24

2007

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.

2007

Apple launched the iPhone in June; Amazon released the Kindle in November — a big year. At three, 24 ways was as diverse as ever, taking a detailed look at font stacks, website performance, working with clients and markup.

1. Transparent PNGs in Internet Explorer 6

Drew McLellan 24ways.org/200701

Newer breeds of browser such as Firefox and Safari have offered support for PNG images with full alpha channel transparency for a few years. With the use of hacks, support has been available in Internet Explorer 5.5 and 6, but the hacks are non-ideal and have been tricky to use. With IE7 winning masses of users from earlier versions over the last year, full PNG alpha-channel transparency is becoming more of a reality for day-to-day use.

However, there are still numbers of IE6 users out there who we can't leave out in the cold this Christmas, so in this article I'm going to look what we can do to support IE6 users whilst taking full advantage of transparency for the majority of a site's visitors.

SO WHAT'S ALPHA CHANNEL TRANSPARENCY?

Cast your minds back to the Ghost of Christmas Past, the humble GIF. Images in GIF format offer transparency, but that transparency is either on or off for any given pixel. Each pixel's either fully transparent, or a solid colour. In GIF, transparency is effectively just a special colour you can chose for a pixel.

The PNG format tackles the problem rather differently. As well as having any colour you chose, each pixel also carries a separate channel of information detailing how transparent it is. This alpha channel enables a pixel to be fully transparent, fully opaque, or critically, *any step in between*.

This enables designers to produce images that can have, for example, soft edges without any of the 'halo effect' traditionally associated with GIF transparency. If you've ever worked on a site that has different colour schemes and therefore requires multiple versions of each graphic against a different colour, you'll immediately see the benefit.

What's perhaps more interesting than that, however, is the extra creative freedom this gives designers in creating beautiful sites that can remain web-like in their ability to adjust, scale and reflow.

THE INTERNET EXPLORER PROBLEM

Up until IE7, there has been no fully native support for PNG alpha channel transparency in Internet Explorer. However, since IE5.5 there has been some support in the form of proprietary filter called the AlphaImageLoader. Internet Explorer filters can be applied directly in your CSS (for both inline and background images), or by setting the same CSS property with JavaScript.

CSS:

```
img {
  filter:
progid:DXImageTransform.Microsoft.AlphaImageLoader(...);
}
```

JavaScript:

```
img.style.filter =
"progid:DXImageTransform.Microsoft.AlphaImageLoader(...)";
```

That may sound like a problem solved, but all is not as it may appear. Firstly, as you may realise, there's no CSS property called `filter` in the W3C CSS spec. It's a proprietary extension added by Microsoft that could potentially cause other browsers to reject your entire CSS rule.

Secondly, AlphaImageLoader does not magically add full PNG transparency support so that a PNG in the page will just start working. Instead, when applied to an element in the page, it draws a new rendering surface in the same space that element occupies and loads a PNG into it. If that sounds weird, it's because that's precisely what it is. However, by and large the result is that PNGs with an alpha channel can be accommodated.

THE PITFALLS

So, whilst support for PNG transparency in IE5.5 and 6 is possible, it's not without its problems.

Background images cannot be positioned or repeated

The AlphaImageLoader does work for background images, but only for the simplest of cases. If your design requires the image to be tiled (`background-repeat`) or positioned (`background-position`) you're out of luck. The AlphaImageLoader allows you to set a `sizingMethod` to either `crop` the image (if necessary) or to `scale` it to fit. Not massively useful, but something at least.

Delayed loading and resource use

The AlphaImageLoader can be quite slow to load, and appears to consume more resources than a standard image when applied. Typically, you'd need to add thousands of GIFs or JPEGs to a page before you saw any noticeable impact on the browser, but with the

AlphaImageLoader filter applied Internet Explorer can become sluggish after just a handful of alpha channel PNGs.

The other noticeable effect is that as more instances of the AlphaImageLoader are applied, the longer it takes to render the PNGs with their transparency. The user sees the PNG load in its original non-supported state (with black or grey areas where transparency should be) before one by one the filter kicks in and makes them properly transparent.

Both the issue of sluggish behaviour and delayed load only really manifest themselves with volume and size of image. Use just a couple of instances and it’s fine, but be careful adding more than five or six. As ever, test, test, test.

Links become unclickable, forms unfocusable

This is a big one. There’s a bug/weirdness with AlphaImageLoader that sometimes prevents interaction with links and forms when a PNG background image is used. This is sometimes reported as a `z-index` issue, but I don’t believe it is. Rather, it’s an artefact of that weird way the filter gets applied to the document almost outside of the normal render process.

Often this can be solved by giving the links or form elements `hasLayout` using `position: relative;` where possible. However, this doesn't always work and the non-interaction problem **cannot always be solved**. You may find yourself having to go back to the drawing board.

SIDESTEPPING THE DANGER ZONES

Frankly, it's pretty bad news if you design a site, have that design signed off by your client, build it and then find out only at the end (because you don't know what might trigger a problem) that your search field can't be focused in IE6. That's an absolute nightmare, and whilst it's not likely to happen, it's possible that it might. It's happened to me. So what can you do?

The best approach I've found to this scenario is

1. Isolate the PNG or PNGs that are causing the problem. Step through the PNGs in your page, commenting them out one by one and retesting. Typically it'll be the nearest PNG to the problem, so try there first. Keep going until you can click your links or focus your form fields.
2. This is where you really need luck on your side, because you're going to have to fake it. This will depend on the design of the site, but some way or other create a replacement GIF or JPEG image that will give you an acceptable result. Then use conditional comments to serve that image to only users of IE older than version 7.

A hack, you say? Well, you started it chum.

APPLYING ALPHAIMAGELOADER

Because the `filter` property is invalid CSS, the safest pragmatic approach is to apply it selectively with JavaScript for only Internet Explorer versions 5.5 and 6. This helps ensure that by default you're serving standard CSS to browsers that support both the CSS and PNG standards correct, and then selectively patching up only the browsers that need it.

Several years ago, Aaron Boodman wrote and released a script called sleight for doing just that. However, sleight dealt only with images in the page, and not background images applied with CSS. Building on top of Aaron's work, I hacked sleight and came up with bgsleight for applying the filter to background images instead. That was in 2003, and over the years I've made a couple of improvements here and there to keep it ticking over and to resolve conflicts between sleight and bgsleight when used together. However, with alpha channel PNGs becoming much more widespread, it's time for a new version.

INTRODUCING SUPERSLEIGHT

SuperSleight adds a number of new and useful features that have come from the day-to-day needs of working with PNGs.

- Works with both inline and background images, replacing the need for both sleight and bgsleight
- Will automatically apply `position: relative` to links and form fields if they don't already have `position` set. (Can be disabled.)
- Can be run on the entire document, or just a selected part where you know the PNGs are. This is better for performance.
- Detects background images set to `no-repeat` and sets the `scaleMode` to `crop` rather than `scale`.
- Can be re-applied by any other JavaScript in the page – useful if new content has been loaded by an Ajax request.

Download SuperSleight

Implementation

Getting SuperSleight running on a page is quite straightforward, you just need to link the supplied JavaScript file (or the minified version if you prefer) into your document **inside conditional comments** so that it is delivered to only Internet Explorer 6 or older.

```
<!--[if lte IE 6]>
  <script type="text/javascript"
src="supersleight-min.js"></script>
<![endif]-->
```

Supplied with the JavaScript is a simple transparent GIF file. The script replaces the existing PNG with this before re-layering the PNG over the top using AlphaImageLoaded. You can change the name or path of the image in the top of the JavaScript file, where you'll also find the option to turn off the adding of `position: relative` to links and fields if you don't want that.

The script is kicked off with a call to `supersleight.init()` at the bottom. The scope of the script can be limited to just one part of the page by passing an ID of an element to `supersleight.limitTo()`. And that's all there is to it.

Update March 2008: a version of this script as a jQuery plugin is also now available.

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. Get To Grips with Slippy Maps

Andrew Turner 24ways.org/200702

Online mapping has definitely hit mainstream. Google Maps made ‘slippy maps’ popular and made it easy for any developer to quickly add a dynamic map to his or her website. You can now find maps for store locations, friends nearby, upcoming events, and embedded in blogs.

In this tutorial we’ll show you how to easily add a map to your site using the Mapstraction mapping library. There are many map providers available to choose from, each with slightly different functionality, design, and terms of service. Mapstraction makes deciding which provider to use easy by allowing you to write your mapping code once, and then easily switch providers.

ASSEMBLE THE PIECES

Utilizing any of the mapping library typically consists of similar overall steps:

1. Create an HTML div to hold the map
2. Include the Javascript libraries
3. Create the Javascript Map element
4. Set the initial map center and zoom level
5. Add markers, lines, overlays and more

CREATE THE MAP DIV

The HTML `div` is where the map will actually show up on your page. It needs to have a unique `id`, because we'll refer to that later to actually put the map here. This also lets you have multiple maps on a page, by creating individual `div`s and Javascript map elements. The size of the `div` also sets the height and width of the map. You set the size using CSS, either inline with the element, or via a CSS reference to the element `id` or class. For this example, we'll use inline styling.

```
<div id="map" style="width: 400px; height: 400px;"></div>
```

INCLUDE JAVASCRIPT LIBRARIES

A mapping library is like any Javascript library. You need to include the library in your page before you use the methods of that library. For our tutorial, we'll need to include at least two libraries: Mapstraction, and the mapping API(s) we want to display. Our first example we'll use the ubiquitous Google Maps library. However, you can just as easily include Yahoo, MapQuest, or any of the other supported libraries.

Another important aspect of the mapping libraries is that many of them require an API key. You will need to agree to the terms of service, and get an API key these.

```
<script src="http://maps.google.com/
maps?file=api&v=2&key=YOUR_KEY" type="text/
javascript"></script>
<script type="text/javascript"
src="http://mapstraction.com/src/
mapstraction.js"></script>
```

CREATE THE MAP

Great, we've now put in all the pieces we need to start actually creating our map. This is as simple as creating a new Mapstraction object with the `id` of the HTML `div` we created earlier, and the name of the mapping provider we want to use for this map.

With several of the mapping libraries you will need to set the map center and zoom level before the map will appear. The map centering actually triggers the initialization of the map.

```
var mapstraction = new Mapstraction('map','google');
var myPoint = new LatLonPoint(37.404,-122.008);
mapstraction.setCenterAndZoom(myPoint, 10);
```

A note about zoom levels. The `setCenterAndZoom` function takes two parameters, the center as a `LatLonPoint`, and a zoom level that has been defined by mapping libraries. The current usage is for zoom level 1 to be "zoomed out", or view the entire earth – and increasing the zoom level as you zoom in. Typically 17 is the maximum zoom, which is about the size of a house.

Different mapping providers have different quality of zoomed in maps over different parts of the world. This is a perfect reason why using a library like Mapstraction is very useful, because you can quickly change mapping providers to accommodate users in areas that have bad coverage with some maps.

To switch providers, you just need to include the Javascript library, and then change the second parameter in the Mapstraction creation. Or, you can call the `switch` method to dynamically switch the provider.

So for Yahoo Maps (demo):

```
var mapstraction = new Mapstraction('map','yahoo');
```

or Microsoft Maps (demo):

```
var mapstraction = new Mapstraction('map','microsoft');
```

want a 3D globe in your browser? try FreeEarth (demo):

```
var mapstraction = new Mapstraction('map','freeearth');
```

or even OpenStreetMap (free your data!) (demo):

```
var mapstraction = new
Mapstraction('map','openstreetmap');
```

Visit the Mapstraction multiple map demo page for an example of how easy it is to have many maps on your page, each with a different provider.

ADDING MARKERS

While adding your first map is fun, and you can probably spend hours just sliding around, the point of adding a map to your site is usually to show the location of something. So now you want to add some markers. There are a couple of ways to add to your map.

The simplest is directly creating markers. You could either hard code this into a rather static page, or dynamically generate these using whatever tools your site is built on.

```
var marker = new Marker( new
LatLonPoint(37.404,-122.008) );
marker.setInfoBubble("It's easy to add maps to your
site");
mapstraction.addMarker( marker );
```

There is a lot more you can do with markers, including changing the icon, adding timestamps, automatically opening the bubble, or making them draggable.

While it is straight-forward to create markers one by one, there is a much easier way to create a large set of markers. And chances are, you can make it very easy by extending some data you already are sharing: RSS.

Specifically, using GeoRSS you can easily add a large set of markers directly to a map. GeoRSS is a community built standard (like Microformats) that added geographic markup to RSS and Atom entries. It's as simple as adding `<georss:point>42 -83</georss:point>` to your feeds to share items via GeoRSS. Once you've done that, you can add that feed as an 'overlay' to your map using the function:

```
mapstraction.addOverlay("http://api.flickr.com/services/
feeds/
groups_pool.gne?id=322338@N20&format=rss_200&georss=1");
```

Mapstraction also supports KML for many of the mapping providers. So it's easy to add various data sources together with your own data. Check out Mapufacture for a growing index of available GeoRSS feeds and KML documents.

PLAY WITH YOUR NEW TOYS

Mapstraction offers a lot more functionality you can utilize for demonstrating a lot of geographic data on your website. It also includes geocoding and routing abstraction layers for making sure your users know where to go. You can see more on the Mapstraction website: http://mapstraction.com.

ABOUT THE AUTHOR

Andrew Turner is a neogeographer and co-founder of Mapufacture, a personalizable geospatial search and aggregation company. He helps expand the GeoWeb by advocating open standards and developing tools such as GeoPress to make it easy to add location to your blog or CMS. Andrew also wrote O'Reilly's Introduction to Neogeography.

3. The Neverending (Background Image) Story

Elliot Jay Stocks 24ways.org/200703

Everyone likes candy for Christmas, and there's none better than eye candy. Well, that, and just more of the stuff. Today we're going to combine both of those good points and look at how to create a beautiful background image that goes on and on... forever!

