and formerly Group Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a personal site covering web development issues and themes, takes photos and tweets a lot.

24. Have Your DOM and Script It Too

Shaun Inman 24ways.org/200524

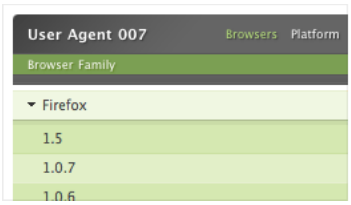
When working with the `XMLHttpRequest` object it appears you can only go one of three ways:

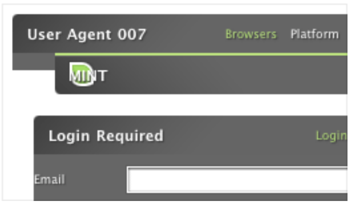
1. You can stay true to the colorful moniker du jour and stick strictly to the `responseXML` property
2. You can play with proprietary – yet widely supported – fire and inject the value of responseText property into the `innerHTML` of an element of your choosing
3. Or you can be eval() and parse JSON or arbitrary JavaScript delivered via `responseText`

But did you know that there's a fourth option giving you the best of the latter two worlds? Mint uses this unmentioned approach to grab fresh HTML and run arbitrary JavaScript simultaneously. **Without** relying on `eval()`. "But wait-", you might say, "when would I need to do this?" Besides the example below this technique is handy for things like tab groups that need initialization `onload` but miss the main `onload` event handler by a mile thanks to asynchronous scripting.

CONSIDER THE PROBLEM

Originally Mint used option 2 to refresh or load new tabs into individual Pepper panes without requiring a full roundtrip to the server. This was all well and good until I introduced the new Client Mode which when enabled allows anyone to view a Mint installation without being logged in. If voyeurs are afoot as Client Mode is disabled, the next time they refresh a pane the entire login page is inserted into the current document. That's not very helpful so I needed a way to redirect the current document to the login page.





ENTER THE SOLUTION

Wouldn't it be cool if browsers interpreted the contents of script tags crammed into `innerHTML`? Sure, but unfortunately, that just wasn't meant to be. However like the body element, image elements have an onload event handler. When the image has fully loaded the handler runs the code applied to it. See where I'm going with this?

By tacking a tiny image (think single pixel, transparent spacer gif – ***shudder***) onto the end of the HTML returned by our Ajax call, we can smuggle our arbitrary JavaScript into the existing document. The image is added to the DOM, and our stowaway can go to town.

```
<p>This is the results of our Ajax call.</p>
 <img src="../images/loaded.gif" alt=""
onload="alert('Now that I have your attention...');" />
```

PLEASE BE NEAT

So we've just jammed some meaningless cruft into our DOM. If our script does anything with images this addition could have some unexpected side effects. (Remember The Fly?) So in order to save that poor, unsuspecting element whose `innerHTML` we just swapped out from sharing Jeff Goldblum's terrible fate we should tidy up after ourselves. And by using the `removeChild` method we do just that.

```
<p>This is the results of our Ajax call.</p>
 <img src="../images/loaded.gif" alt=""
     onload="alert('Now that I have your
attention...');this.parentNode.removeChild(this);" />
```

ABOUT THE AUTHOR

Shaun Inman designed and developed Mint, the curiously successful web site analytic tool. He passes the time (literally) tinkering on ShaunInman.com while nervously eyeing the dust gathering on Designologue.