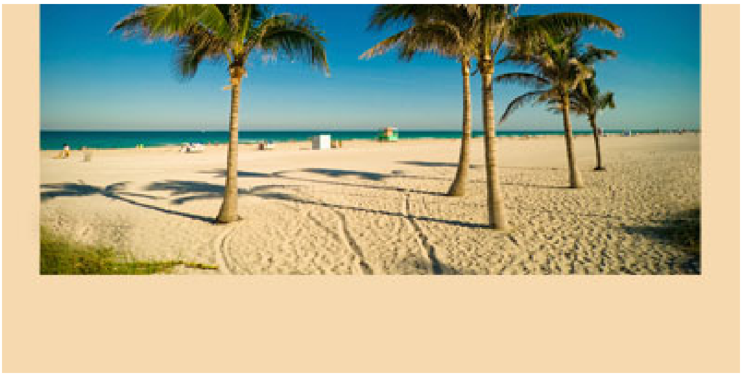
Of course, each background image is different, so instead of agonising over each and every pixel, I'm going to concentrate on five key steps that you can apply to *any* of your own repeating background images. In this example, we'll look at the Miami Beach background image used on the new FOWA site, which I'm afraid is about as un-festive as you can get.

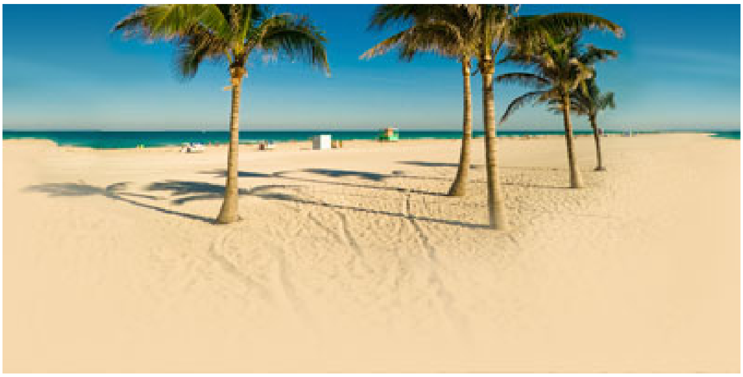
1. CHOOSE YOUR IMAGE WISELY

I find there are three main criteria when judging photos you're considering for repetition manipulation (or 'repetulation', as I like to say)…

- simplicity (beware of complex patterns)
- angle and perspective (watch out for shadows and obvious vanishing points)
- consistent elements (for easy cloning)

You might want to check out this annotated version of the image, where I've highlighted elements of the photo that led me to choose it as the right one.

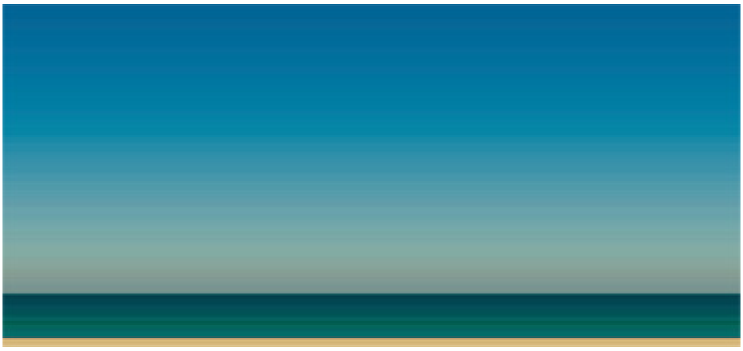

The original image purchased from iStockPhoto.


The Photoshopped version used on the FOWA site.

2. THE POWER OF HORIZONTAL LINES

With the image chosen and your cursor poised for some Photoshop magic, the most useful thing you can do is drag out the edge pixels from one side of the image to create a kind of rough colour 'template' on which to work over. It doesn't matter which side you choose, although you might find it beneficial to use the one with the simplest spread of colour and complex elements.

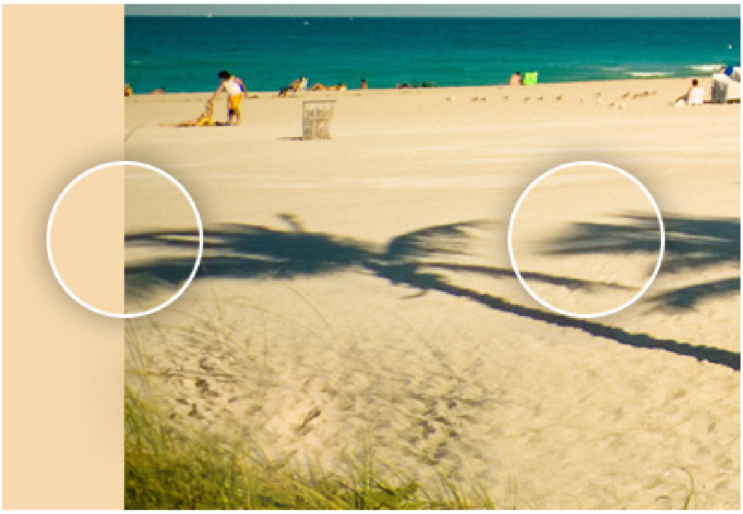
Click and hold on the marquee tool in the toolbar and select the 'single column marquee tool', which will span the full height of your document but will only be one pixel wide. Make the selection right at the edge of your document, press ctrl-c / cmd-c to copy the selection you made, create a new layer, and hit ctrl-v / cmd-v to paste the selection onto your new layer. using free transform (ctrl-t / cmd-t), drag out your selection so that it becomes as wide as your entire canvas.

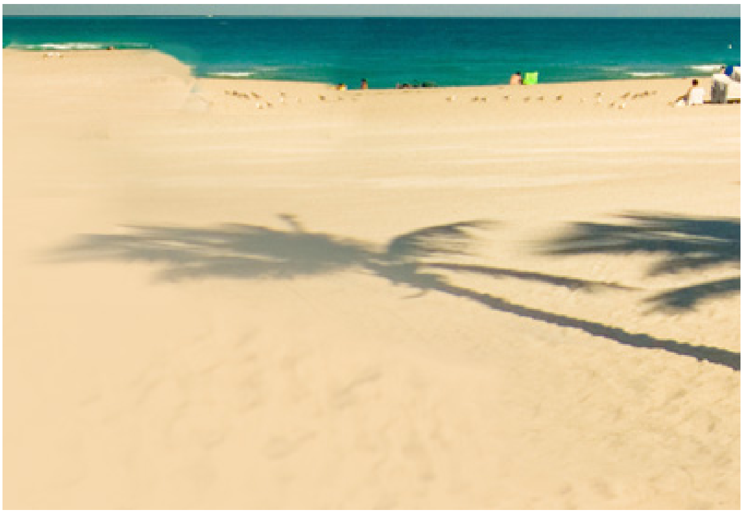

A one-pixel-wide selection stretched out to the entire width of the canvas.

3. CLONING

It goes without saying that the trusty clone tool is one of the most important in the process of creating a seamlessly repeating background image, but I think it's important to be fairly loose with it. Always clone on to a new layer so that you've got the freedom to move it around, but above all else, use the eraser tool to tweak your cloned areas: let that handle the precision stuff and you won't have to worry about getting your clones right first time.

In the example below, you can see how I overcame the problem of the far-left tree shadow being chopped off by cloning the shadow from the tree on its right.

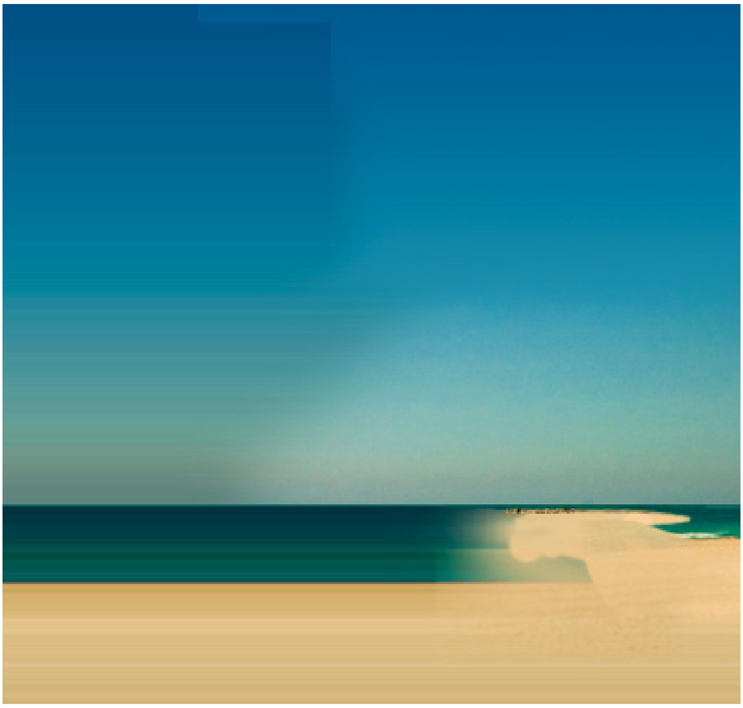

The edge of the shadow is cut off and needs to be 'made' from a pre-existing element.

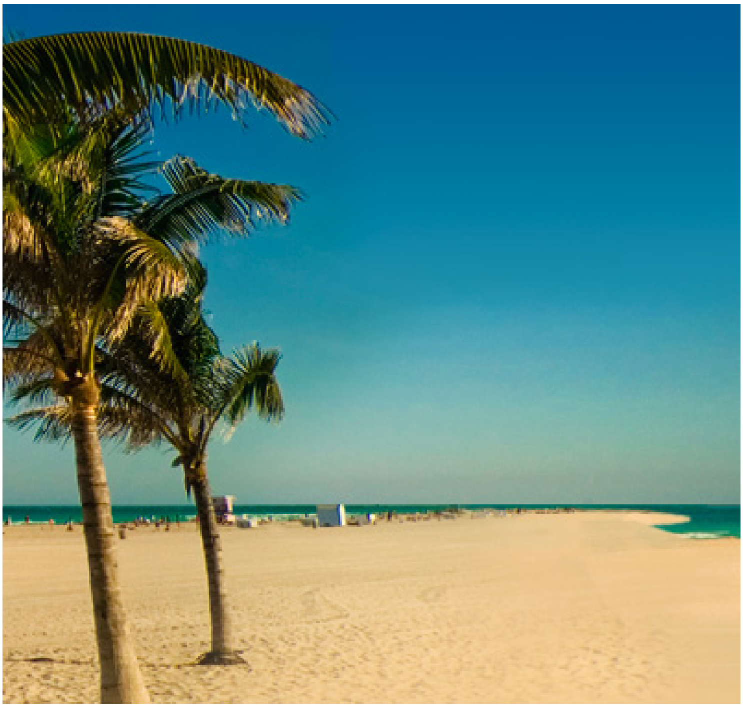

The successful clone completes the missing shadow.

The two elements are obviously very similar but it doesn’t look like a clone because the majority of the shape is ‘genuine’ and only a small part is a duplicate. Also, after cloning I transformed the duplicate, erased parts of it, used gradients, and — ooh, did someone mention gradients?

4. NEVER UNDERESTIMATE A GRADIENT

For this image, I used gradients in a similar way to a brush: covering large parts of the canvas with a colour that faded out to a desired point, before erasing certain parts for accuracy.


Several of the gradients and brushes that make up the ‘customised’ part of the image, visible when the main photograph layer is hidden.


The full composite.

Gradients are also a bit of an easy fix: you can use a gradient on one side of the image, flip it horizontally, and then use it again on the opposite side to make a more seamless join.

Speaking of which…

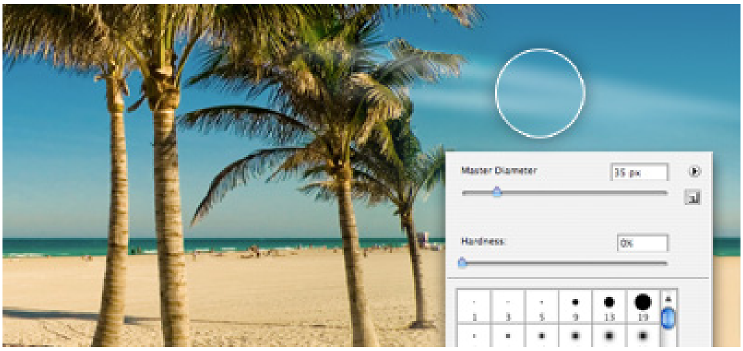
5. SEWING THE SEAMS

No matter what kind of magic Photoshop dust you sprinkle over your image, there will still always be the area where the two edges meet: that scary 'loop' point. Fret ye not, however, for there's help at hand in the form of a nice little cheat. Even though the loop point might still be apparent, we can help hide it by doing something to throw viewers off the scent.

The seam is usually easy to spot because it's a blank area with not much detail or colour variation, so in order to disguise it, go against the rule: put something across it!

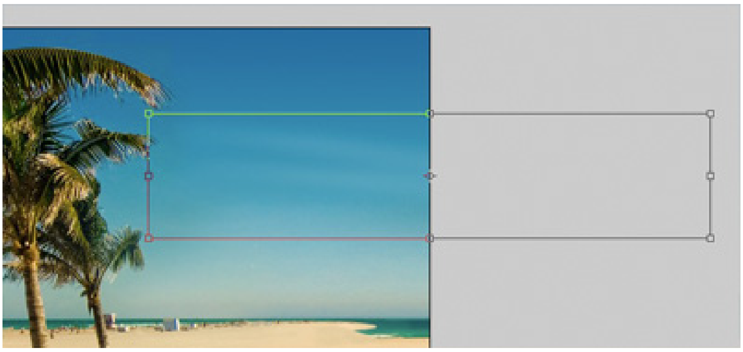
This isn't quite as challenging as it may sound, because if we intentionally make our own 'object' to span the join, we can accurately measure the exact halfway point where we need to split it across the two sides of the image. This is exactly what I did with the FOWA background image: I made some clouds!

A sky with no clouds in an unhappy one.

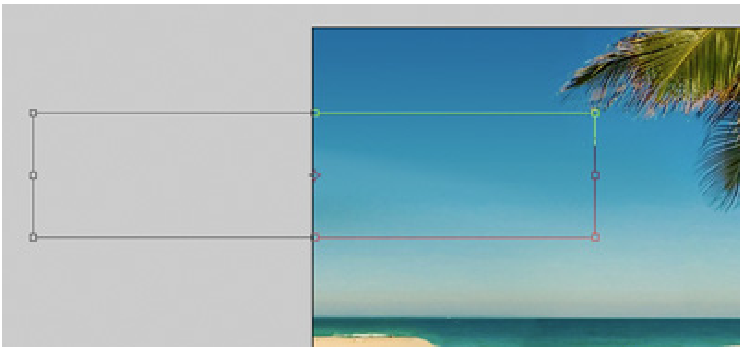

Master Diameter 35 px

Hardness: 0%

A simple soft white brush creates a cloud-like formation in the sky.



After taking the cloud's opacity down to 20%, I used free transform to highlight the boundaries of the layer. I then moved it over to the right, so that the middle of the layer perfectly aligned with the right side of the canvas.



Finally, I duplicated the layer and did the same in reverse: dragging the layer over to the left and making sure that the middle of the duplicate layer perfectly aligned with the *left* side of the canvas.

And there you have it! Boom! Ta-da! Et Voila! To see the repeating background image in action, visit futureofwebapps.com on a large widescreen monitor or see a simulation of the effect.

Thanks for reading, folks. Have a great Christmas!

ABOUT THE AUTHOR

Elliot Jay Stocks is a designer, speaker, and author. He is also the founder of typography magazine 8 Faces and, more recently, the co-founder of Viewport Industries. He lives and works in the countryside between Bristol and Bath, England.

Photo: Samantha Cliffe

4. Capturing Caps Lock

Stuart Langridge 24ways.org/200704

One of the more annoying aspects of having to remember passwords (along with having to remember loads of them) is that if you've got Caps Lock turned on accidentally when you type one in, it won't work, and you won't know why. Most desktop computers alert you in some way if you're trying to enter your password to log on and you've enabled Caps Lock; there's no reason why the web can't do the same. What we want is a warning – maybe the user wants Caps Lock on, because maybe their password is in capitals – rather than something that interrupts what they're doing. Something subtle.

But that doesn't answer the question of how to do it. Sadly, there's no way of actually detecting whether Caps Lock is on directly. However, there's a simple work-around; if the user presses a key, and it’s a capital letter, and they don’t have the Shift key depressed, why then they must have Caps Lock on! Simple.

DOM scripting allows your code to be notified when a key is pressed in an element; when the key is pressed, you get the ASCII code for that key. Capital letters, A to Z, have ASCII codes 65 to 90. So, the code would look something like:

```
on a key press
  if the ASCII code for the key is between 65 and 90
*and* if shift is pressed
    warn the user that they have Caps Lock on, but let
them carry on
  end if
end keypress
```

The actual JavaScript for this is more complicated, because both event handling and keypress information differ across browsers. Your event handling functions are passed an event object, except in Internet Explorer where you use the global event object; the event object has a which parameter containing the ASCII code for the key pressed, except in Internet Explorer where the event object has a keyCode parameter; some browsers store whether the shift key is pressed in a shiftKey parameter and some in a modifiers parameter. All this boils down to code that looks something like this:

```
keypress: function(e) {
  var ev = e ? e : window.event;
  if (!ev) {
    return;
  }
  var targ = ev.target ? ev.target : ev.srcElement;
  // get key pressed
  var which = -1;
  if (ev.which) {
    which = ev.which;
  } else if (ev.keyCode) {
    which = ev.keyCode;
  }
  // get shift status
  var shift_status = false;
  if (ev.shiftKey) {
    shift_status = ev.shiftKey;
  } else if (ev.modifiers) {
    shift_status = !!(ev.modifiers & 4);
  }
```

// At this point, you have the ASCII code in “which”, // and shift_status is true if the shift key is pressed
}

Then it’s just a check to see if the ASCII code is between 65 and 90 *and* the shift key is pressed. (You also need to do the same work if the ASCII code is between 97 (a) and 122 (z) and the shift key is *not* pressed, because shifted letters are lower-case if Caps Lock is on.)

```
if (((which >= 65 && which <= 90) && !shift_status) ||
  ((which >= 97 && which <= 122) && shift_status)) {
  // uppercase, no shift key
  /* SHOW THE WARNING HERE */
} else {
  /* HIDE THE WARNING HERE */
}
```

The warning can be implemented in many different ways: highlight the password field that the user is typing into, show a tooltip, display text next to the field. For simplicity, this code shows the warning as a previously created image, with appropriate `alt` text. Showing the warning means creating a new `<img>` tag with DOM scripting, dropping it into the page, and positioning it so that it's next to the appropriate field. The image looks like this:

Warning!
Caps Lock is on

You know the position of the field the user is typing into (from its `offsetTop` and `offsetLeft` properties) and how wide it is (from its `offsetWidth` properties), so use `createElement` to make the new `img` element, and then absolutely position it with style properties so that it appears in the appropriate place (near to the text field).

The image is a transparent PNG with an alpha channel, so that the drop shadow appears nicely over whatever else is on the page. Because Internet Explorer version 6 and below doesn't handle transparent PNGs correctly, you need to use the AlphaImageLoader technique to make the image appear correctly.

```
newimage = document.createElement('img');
newimage.src = "http://farm3.static.flickr.com/2145/
2067574980_3ddd405905_o_d.png";
newimage.style.position = "absolute";
newimage.style.top = (targ.offsetTop - 73) + "px";
newimage.style.left = (targ.offsetLeft +
targ.offsetWidth - 5) + "px";
newimage.style.zIndex = "999";
newimage.setAttribute("alt", "Warning: Caps Lock is on");
if (newimage.runtimeStyle) {
  // PNG transparency for IE
  newimage.runtimeStyle.filter +=
"progid:DXImageTransform.Microsoft.AlphaImageLoader(src='http://farm3.
2145/
2067574980_3ddd405905_o_d.png',sizingMethod='scale')";
}
document.body.appendChild(newimage);
```

Note that the `alt` text on the image is also correctly set. Next, all these parts need to be pulled together. On page load, identify all the password fields on the page, and attach a keypress handler to each. (This only needs to be done for password fields because the user can see if Caps Lock is on in ordinary text fields.)

```
var inps = document.getElementsByTagName("input");
for (var i=0, l=inps.length; i
```

The “create an image” code from above should only be run if the image is not already showing, so instead of creating a `newimage` object, create the image and attach it to the password field so that it can be checked for later (and not shown if it’s already showing). For safety, all the code should be wrapped up in its own object, so that its functions don’t collide with anyone else’s functions. So, create a single object called `capslock` and make all the functions be named methods of the object:

```
var capslock = {
  ...
  keypress: function(e) {
  }
  ...
}
```

Also, the “create an image” code is saved into its own named function, `show_warning()`, and the converse “remove the image” code into `hide_warning()`. This has the advantage that developers can include the JavaScript library that has been written here, but override what actually happens with their own code, using something like:

```
<script src="jscapslock.js" type="text/
javascript"></script>
<script type="text/javascript">
```

```
  capslock.show_warning(target) {
    // do something different here to warn the user
  }
  capslock.hide_warning(target) {
    // hide the warning that we created in
show_warning() above
  }
</script>
```

And that’s all. Simply include the JavaScript library in your pages, override what happens on a warning if that’s more appropriate for what you’re doing, and that’s all you need.

See the script in action.

ABOUT THE AUTHOR

Stuart Langridge is a web hacker, author, and speaker living in the UK. When not writing books about JavaScript or trying to convince more people to use Ubuntu, he's a founder member of the WaSP's DOM Scripting Task Force and one quarter of the team at LugRadio, the world's best open source radio show. Code and writings and (the occasional rant) are to be found at kryogenix.org; Stuart is to be found outside in the rain looking for the smoking area.

Photo: Lodewijk Schutte

5. My Other Christmas Present Is a Definition List

Mark Norman Francis

24ways.org/200705

A note from the editors: readers should note that the HTML5 redefinition of definition lists has come to pass and is now à la mode.

Last year, I looked at how the markup for tag clouds was generally terrible. I thought this year I would look not at a method of marking up a common module, but instead just at a simple part of HTML and how it generally gets abused.

No, not tables. Definition lists. Ah, definition lists. Often used but rarely understood.

Examining the definition of definitions

To start with, let's see what the HTML spec has to say about them.

> Definition lists vary only slightly from other types of lists in that list items consist of two parts: a term and a description.

The canonical example of a definition list is a dictionary. Words can have multiple descriptions (even the word *definition* has at least five). Also, many terms can share a single definition (for example, the word colour can also be spelt color, but they have the same definition).

Excellent, we can all grasp that. But it very quickly starts to fall apart. Even in the HTML specification the definition list is mis-used.

> Another application of `DL`, for example, is for marking up dialogues, with each `DT` naming a speaker, and each `DD` containing his or her words.

Wrong. Completely and utterly wrong. This is the biggest flaw in the HTML spec, along with dropping support for the `start` attribute on ordered lists. "Why?", you may ask. Let me give you an example from Romeo and Juliet, act 2, scene 2.

```
<dt>Juliet</dt>
  <dd>Romeo!</dd>
<dt>Romeo</dt>
  <dd>My niesse?</dd>
<dt>Juliet</dt>
  <dd>At what o'clock tomorrow shall I send to thee?</dd>
<dt>Romeo</dt>
  <dd>At the hour of nine.</dd>
```

Now, the problem here is that a given definition can have multiple descriptions (the DD). Really the dialog "descriptions" should be rolled up under the terms, like so:

```
<dt>Juliet</dt>
  <dd>Romeo!</dd>
  <dd>At what o'clock tomorrow shall I send to thee?</dd>
<dt>Romeo</dt>
  <dd>My niesse?</dd>
  <dd>At the hour of nine.</dd>
```

Suddenly the play won't make anywhere near as much sense. (If it's anything, the correct markup for a play is an ordered list of CITE and BLOCKQUOTE elements.)

This is the first part of the problem. That simple example has turned definition lists in everyone's mind from pure definitions to more along the lines of a list with pre-configured heading(s) and text(s).

Screen reader, enter stage left.

In many screen readers, a simple definition list would be read out as “definition term **equals** definition description”. So in our play excerpt, Juliet **equals** Romeo! That’s not right, either. But this also leads a lot of people astray with definition lists to believing that they are useful for key/value pairs.

Behaviour and convention

The WHAT-WG have noticed the common mis-use of the DL, and have codified it into the new spec. In the HTML5 draft, a definition list is no longer a definition list.

> The `dl` element introduces an unordered association list consisting of zero or more name–value groups (a description list). Each group must consist of one or more names (`dt` elements) followed by one or more values (`dd` elements).

They also note that the “`dl` element is inappropriate for marking up dialogue, since dialogue is ordered”. So for that example they have created a `DIALOG` (sic) element.

Strange, then, that they keep `DL` as-is but instead refer to it an “association list”. They have not created a new `AL` element, and kept `DL` for the original purpose. They have chosen not to correct the usage or to create a new opportunity for increased specificity in our HTML, but to "pave the cowpath" of convention.

How to use a definition list

Given that everyone else is using a `DL` incorrectly, should we? Well, if they all jumped off a bridge, would you too? No, of course you wouldn't. We don't have HTML5 yet, so we're stuck with the existing semantics of HTML4 and XHTML1. Which means that:

- Listing dialogue is not defining anything.
- Listing the attributes of a piece of hardware (resolution = 1600×1200) is illustrating sample values, not defining anything (however, stating what 'resolution' actually means in this context would be a definition).
- Listing the cast and crew of a given movie is not defining the people involved in making movies. (Stuart Gordon may have been the director of Space Truckers, but that by no means makes him the true definition of a director.)
- A menu of navigation items is simply a nested ordered or unordered list of links, not a definition list.
- Applying styling handles to form labels and elements is not a good use for a definition list.

And so on.

Living by the specification, a definition list should be used for term definitions – glossaries, lexicons and dictionaries – only.

Anything else is a crime against markup.

ABOUT THE AUTHOR

Mark Norman Francis is obsessed with HTML, semantics, code quality and doing things right. He is based in London, England and hopes one day to start blogging properly at marknormanfrancis.com.

6. Minification: A Christmas Diet

Gareth Rushgrove 24ways.org/200706

The festive season is generally more about gorging ourselves than staying thin but we're going to change all that with a quick introduction to minification.

Performance has been a hot topic this last year. We're building more complex sites and applications but at the same time trying to make then load faster and behave more responsively. What is a discerning web developer to do?

Minification is the process of make something smaller, in the case of web site performance we're talking about reducing the size of files we send to the browser. The primary front-end components of any website are HTML, CSS, Javascript and a sprinkling of images. Let's find some tools to trim the fat and speed up our sites.

For those that want to play along at home you can download the various utilities for Mac or Windows. You'll want to be familiar with running apps on the command line too.

HTMLTIDY

HTMLTidy optimises and strips white space from HTML documents. It also has a pretty good go at correcting any invalid markup while it's at it.

```
tidy -m page.html
```

CSSTIDY

CSSTidy takes your CSS file, optimises individual rules (for instance transforming `padding-top: 10px; padding-bottom: 10px;` to `padding: 10px 0;`) and strips unneeded white space.

```
csstidy style.css style-min.css
```

JSMIN

JSMin takes your javascript and makes it more compact. With more and more websites using javascript to power (progressive) enhancements this can be a real bandwidth hog. Look out for pre-minified versions of libraries and frameworks too.

```
jsmin <script.js >script-min.js
```

Remember to run JSLint before you run JSMin to catch some common problems.

OPTIPNG

Images can be a real bandwidth hog and making all of them smaller with OptiPNG should speed up your site.

```
optipng image.png
```

All of these tools have an often bewildering array of options and generally good documentation included as part of the package. A little experimentation will get you even more bang for your buck.

For larger projects you likely won't want to be manually minifying all your files. The best approach here is to integrate these tools into your build process and have your live website come out the other side smaller than it went in.

You can also do things on the server to speed things up; GZIP compression for instance or compilation of resources to reduce the number of HTTP requests. If you're interested in performance a good starting point is the Exceptional Performance section on the Yahoo Developer Network and remember to install the YSlow Firebug extension while you're at it.

ABOUT THE AUTHOR

Gareth Rushgrove is web developer based in Cambridge and working in London for Global Radio. At work he spends most of his time writing Python and Django based applications or tinkering with testing tools.

In the past Gareth worked on everything from successful marketing campaigns to enterprise content management and financial service applications. These days he's generally found shouting about the benefits of APIs, XMPP and embracing the web as a platform.

When not working, Gareth can be found blogging over on morethanseven.net or uploading code to github.com/garethr. He's previously kept himself busy organising a BarCamp in

Newcastle upon Tyne and starting the Refresh Newcastle user group. He also helped out on the board of the Thinking Digital conference last year.

Photo: David Thompson

7. Typesetting Tables

Mark Boulton 24ways.org/200707

Tables have suffered in recent years on the web. They were used for laying out web pages. Then, following the Web Standards movement, they've been renamed by the populous as `data tables' to ensure that we all know what they're for. There have been some great tutorials for the designing tables using CSS for presentation and focussing on the semantics in the displaying of data in the correct way. However, typesetting tables is a subtle craft that has hardly had a mention.

Table design can often end up being a technical exercise. What data do we need to display? Where is the data coming from and what form will it take? When was the last time your heard someone talk about lining numerals? Or designing to the reading direction?

TABLES ARE NOT READ LIKE SENTENCES

When a reader looks at, and tries to understand, tabular data, they're doing a bunch of things at the same time.

1. Generally, they're task based; they're looking for something.
2. They are reading horizontally AND vertically

Reading a table is not like reading a paragraph in a novel, and therefore shouldn't be typeset in the same way. Designing tables is information design, it's functional typography—it's not a time for eye candy.

TYPESETTING TABLES

Typesetting great looking tables is largely an exercise in restraint. Minimal interference with the legibility of the table should be in the forefront of any designers mind.

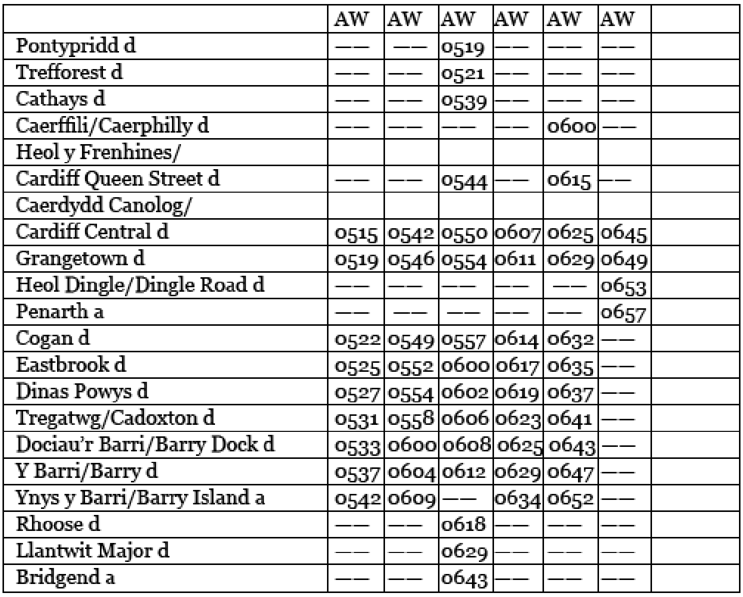
When I'm designing tables I apply some simple rules:

1. Plenty of negative space
2. Use the right typeface
3. Go easy on the background tones, unless you're giving reading direction visual emphasis
4. Design to the reading direction

By way of explanation, here are those rules as applied to the following badly typeset table.

YOUR DEFAULT TABLE

This table is a mess. There is no consideration for the person trying to read it. Everything is too tight. The typeface is wrong. It's flat. A grim table indeed.



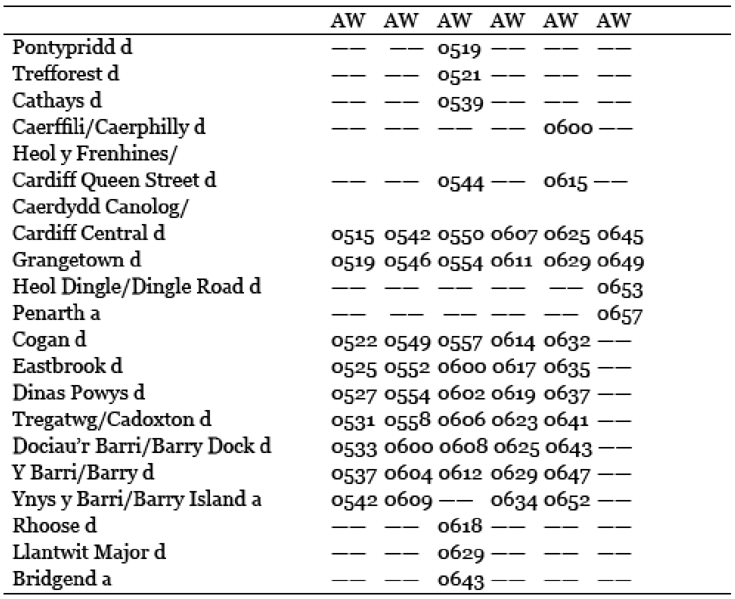
	AW	AW	AW	AW	AW	AW	
Pontypridd d	——	——	0519	——	——	——	
Trefforest d	——	——	0521	——	——	——	
Cathays d	——	——	0539	——	——	——	
Caerffili/Caerphilly d	——	——	——	——	0600	——	
Heol y Frenhines/							
Cardiff Queen Street d	——	——	0544	——	0615	——	
Caerdydd Canolog/							
Cardiff Central d	0515	0542	0550	0607	0625	0645	
Grangetown d	0519	0546	0554	0611	0629	0649	
Heol Dingle/Dingle Road d	——	——	——	——	——	0653	
Penarth a	——	——	——	——	——	0657	
Cogan d	0522	0549	0557	0614	0632	——	
Eastbrook d	0525	0552	0600	0617	0635	——	
Dinas Powys d	0527	0554	0602	0619	0637	——	
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——	
Dociau’r Barri/Barry Dock d	0533	0600	0608	0625	0643	——	
Y Barri/Barry d	0537	0604	0612	0629	0647	——	
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——	
Rhoose d	——	——	0618	——	——	——	
Llantwit Major d	——	——	0629	——	——	——	
Bridgend a	——	——	0643	——	——	——	

Let’s see what we can do about that.

PLENTY OF NEGATIVE SPACE

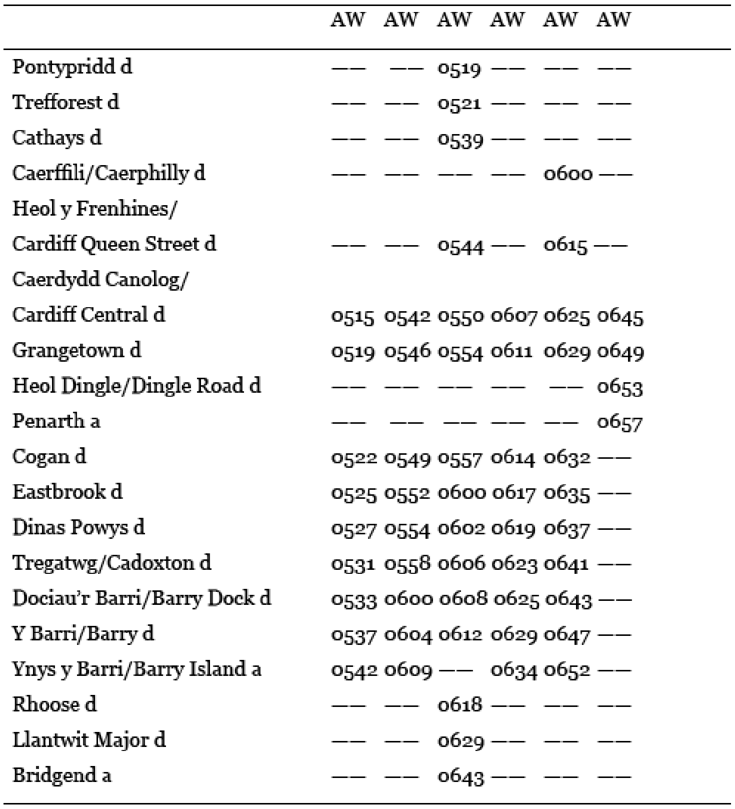
The badly typeset table has been set with default padding. There has been little consideration for the ascenders and descenders in the type interfering with the many horizontal rules.

The first thing we do is remove most of the lines, or rules. You don’t need them – the data in the rows forms its own visual rules. Now, with most of the rules removed, the ones that remain mean something; they are indicating some kind of hierarchy to the help the reader understand what the different table elements mean – in this case the column headings.



	AW	AW	AW	AW	AW	AW
Pontypridd d	——	——	0519	——	——	——
Trefforest d	——	——	0521	——	——	——
Cathays d	——	——	0539	——	——	——
Caerffili/Caerphilly d	——	——	——	——	0600	——
Heol y Frenhines/ Cardiff Queen Street d	——	——	0544	——	0615	——
Caerdydd Canolog/ Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	——	——	——	——	——	0653
Penarth a	——	——	——	——	——	0657
Cogan d	0522	0549	0557	0614	0632	——
Eastbrook d	0525	0552	0600	0617	0635	——
Dinas Powys d	0527	0554	0602	0619	0637	——
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	——
Y Barri/Barry d	0537	0604	0612	0629	0647	——
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——
Rhoose d	——	——	0618	——	——	——
Llantwit Major d	——	——	0629	——	——	——
Bridgend a	——	——	0643	——	——	——

Now we need to give the columns and rows more negative space. Note the framing of the column headings. I'm giving them more room at the bottom. This negative space is active—it's empty for a reason. The extra air in here also gives more hierarchy to the column headings.

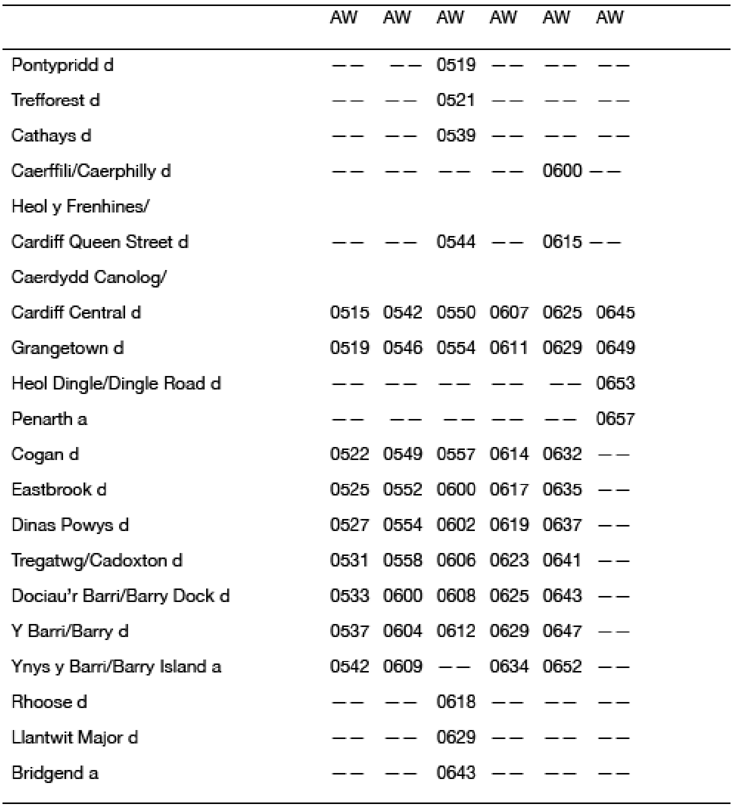


	AW	AW	AW	AW	AW	AW
Pontypridd d	——	——	0519	——	——	——
Trefforest d	——	——	0521	——	——	——
Cathays d	——	——	0539	——	——	——
Caerffili/Caerphilly d	——	——	——	——	0600	——
Heol y Frenhines/						
Cardiff Queen Street d	——	——	0544	——	0615	——
Caerdydd Canolog/						
Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	——	——	——	——	——	0653
Penarth a	——	——	——	——	——	0657
Cogan d	0522	0549	0557	0614	0632	——
Eastbrook d	0525	0552	0600	0617	0635	——
Dinas Powys d	0527	0554	0602	0619	0637	——
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	——
Y Barri/Barry d	0537	0604	0612	0629	0647	——
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——
Rhoose d	——	——	0618	——	——	——
Llantwit Major d	——	——	0629	——	——	——
Bridgend a	——	——	0643	——	——	——

USE THE RIGHT TYPEFACE

The default table is set in a serif typeface. This isn't ideal for a couple of reasons. This serif typeface has a standard set of text numerals. These dip below the baseline and are designed for using figures within text, not on their own.

What you need to use is a typeface with lining numerals. These align to the baseline and are more legible when used for tables.



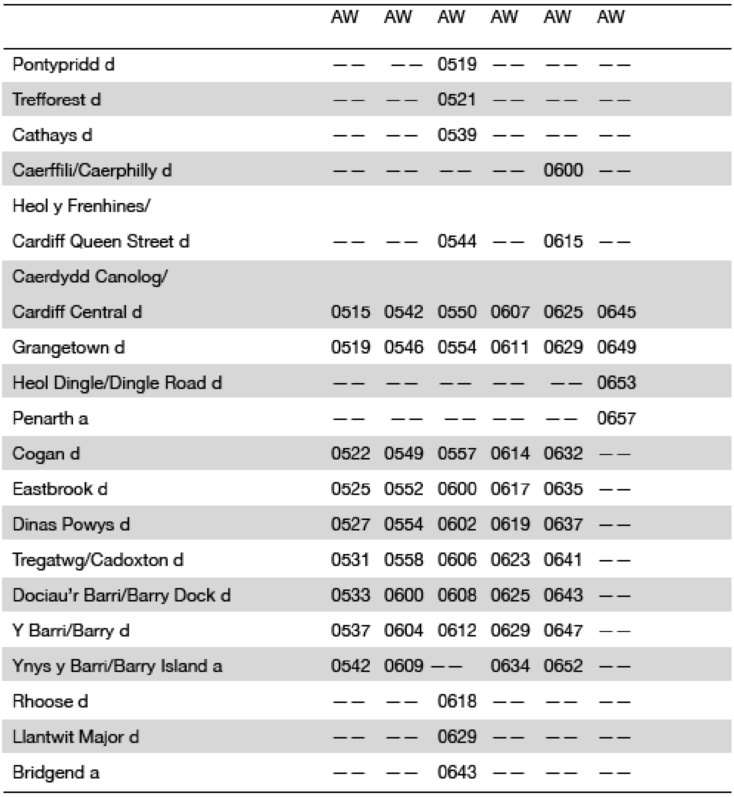
	AW	AW	AW	AW	AW	AW
Pontypridd d	— —	— —	0519	— —	— —	— —
Trefforest d	— —	— —	0521	— —	— —	— —
Cathays d	— —	— —	0539	— —	— —	— —
Caerffili/Caerphilly d	— —	— —	— —	— —	0600	— —
Heol y Frenhines/						
Cardiff Queen Street d	— —	— —	0544	— —	0615	— —
Caerdydd Canolog/						
Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	— —	— —	— —	— —	— —	0653
Penarth a	— —	— —	— —	— —	— —	0657
Cogan d	0522	0549	0557	0614	0632	— —
Eastbrook d	0525	0552	0600	0617	0635	— —
Dinas Powys d	0527	0554	0602	0619	0637	— —
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	— —
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	— —
Y Barri/Barry d	0537	0604	0612	0629	0647	— —
Ynys y Barri/Barry Island a	0542	0609	— —	0634	0652	— —
Rhoose d	— —	— —	0618	— —	— —	— —
Llantwit Major d	— —	— —	0629	— —	— —	— —
Bridgend a	— —	— —	0643	— —	— —	— —

Sans serif typefaces generally have lining numerals. They are also arguably more legible when used in tables.

GO EASY ON THE BACKGROUND TONES, UNLESS YOU'RE GIVING READING DIRECTION VISUAL EMPHASIS

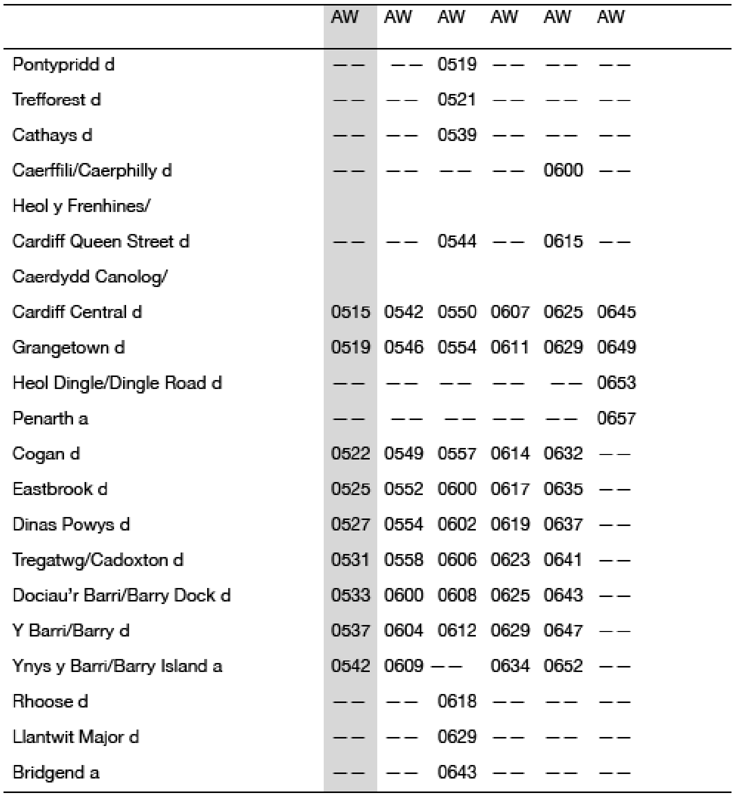
We've all seen background tones on tables. They have their use, but my feeling is that use should be functional and not decorative.

If you have a table that is long, but only a few columns wide, then alternate row shading isn't that useful for showing the different lines of data. It's a common misconception that alternate row shading is to increase legibility on long tables. That's not the case. Shaded rows are to aid horizontal reading across multiple table columns. On wide tables they are incredibly useful for helping the reader find what they want.



	AW	AW	AW	AW	AW	AW
Pontypridd d	——	——	0519	——	——	——
Trefforest d	——	——	0521	——	——	——
Cathays d	——	——	0539	——	——	——
Caerffili/Caerphilly d	——	——	——	——	0600	——
Heol y Frenhines/ Cardiff Queen Street d	——	——	0544	——	0615	——
Caerdydd Canolog/ Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	——	——	——	——	——	0653
Penarth a	——	——	——	——	——	0657
Cogan d	0522	0549	0557	0614	0632	——
Eastbrook d	0525	0552	0600	0617	0635	——
Dinas Powys d	0527	0554	0602	0619	0637	——
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	——
Y Barri/Barry d	0537	0604	0612	0629	0647	——
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——
Rhoose d	——	——	0618	——	——	——
Llantwit Major d	——	——	0629	——	——	——
Bridgend a	——	——	0643	——	——	——

Background tone can also be used to give emphasis to the reading direction. If we want to emphasis a column, that can be given a background tone.

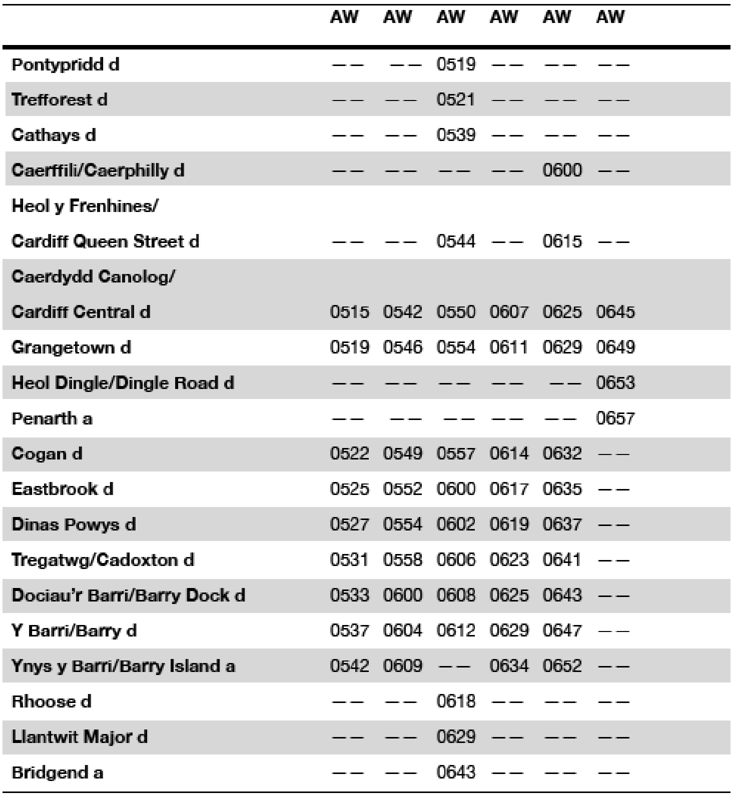


	AW	AW	AW	AW	AW	AW
Pontypridd d	——	——	0519	——	——	——
Trefforest d	——	——	0521	——	——	——
Cathays d	——	——	0539	——	——	——
Caerffili/Caerphilly d	——	——	——	——	0600	——
Heol y Frenhines/						
Cardiff Queen Street d	——	——	0544	——	0615	——
Caerdydd Canolog/						
Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	——	——	——	——	——	0653
Penarth a	——	——	——	——	——	0657
Cogan d	0522	0549	0557	0614	0632	——
Eastbrook d	0525	0552	0600	0617	0635	——
Dinas Powys d	0527	0554	0602	0619	0637	——
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	——
Y Barri/Barry d	0537	0604	0612	0629	0647	——
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——
Rhoose d	——	——	0618	——	——	——
Llantwit Major d	——	——	0629	——	——	——
Bridgend a	——	——	0643	——	——	——

HIERARCHY

As I said earlier, people may be reading a table vertically, and horizontally in order to find what they want. Sometimes, especially if the table is complex, we need to give them a helping hand.

Visually emphasising the hierarchy in tables can help the reader scan the data. Column headings are particularly important. Column headings are often what a reader will go to first, so we need to help them understand that the column headings are different to the stuff beneath them, and we also need to give them more visual importance. We can do this by making them bold, giving them ample negative space, or by including a thick rule above them. We can also give the row titles the same level of emphasis.



	AW	AW	AW	AW	AW	AW
Pontypridd d	——	——	0519	——	——	——
Trefforest d	——	——	0521	——	——	——
Cathays d	——	——	0539	——	——	——
Caerffili/Caerphilly d	——	——	——	——	0600	——
Heol y Frenhines/ Cardiff Queen Street d	——	——	0544	——	0615	——
Caerdydd Canolog/ Cardiff Central d	0515	0542	0550	0607	0625	0645
Grangetown d	0519	0546	0554	0611	0629	0649
Heol Dingle/Dingle Road d	——	——	——	——	——	0653
Penarth a	——	——	——	——	——	0657
Cogan d	0522	0549	0557	0614	0632	——
Eastbrook d	0525	0552	0600	0617	0635	——
Dinas Powys d	0527	0554	0602	0619	0637	——
Tregatwg/Cadoxton d	0531	0558	0606	0623	0641	——
Dociau'r Barri/Barry Dock d	0533	0600	0608	0625	0643	——
Y Barri/Barry d	0537	0604	0612	0629	0647	——
Ynys y Barri/Barry Island a	0542	0609	——	0634	0652	——
Rhoose d	——	——	0618	——	——	——
Llantwit Major d	——	——	0629	——	——	——
Bridgend a	——	——	0643	——	——	——

In addition to background tones, you can give emphasis to reading direction by typesetting those elements in bold. You shouldn't use italics—with sans serif typefaces the difference is too subtle.

So, there you have it. A couple of simple guidelines to make your tables cleaner and more readable.

ABOUT THE AUTHOR

Mark Boulton is a graphic designer from near Cardiff in the UK. He used to work as a Senior Designer for the BBC, before he took leave of his senses and formed his own design consultancy, Mark Boulton Design. He studied typography, enjoys watching a good boxing match, and is partial to a really good cuppa.

8. JavaScript Internationalisation

Matthew Somerville 24ways.org/200708

OR: WHY RUDOLPH IS MORE THAN JUST A SHINY NOSE

Dunder sat, glumly staring at the computer screen.

"What's up, Dunder?" asked Rudolph, entering the stable and shaking off the snow from his antlers.

"Well," Dunder replied, "I've just finished coding the new reindeer intranet Santa Claus asked me to do. You know how he likes to appear to be at the cutting edge, talking incessantly about Web 2.0, AJAX, rounded corners; he even spooked Comet recently by talking about him as if he were some pushy web server.

"I've managed to keep him happy, whilst also keeping it usable, accessible, and gleaming — and I'm *still* on the back row of the sleigh! But anyway, given the elves will be the ones using the site, and they come from all over the world, the site is in multiple languages. Which is great, except when it comes to the preview JavaScript I've written for the reindeer order form. Here, have a look…"

As he said that, he brought up the textileRef:8234272265470b85d91702:linkStartMarker:“order form in French”:/examples/javascript-internationalisation/initial.fr.html on the screen. (Same in English).

“Looks good,” said Rudolph.

“But if I add some items,” said Dunder, “the preview appears in English, as it’s hard-coded in the JavaScript. I don’t want separate code for each language, as that’s just silly — I thought about just having if statements, but that doesn’t scale at all…”

“And there’s more, you aren’t displaying large numbers in French properly, either,” added Rudolph, who had been playing and looking at part of the source code:

```
function update_text() {
  var hay = getValue('hay');
  var carrots = getValue('carrots');
  var bells = getValue('bells');
  var total = 50 * bells + 30 * hay + 10 * carrots;
  var out = 'You are ordering '
    + pretty_num(hay) + ' bushel' + pluralise(hay) + '
of hay, '
    + pretty_num(carrots) + ' carrot' +
pluralise(carrots)
    + ', and ' + pretty_num(bells) + ' shiny bell' +
pluralise(bells)
    + ', at a total cost of <strong>' + pretty_num(total)
    + '</strong> gold pieces. Thank you.';
  document.getElementById('preview').innerHTML = out;
}
function pretty_num(n) {
  n += '';
  var o = '';
  for (i=n.length; i>3; i-=3) {
    o = ',' + n.slice(i-3, i) + o;
  }
  o = n.slice(0, i) + o;
  return o;
}
function pluralise(n) {
  if (n!=1) return 's';
  return '';
}
```

“Oh, botheration!” cried Dunder. “This is just so complicated.”

“It doesn’t have to be,” said Rudolph, “you just have to think about things in a slightly different way from what you’re used to. As we’re only a simple example, we won’t be able to cover all possibilities, but for starters, we need some way of providing different information to the script dependent on the language. We’ll create a global i18n object, say, and fill it with the correct language information. The first variable we’ll need will be a thousands separator, and then we can change the `pretty_num` function to use that instead:

```
function pretty_num(n) {
  n += '';
  var o = '';
  for (i=n.length; i>3; i-=3) {
    o = i18n.thousands_sep + n.slice(i-3, i) + o;
  }
  o = n.slice(0, i) + o;
  return o;
}
```

“The i18n object will also contain our translations, which we will access through a function called `_()` — that’s just an underscore. Other languages have a function of the same name doing the same thing. It’s very simple:

```
function _(s) {
  if (typeof(i18n)!='undefined' && i18n[s]) {
    return i18n[s];
  }
  return s;
}
```

“So if a translation is available and provided, we’ll use that; otherwise we’ll default to the string provided — which is helpful if the translation begins to lag behind the site’s text at all, as at least something will be output.”

“Got it,” said Dunder. “ `_('Hello Dunder')` will print the translation of that string, if one exists, ‘Hello Dunder’ if not.”

"Exactly. Moving on, your plural function breaks even in English if we have a word where the plural doesn't add an s — like 'children'."

"You're right," said Dunder. "How did I miss that?"

"No harm done. Better to provide both singular and plural words to the function and let it decide which to use, performing any translation as well:

```
function pluralise(s, p, n) {
  if (n != 1) return _(p);
  return _(s);
}
```

"We'd have to provide different functions for different languages as we employed more elves and got more complicated — for example, in Polish, the word 'file' pluralises like this: 1 plik, 2-4 pliki, 5-21 plików, 22-24 pliki, 25-31 plików, and so on." (More information on plural forms)

"Gosh!"

"Next, as different languages have different word orders, we must stop using concatenation to construct sentences, as it would be impossible for other languages to fit in; we have to keep coherent strings together. Let's rewrite your update function, and then go through it:

```
function update_text() {
  var hay = getValue('hay');
  var carrots = getValue('carrots');
  var bells = getValue('bells');
  var total = 50 * bells + 30 * hay + 10 * carrots;
  hay = sprintf(pluralise('%s bushel of hay', '%s
bushels of hay', hay), pretty_num(hay));
  carrots = sprintf(pluralise('%s carrot', '%s carrots',
carrots), pretty_num(carrots));
  bells = sprintf(pluralise('%s shiny bell', '%s shiny
bells', bells), pretty_num(bells));
  var list = sprintf(_('%s, %s, and %s'), hay, carrots,
bells);
  var out = sprintf(_('You are ordering %s, at a total
cost of <strong>%s</strong> gold pieces.'),
    list, pretty_num(total));
  out += ' ';
  out += _('Thank you.');
  document.getElementById('preview').innerHTML = out;
}
```

“ `sprintf` is a function in many other languages that, given a format string and some variables, slots the variables into place within the string. JavaScript doesn’t have such a function, so we’ll write our own. Again, keep it simple for now, only integers and strings; I’m sure more complete ones can be found on the internet.

```
function sprintf(s) {
  var bits = s.split('%');
  var out = bits[0];
  var re = /^([ds])(.*)$/;
  for (var i=1; i<bits.length; i++) {
    p = re.exec(bits[i]);
    if (!p || arguments[i]==null) continue;
    if (p[1] == 'd') {
      out += parseInt(arguments[i], 10);
    } else if (p[1] == 's') {
      out += arguments[i];
    }
    out += p[2];
  }
  return out;
}
```

"Lastly, we need to create one file for each language, containing our i18n object, and then include that from the relevant HTML. Here's what a blank translation file would look like for your order form:

```
var i18n = {
  thousands_sep: ',',
  "%s bushel of hay": '',
  "%s bushels of hay": '',
  "%s carrot": '',
  "%s carrots": '',
  "%s shiny bell": '',
  "%s shiny bells": '',
  "%s, %s, and %s": '',
  "You are ordering %s, at a total cost of
<strong>%s</strong> gold pieces.": '',
  "Thank you.": ''
};
```

"If you implement this across the intranet, you'll want to investigate the xgettext program, which can automatically extract all strings that need translating from all sorts of code files into a standard `.po` file (I think Python mode works best for JavaScript). You can then use a different program to take the translated `.po` file and automatically create the language-specific JavaScript files for us." (e.g. German .po file for PledgeBank, mySociety's .po-.js script, example output)

With a flourish, Rudolph finished editing. "And there we go, localised JavaScript in English, French, or German, all using the same main code."

"Thanks so much, Rudolph!" said Dunder.

"I'm not just a pretty nose!" Rudolph quipped. "Oh, and one last thing — please comment liberally explaining the context of strings you use. Your translator will thank you, probably at the same time as they point out the four hundred places you've done something in code that only works in your language and no-one else's…"

Thanks to Tim Morley and Edmund Grimley Evans for the French and German translations respectively.

ABOUT THE AUTHOR

Matthew Somerville is a former civil servant, who realised that actually getting to design and program stuff was more fun and rewarding. He has helped to create various popular democracy and civic websites, such as TheyWorkForYou and FixMyStreet, wrote a nicer version of the UK train timetable site with bookmarkable URLs, and holidays in Landmark Trust properties as often as he can.

Photo: Tom Coates

9. Back To The Future of Print

Natalie Downe 24ways.org/200709

By now we have weathered the storm that was the early days of web development, a dangerous time when we used tables, inline CSS and separate pages for print only versions. We can reflect in a haggard old sea-dog manner ("yarrr… I remember back in the browser wars…") on the bad practices of the time. We no longer need convincing that print stylesheets are the way to go[1], though some of the documentation for them is a little outdated now.

I am going to briefly cover 8 tips and 4 main gotchas when creating print stylesheets in our more enlightened era.

GETTING STARTED

As with regular stylesheets, print CSS can be included in a number of ways[2], for our purposes we are going to be using the link
element.

```
<link rel="stylesheet" type="text/css"
media="print" href="print.css">
```

This is still my favourite way of linking to CSS files, its easy to see what files are being included and to what media they are being applied to. Without the media attribute specified the link element defaults to the media type 'all' which means that the styles within then apply to print and screen alike. The media type 'screen' only applies to the screen and wont be picked up by print, this is the best way of hiding styles from print.

Make sure you include your print styles after all your other CSS, because you will need to override certain rules and this is a lot easier if you are flowing with the cascade than against it!

Another thing you should be thinking is 'does it need to be printed'. Consider the context[3], if it is not a page that is likely to be printed, such as a landing page or a section index then the print styles should resemble the way the page looks on the screen.

Context is really important for the design of your print stylesheet, all the tips and tricks that follow should be taken in the context of the page. If for example you are designing a print stylesheet for an item in a shopping cart, it is irrelevant for the user to know the exact url of the link that takes them to your checkout.

TIPS AND TRICKS

During these tip's we are going to build up print styles for a textileRef:11112857385470b854b8411:linkStartMarker:"simple example":/examples/back-to-the-future-of-print/demo-1.html

1. Remove the cruft

First things first, navigation, headers and most page furniture are pretty much useless and dead space in print so they will need to be removed, using `display:none;`.

2. Linearise your content

Content doesn't work so well in columns in print, especially if the content columns are long and intend to stretch over multiple columns (as mentioned in the gotcha section below). You might want to consider Lineariseing the content to flow down the page. If you have your source order in correct priority this will make things a lot easier[4].

3. Improve your type

Once you have removed all the useless cruft and jiggled things about a bit, you can concentrate more on the typography of the page.

Typography is a complex topic[5], but simply put serif-ed fonts such as Georgia work better on print and sans serif-ed fonts such as Verdana are more appropriate for screen use. You will probably want to increase font size and line height and change from `px` to `pt` (which is specifically a print measurement).

4. Go wild on links

There are some incredibly fun things you can do with links in print using CSS. There are two schools of thought, one that consider it is best to disguise inline links as body text because they are not click-able on paper. Personally I believe it is useful to know for reference that the document did link to somewhere originally.

When deciding which approach to take, consider the context of your document, do people need to know where they would have gone to? will it help or hinder them to know this information? Also for an alternative to the below, take a look at Aaron Gustafson's article on generating footnotes for print[6].

Using some clever selector trickery and CSS generated content you can have the location of the link generated after the link itself.

HTML:

```
<p>I wish <a href="http://www.google.com/">Google</a>
could find <a href="/photoOfMyKeys.jpg">my keys</a></p>
```

CSS:

```
a:link:after,
a:visited:after,
a:hover:after,
a:active:after {
  content: " <" attr(href) "> ";
}
```

But this is not perfect, in the above example the content of the `href` is just naively plonked after the link text:

```
I wish Google <http://www.google.com/> would find my
keys </photoOfMyKeys.jpg>
```

As looking back over this printout the user is not immediately aware of the location of the link, a better solution is to use even more crazy selectors to deal with relative links. We can also add a style to the generated content so it is distinguishable from the link text itself.

CSS:

```
a:link:after,
a:visited:after,
a:hover:after,
a:active:after {
  content: " <" attr(href) "> ";
  color: grey;
  font-style: italic;
  font-weight: normal;
}
a[href^="/"]:after {
  content: " <http://www.example.com"attr(href)"> ";
}
```

The output is now what we were looking for (you will need to replace example.com with your own root URL):

```
I wish Google <http://www.google.com/> would find my
keys <http://www.example.com/photoOfMyKeys.jpg>
```

Using regular expressions on the attribute selectors, one final thing you can do is to suppress the generated content on `mailto:` links, if for example you know the link text always reflects the email address. Eg:

HTML:

```
<a href="mailto:me@example.com">me@example.com</a>
```

CSS:

```
a[href^="mailto"]:after {
  content: "";
}
```

This example shows the above in action.

Of course with this clever technique, there are the usual browser support issues. While it won't look as intended in browsers such as Internet Explorer 6 and 7 (IE6 and IE7) it will not break either and will just degrade gracefully because IE cannot do generated content. To the best of my knowledge Safari 2+ and Opera 9.X support a colour set on generated content whereas Firefox 2 & Camino display this in black regardless of the link or inherited text colour.

5. Jazz your headers for print

This is more of a design consideration, don't go too nuts though; there are a lot more limitations in print media than on screen. For this example we are going to go for is having a bottom border on `h2`'s and styling other headings with graduating colors or font sizes.

And here is the example complete with jazzy headers.

6. Build in general hooks

If you are building a large site with many different types of page, you may find it useful to build into your CSS structure, classes that control what is printed (e.g. noprint and printonly). This may not be semantically ideal, but in the past I have found it really useful for maintainability of code across large and varied sites

7. For that extra touch of class

When printing pages from a long URL, even if the option is turned on to show the location of the page in the header, browsers may still display a truncated (and thus useless) version.

Using the above tip (or just simply setting to `display:none` in screen and `display:block` in print) you can insert the URL of the page you are currently on for print only, using JavaScript's `window.location.href` variable.

```
function addPrintFooter() {
  var p = document.createElement('p');
  p.className = 'print-footer';
  p.innerHTML = window.location.href;
  document.body.appendChild(p);
}
```

You can then call this function using whichever `onload` or `ondomready` handler suits your fancy. Here is our familiar demo to show all the above in action

8. Tabular data across pages

If you are using tabled data in your document there are a number of things you can do to increase usability of long tables over several pages. If you use the `<thead>` element this should repeat your table headers on the next page should your table be split. You will need to set `thead {display: table-header-group;}` explicitly for IE even though this should be the default value.

Also if you use `tr {page-break-inside: avoid;}` this should (browser support depending) stop your table row from breaking across two pages. For more information on styling tables for print please see the CSS discuss wiki[7].

GOTCHAS

1. Where did all my content go?

Absolutely *the* most common mistake I see with print styles is the **truncated content bug**. The symptom of this is that only the first page of a `div`'s content will be printed, the rest will look truncated after this.

Floating long columns may still have this affect, as mentioned in Eric Meyer's article on 'A List Apart' article from 2002[8]; though in testing I am no longer able to replicate this. Using `overflow:hidden` on long content in Firefox however still causes this truncation. Overflow hidden is commonly used to clear floats[9].

A simple fix can be applied to resolve this, if the column is floated you can override this with `float:none` similarly `overflow:hidden` can be overridden with `overflow:visible` or the offending rules can be banished to a screen only stylesheet.

Using `position:absolute` on long columns also has a very similar effect in truncating the content, but can be overridden in print with `position:static;`

Whilst I only recommend having a print stylesheet for content pages on your site; do at least check other landing pages, section indexes and your homepage. If these are inaccessible in print possibly due to the above gotcha, it might be wise to provide a light dusting of print styles or move the offending overflow / float rules to a screen only stylesheet to fix the issues.

2. Damn those background browser settings

One of the factors of life you now need to accept is that you can’t control the user’s browser settings, no more than you can control whether or not they use IE6. Most browsers by default will not print background colours or images unless explicitly told to by the user.

Naturally this causes a number of problems, for starters you will need to rethink things like branding. At this point it helps if you are doing the print styles early in the project so that you can control the logo not being a background image for example.

Where colour is important to the meaning of the document, for example a status on an invoice, bear in mind that a textural representation will also need to be supplied as the user may be printing in black and white. Borders will print however regardless of setting, so assuming the user is printing in colour you can always use borders to indicate colour.

Check the colour contrast of the text against white, this may need to be altered without backgrounds. You should check how your page looks with backgrounds turned on too, for consistency with the default browser settings you may want to override your background anyway.

One final issue with backgrounds being off is list items. It is relatively common practice to suppress the `list-style-type` and replace with a background image to finely control the bullet positioning. This technique doesn't translate to print, you will need to disable this background bullet and re-instate your trusty friend the `list-style-type`.

3. Using JavaScript in your CSS? … beware IE6

Internet explorer has an issue that when Javascript is used in a stylesheet it applies this to all media types even if only applied to screen. For example, if you happen to be using expressions to set a width for IE, perhaps to mimic `min-width`, a simple `width:100% !important` rule can overcome the effects the expression has on your print styles[10].

4. De-enhance your Progressive enhancements

Quite a classic "doh" moment is when you realise that, of course paper doesn't support Javascript. If you have any dynamic elements on the page, for example a document collapsed per section, you really should have been using Progressive enhancement techniques[11] and building for browsers without Javascript first, adding in the fancy stuff later.

If this is the case it should be trivial to override your wizzy JS styles in your print stylesheet, to display all your content and make it accessible for print, which is by far the best method of achieving this affect.

AND FINALLY…

I refer you back to the nature of the document in hand, consider the context of your site and the page. Use the tips here to help you add that extra bit of flair to your printed media.

Be careful you don't get caught out by the common gotchas, keep the design simple, test cross browser and relish in the medium of print.

FURTHER READING

[1] For more information constantly updated, please see the CSS discuss wiki on print stylesheets

[2] For more information on media types and ways of including CSS please refer to the CSS discuss wiki on Media Stylesheets

[3] Eric Meyer talks to ThinkVitamin about the importance of context when designing your print strategy.

[4] Mark Boulton describes how he applies a newspaper like print stylesheet to an article in the Guardian website. Mark also has some persuasive arguments that print should not be left to last

[5] Richard Rutter Has a fantastic resource on typography which also applies to print.

[6] Aaron Gustafson has a great solution to link problem by creating footnotes at the end of the page.

[7] The CSS discuss wiki has more detailed information on printing tables and detailed browser support

[8] This 'A List Apart' article is dated May 10th 2002 though is still mostly relevant

[9] Float clearing technique using 'overflow:hidden'

[10] Autistic Cuckoo describes the interesting insight with regards to expressions specified for screen in IE6 remaining in print

[11] Wikipedia has a good article on the definition of progressive enhancement

[12] For a really neat trick involving a dynamically generated column to displaying `<abbr>` and `<dfn>` meanings (as well as somewhere for the user to write notes), try print previewing on Brian Suda's site

ABOUT THE AUTHOR

Natalie Downe is an excitable client-side web developer at Clearleft in Brighton, a perfectionist by nature and comes with the expertise and breadth of knowledge of a web agency background. Although front-end development and usability engineering are her first loves, Natalie still has fun dabbling with Python and poking the odd API. Natalie is also an experienced usability consultant and project manager.

10. 10 Ways To Get Design Approval

Paul Boag 24ways.org/200710

One of the most challenging parts of the web design process is getting design sign off. It can prove time consuming, demoralizing and if you are not careful can lead to a dissatisfied client. What is more you can end up with a design that you are ashamed to include in your portfolio.

How then can you ensure that the design you produce is the one that gets built? How can you get the client to sign off on your design? Below are 10 tips learnt from years of bitter experience.

1. DEFINE THE ROLE OF THE CLIENT AND DESIGNER

Many of the clients you work with will not have been involved in a web project before. Even if they have they may have worked in a very different way to what you would expect. Take the time at the beginning of the project to explain their role in the design of the site.

The best approach is to emphasis that their job is to focus on the needs of their users and business. They should concentrate on the broad issues, while you worry about the details of layout, typography and colour scheme.

By clarifying what you expect from the client, you help them to provide the right kind of input throughout the process.

2. UNDERSTAND THE BUSINESS

Before you open up Photoshop or put pen to paper, take the time to make sure you properly understand not only the brief but the organization behind the site. By understanding their business objectives, organizational structure and marketing strategy your design decisions will be better informed.

You cannot rely upon the brief to provide all of the information you need. It is important to dig deeper and get as good an understanding of their business as possible. This information will prove invaluable when justifying your design decisions.

3. UNDERSTAND THE USERS

We all like to think of ourselves as user centric designers, but exactly how much effort do you put into knowing your users before beginning the design process?

Take the time to really understand them the best you can. Try to meet with some real prospective users and get to know their needs. Failing that work with the client to produce user personas to help picture exactly what kind of people they are.

Understanding your users not only improves the quality of your work, but also helps move the discussion away from the personal preferences of the client, to the people who's opinion really matters.

4. AVOID MULTIPLE CONCEPTS

Many clients like the idea of having the option to choose between multiple design concepts. However, although on the surface this might appear to be a good idea it can ultimately be counterproductive for design sign off.

In a world of limited budgets it is unwise to waste money on producing designs that are ultimately going to be thrown away. The resources would be better spent refining a single design through multiple iterations.

What is more, multiple concepts often cause confusion rather than clarity. It is common for a client to request one element from one design and another from the second. As any designer knows this seldom works.

5. USE MOOD BOARDS

Clients are often better at expressing what they don't like than what they do. This is one of the reasons why they favour producing multiple design concepts. An alternative less costly approach is to create a series of mood boards. These boards contain a collection of colours, typography and imagery which represent different "moods" or directions, which the design could take.

Mood boards are quick and easy to produce allowing you to try out various design approaches with the client without investing the time needed to produce complete design concepts. This means that by the time you develop a concept the client and designer have already established an understanding about the direction of the design.

6. SAY WHAT YOU LIKE

It is not uncommon for a client to ask for a design that looks similar to another site they like. The problem is that it can often be hard to establish exactly what it is about the site that attracts them. Also in many cases the sites they like are not something you are keen to emulate!

A better approach that was suggested to me by Andy Budd is to show them sites that you think the design should emulate. Keep a collection of screen captures from well designed sites and pick out a few that are relevant to that particular client. Explain why you feel these designs might suit their project and ask for their feedback. If they don't like your choices then expose them to more of your collection and see what they pick out.

7. WIREFRAME THE HOMEPAGE

Often clients find it hard to distinguish between design and content and so sometimes reject a design on the basis that the content is not right. This is particularly true when signing off the homepage.

You may therefore find it useful to establish the homepage content before producing the design. That way once they see the design they will not be distracted by the content. One of the best ways to do this is by producing a basic wireframe consisting of a series of content boxes. Once this has been approved you will find the sign off of design much easier.

8. PRESENT YOUR DESIGNS

Although it is true that a good design should speak for itself it still needs presenting to the client. The client needs to understand why you have made the design decisions you have, otherwise they will judge the design purely on personal preference.

Talk them through the design explaining how it meets the needs of their users and business objectives. Refer to the mood boards and preferred sites the client approved and explain how the design is a continuation of those. Never simply email the design through and hope the client interprets your work correctly!

9. PROVIDE WRITTEN SUPPORTING MATERIAL

Unfortunately, no matter how well you justify the design to the client he is almost certain to want to show it to others. He may need his bosses approval or require internal buy in. At the very least he is going to want to get a second opinion from a friend or colleague.

The problem with this is that you are not going to be there to present to these people in the same way you did for the client. You cannot expect the client to present your ideas as well as you did. The reality is that you have lost control of how the design is perceived.

One way to minimize this problem is to provide written documentation supporting the design. This can be a summary of the presentation you gave to the client and allows him to distribute this along with the design. By putting a written explanation with the design you ensure that everybody who sees it gets the same message.

10. CONTROL THE FEEDBACK

My final piece of advice for managing design sign off is to control the way you receive feedback. A clients natural inclination will be to give you his personal opinion on the design. This is reinforced because you ask them what **they** think of the design. Instead ask them what their users will think of the design. Encourage them to think from the users perspective.

Also encourage them to keep that overarching focus I talked about in my first tip. Their tendency will be to try to improve the design, however that should be your problem not theirs. The role of a client should be to defend the needs of their users and business not do the design. Encourage the client to make comments such as "I am not sure that my female users will like the masculine colours" rather than "can we make the whole design pink." It is down to them to identify the problems and for you as the designer to find the most appropriate solution.

So there you have it. My 10 tips to improve design sign off. Will this ensure design approval every time? Unfortunately not. However it should certainly help smooth the way.

ABOUT THE AUTHOR

Paul Boag is a user experience consultant based in Dorset, England. He's the founder of Headscape, a successful web design agency and hosts the longest running web design podcast at boagworld.com. He also writes for web design publications and speaks at various conferences and workshops.

11. Tracking Christmas Cheer with Google Charts

Brian Suda

24ways.org/200711

A note from the editors: Since this article was written Google has retired the Charts API.

Let's get something out in the open: I love statistics. As an informatician I can't get enough graphs, charts, and numbers. So you can imagine when Google released their Charts API I thought Christmas had come early. I immediately began to draw up graphs for the holiday season using the new API; and using my new found chart-making skills I'll show you what you can and can't do with Google Charts.

MUMMY, IT'S MY FIRST CHART

The Google Charts API allows you to send data to Google; in return they give you back a nicely-rendered graph. All the hard work is done on Google's servers — you need only reference an image in your HTML. You pass along the data — the numbers for the charts, axis labels, and so on —

in the query string of the image's URL. If you want to add charts to your blog or web site, there's probably no quicker way to get started.

Here's a simple example: if we add the following line to an HTML page:

```
<img src="http://chart.apis.google.com/
chart?cht=lc&chs=200x125&chd=s:ZreelPuevfgznf2008" />
```

Then we'll see the line graph in Figure 1 appear in our page. That graph is hosted on Google's own server[1]: `http://chart.apis.google.com/`.

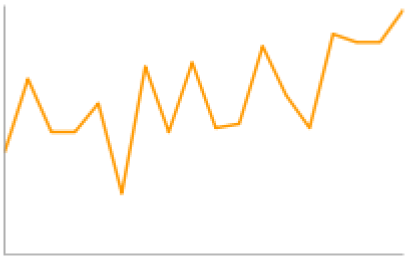


Figure 1: A simple example of a line graph created with Google Charts.

If you look at the URL used in the example you'll notice we're passing some parameters along in the query string (the bit after the question mark). The query string looks like this:

```
cht=lc&chs=200x125&chd=s:ZreelPuevfgznf2008
```

It's contains everything Google Charts needs to draw the graph. There are three parameters in the query string:

- `cht`; this specifies the type of chart Google Charts will generate (in this case, `lc` is a line chart).
- `chs`, the value of which is `200x125`; this defines the chart's size (200 pixels wide by 125 pixels high).
- `chd`, the value of which is `s:ZreelPuevfgznf2008`; this is the actual chart data, which we'll discuss in more detail later.

These three parameters are the minimum you need to send to Google Charts in order to create a chart. There are lots more parameters you can send too (giving you more choice over how a chart is displayed), but you have to include at least these three before a chart can be created. Using these three parameters you can create pie charts, scatter plots, Venn diagrams, bar charts (and more) up to 1,000 pixels wide or 1,000 pixels high (but no more than 300,000 pixels in total).

CHRISTMAS PIE

After I discovered the option to create a pie chart I instantly thought of graphing all the types of food and beverages that I'll consume at this year's Christmas feast. I can represent each item as a percentage of all the food on a pie chart (just thinking about that makes me hungry).

By changing the value of the `cht` parameter in the image's query string I can change the chart type from a line chart to a pie chart. Google Charts offers two different types of pie chart: a fancy three-dimensional version and a two-dimensional overhead version. I want to stick with the latter, so I need to change `cht=lc` to `cht=p` (the `p` telling Google Charts to create a pie chart; if you want the three-dimensional version, use `cht=p3`). As a pie chart is circular I also need to adjust the size of the chart to make it square. Finally, it would be nice to add a title to the graph. I can do this by adding the optional parameter, `chtt`, to the end of the image URL. I end up with the chart you see in Figure 2.

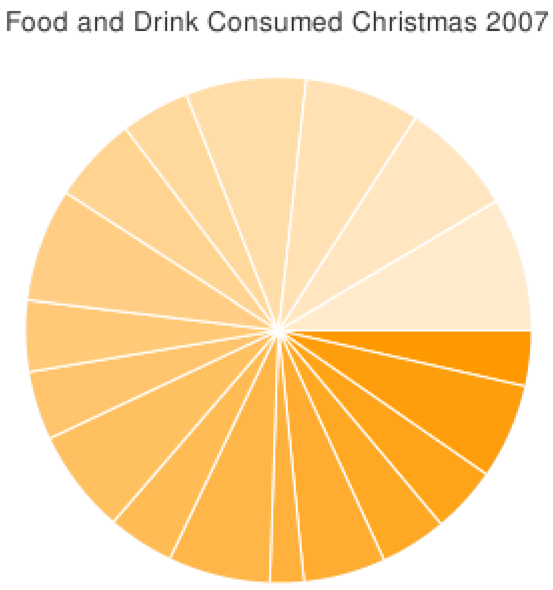


Figure 2: Pie chart with a title.

To add this chart to your own page, you include the following (notice that you can't include spaces in URLs, so you need to encode them as plus-signs.):

```
<img src="http://chart.apis.google.com/
chart?chtt=Food+and+Drink+Consumed+Christmas+2007&cht=p&chs=300x300&ch
/>
```

Ok, that's great, but there are still two things I want to do before I can call this pie chart complete. First I want to label each slice of the pie. And second I want to include the proper data (at the moment the slices are meaningless). If 2007 is anything like 2006, the break down will be roughly as follows:

Egg nog	10%
Christmas Ham	20%
Milk (not including egg nog)	8%
Cookies	25%
Roasted Chestnuts	5%
Chocolate	3%
Various Other Beverages	15%
Various Other Foods	9%
Snacks	5%

I have nine categories of food and drink to be tracked, so I need nine slice labels. To add these to the chart, I use the chl parameter. All nine labels are sent in one value; I use the vertical-pipe character, |, to separate them. So I need to append the following to the query string:

```
chl=Egg+nog|Christmas+Ham|Milk+(not+including+egg+nog)|Cookies|Roast+C
```

Next I need to add the corresponding percentage values to the chart labels. Encoding the chart data is the trickiest part of the Google Charts API — but by no means complicated. There are three different ways to encode your data on a chart. As I'm only dealing with small numbers, I'm going to use what Google calls *simple encoding*.

Simple encoding offers a sixty-two value spectrum in which to represent data. Remember the mandatory option, `chd`, from the first example? The value for this is split into two parts: the type of encoding and the graph data itself. These two parts are separated with a colon. To use simple encoding, the first character of the chd option must be a lower case s. Follow this with a colon and everything after it is considered data for the graph.

In simple encoding, you have sixty-two values to represent your data. These values are lowercase and uppercase letters from the Latin alphabet (fifty-two characters in total) and the digits 0 to 9. Each letter of the alphabet represents a single number: A equals 0, B equals

1, and so on up to Z, which equals 25; a equals 26, b equals 27, and so on up to z, which equals 51. The ten digits represent the numbers 52 to 61: 0 equals 52, 1 equals 53, and 9 equals 61.

In the previous two examples we used the string `ZreelPuevfgznf2008` as our chart data; the Z is equal to 25, the r is equal to 42, the e is equal to 30, and so on. I want to encode the percentage values 10, 20, 8, 25, 5, 3, 15, 9 and 5, so in simple encoding I would use the string `KUIZFDPJF`.

If you think figuring this out for each chart may make your head explode, don't worry: help is out there.

Do you remember I said I needed to change the image dimensions to be square, to accommodate the pie chart? Well now I'm including labels I need even more room. And as I'm in a Christmassy mood I'm going to add some festive colours too.

The optional `chco` parameter is used to change the chart color. You set this using the same hexadecimal ("hex") notation found in CSS. So let's make our pie chart green by adding `chco=00AF33` (don't start it with a hash character as in CSS) to the image URL. If we only specify one hex colour for the pie chart Google Charts will use shades of that colour for each of the slices. To choose your own colours, pass a comma separated list of colours. The "Milk" and "Cookies" slices were consumed together, so we can make those two slices more of a redish colour. I'll use shades of green for the other slices. My `chco` parameter now looks like this:

```
chco=00AF33,4BB74C,EE2C2C,CC3232,33FF33,66FF66,9AFF9A,C1FFC1,CCFFCC.
```

After all this, I'm left with the following URL:

```
http://chart.apis.google.com/
chart?chco=00AF33,4BB74C,EE2C2C,CC3232,33FF33,66FF66,9AFF9A,C1FFC1,CCF
```

What does that give us? I'm glad you asked. I have the rather beautiful 600-pixel wide pie chart you see in Figure 3.

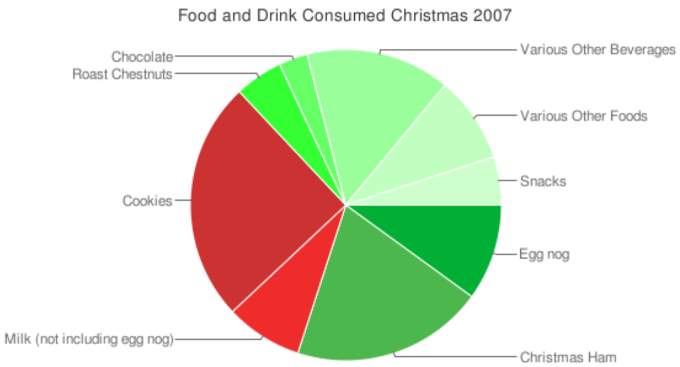


Figure 3: A Christmassy pie chart with labels.

BUT I DON'T LIKE PIE CHARTS

The pie chart was invented by the Scottish polymath William Playfair in 1801. But not everyone is as excited by pie charts as wee Billy, so if you're an anti-pie-chartist, what can you do?

You can easily reuse the same data but display it as a bar graph in a snap. The first thing we need to do is change the value of the `cht` parameter from `p` to `bhg`. This creates a horizontal bar graph (you can request a vertical bar graph using `bvg`). The data and labels all remain the same, but we need to decide where the labels will appear. I'll talk more about how to do all this in the next section.

In Figure 4 you'll see the newly-converted bar graph. The URL for the graph is:

```
http://chart.apis.google.com/
chart?cht=bhg&chs=600x300&chd=s:KUIZFDPJF&chxt=x,y&chtt=Food+and+Drink
```

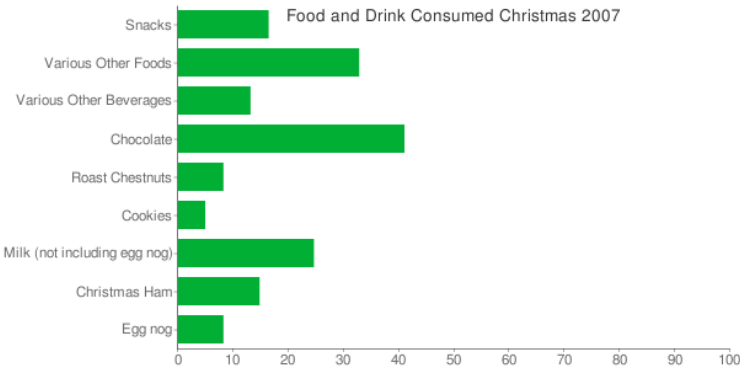


Figure 4: The pie chart from Figure 3 represented as a bar chart.

TWO LINES, ONE GRAPH

Pie charts and bar charts are interesting, but what if I want to compare last year's Christmas cheer with this year's? That sounds like I'll need two lines on one graph.

The code is much the same as the previous examples; the most obvious difference is I need to set up the chart as a line graph. Creating some dummy values for the required parameters, I end up with:

```
<img src="http://chart.apis.google.com/
chart?chs=800x300&cht=lxy&chd=t:0,100|0,100" />
```

The `chs=800x300` sets the dimensions of the new chart, while `cht=lxy` describes the type of chart we are using (in this case a line chart with x and y co-ordinates). For the chart data I'm going to demostrate a different encoding, *text encoding*. To use this I start the value of the chd parameter with "t:" instead of "s:", and follow it with a list of x coordinates, a vertical pipe, |, and a list of y coordinates. Given the URL above, Google Charts will render the chart shown in Figure 5.

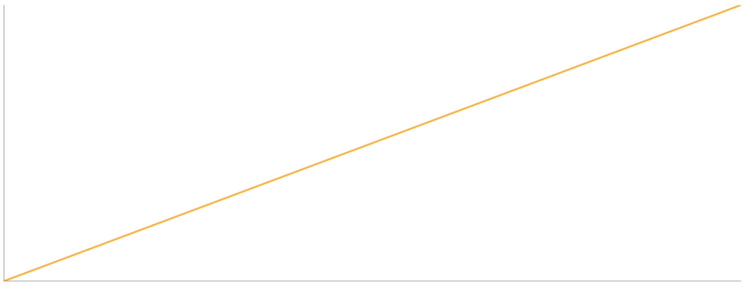


Figure 5: A simple line graph with x and y co-ordinates.

To make this graph a little more pleasing to the eye, I can add much the same as I did to the pie chart. I'll add a chart title. Maybe something like "Projected Christmas Cheer for 2007". Just as before I would add a `chtt` parameter to the image URL:

```
&chtt=Projected+Christmas+Cheer+for+2007
```

Next, let's add some labels on the y axis to represent a scale from 0 to 100. On the x axis let's label for the most important days of December. To do this I need to use the chart axis type parameter, `chxt`. This allows us to specify the axes and associate some labels with them. As I'm only interested in the y-axis (to the left of the chart) and the x-axis (below the chart), we add `chxt=x,y` to our image URL.

Now I need my label data. This is slightly more tricky because I want the data evenly spaced without labelling every item. The parameter for labels is `chxl`, the chart axis label. You match a label to an axis by using a number. So `0:Label1` is the zero index of `chxt` — in this case the x-axis.

`1:Label2` is the first index of `chxt` — the y-axis. The order of these parameters or labels doesn't matter as long as you associate them to their `chxt` correctly.

The next thing to know about `chxl` is that you can add an empty label. Labels are separated by vertical pipe; if you don't put any text in a label, you just leave the two vertical pipes empty ("||") and Google Charts will allocate space but no label.

For our vertical y axis, we want to label only 50% and 100% on the graph and plot them in their respective places. Since the y-axis is the second item, 1: (remember to start counting at zero), we add ten spaces to our image URL, `chxl=1:||||||50|||||100` This will output the 50 halfway and the 100 at the top; all the other spaces will be empty.

We can do the same thing to get specific dates along the x-axis as well. Let's add the 1st of December, St. Nick's Day (the 6th), Christmas Day, Boxing Day (a holiday common in the UK and the Commonwealth, on the 26th), and the final day of the month, the 31st. Since this is the x-axis I'll use `0:` as a reference in the `chxt` parameter tell Google Charts which axis to label. In full, the `chxl` parameter now looks like:

```
chxl=1:||||||50|||||100|0:|Dec+1st|||||6th||||10th|||||15th|||||20th||
```

That's pretty.

Before we begin to graph our data, I'll do one last thing: add some grid lines to the chart so to better connect the data to the labels. The parameter for this is `chg`, short for chart grid lines. The parameter takes four comma-separated arguments. The first is the x-axis spacing for the grid. I have thirty-one days, so I need thirty vertical lines. The chart is 100% wide, so 3.33 (100 divided by 30) is the required spacing.

As for the y-axis: the axis goes up to 100% but we probably only need to have a horizontal line every 10%, so the required spacing is 10 (100 divided by 10). That is the second argument.

The last two arguments control the dash-style of the grid-lines. The first number is the length of the line dash and the second is the space between the dashes. So 6,3 would mean a six-unit dash with a three-unit space. I like a ratio of 1,3 but you can change this as you wish. Now that I have the four arguments, the `chg` parameter looks like:

```
chg=3.333,10,1,3
```

If I add that to the chart URL I end up with:

```
http://chart.apis.google.com/
chart?chs=800x300&cht=lxy&chd=t:0,100|0,100&chtt=Projected+Christmas+C
```

Which results in the chart shown in Figure 6.

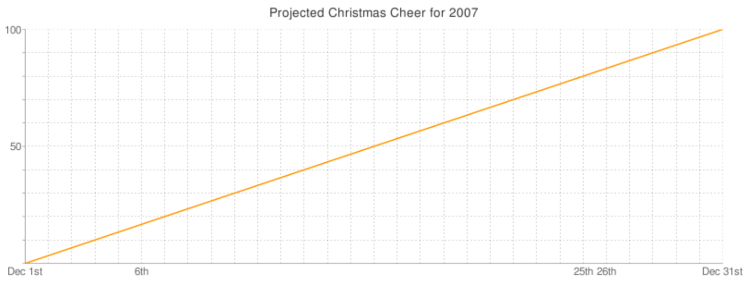


Figure 6: Chart ready to receive the Christmas cheer values.

REAL DATA

Now the chart is ready I can add historical data from 2006 and current data from 2007.

Having a look at last year's cheer levels we find some highs and lows through-out the month:

Dec 1st	Advent starts; life is good	30%
Dec 6th	St. Nick's Day, awake to find good things in my shoes	45%
Dec 8th	Went Christmas carolling, nearly froze	20%
Dec 10th	Christmas party at work, very nice dinner	50%
Dec 18th	Panic Christmas shopping, hate rude people	15%
Dec 23rd	Off Work, home eating holiday food	80%
Dec 25th	Opened presents, good year, but got socks again from Grandma	60%
Dec 26th	Boxing Day; we're off and no one knows why	70%
Dec 28th	Third day of left overs	40%
Dec 29th	Procured some fireworks for new years	55%
Dec 31st	New Year's Eve	80%

Since I'm plotting data for 2006 and 2007 on the same graph I'll need two different colours — one for each year's line — and a key to denote what each colour represents. The key is controlled by the `chdl` (chart data legend) parameter. Again, each part of the parameter is separated by a vertical pipe, so for two labels I'll use `chdl=2006|2007`. I also want to colour-code them, so I'll need to add the `chco` as I did for the pie chart. I want a red line and a green line, so I'll use `chco=458B00,CD2626` and add this to the image URL.

Let's begin to plot the 2006 data on the Chart, replacing our dummy data of `chd=t:0,100|0,100` with the correct information. The `chd` works by first listing all the x coordinates (each separated by a comma), then a vertical pipe, and then all the y coordinates (also comma-separated). The chart is 100% wide, so I need to convert the days into a percentage of the month.

The 1st of December is 0 and the 31st is 100. Everything else is somewhere in between. Our formula is:

(d – 1) × 100 ÷ (31 – 1)

Where d is the day of the month. The formula states that each day will be printed every 3.333 units; so the 6th of December will be printed at 16.665 units. I can repeat the process for the other dates listed to get the following x coordinates: 0,16.7,23.3,33.3,60,76.7,83.3,86.7,93.3,96.7. The y axis coordinates are easy because our scale is 100%, just like our rating, so we can simply copy them across as 30,45,20,50,15,80,60,70,40,55,80. This gives us a final `chd` value of:

```
chd=t:0,16.7,23.3,33.3,60,76.7,83.3,86.7,93.3,96.7,100|30,45,20,50,15,
```

Onto 2007: I can put the data for the month so far to see how we are trending.

Dec 1st	Christmas shopping finished already	50%
Dec 4th	Computer hard disk drive crashed (not Christmas related accident, but put me in a bad mood)	10%
Dec 6th	Missed St. Nick's Day completely due to travelling	30%
Dec 9th	Dinner with friends before they travel	55%
Dec 11th	24ways article goes live	60%

Using the same system we did for 2006, I can take the five data points and plot them on the chart. The new x axis values will be 0,10,16.7,26.7 and the new y axis 50,10,30,65. We incorporate those into the image URL by appending these values onto the chd parameter we already have, which then becomes:

```
chd=t:0,16.7,23.3,33.3,60,76.7,83.3,86.7,93.3,96.7,100|30,45,20,50,15,
```

Passing this to Google Charts results in Figure 7.

```
http://chart.apis.google.com/
chart?chs=800x300&cht=lxy&chd=t:0,100|0,100&chtt=Projected+Christmas+C
```

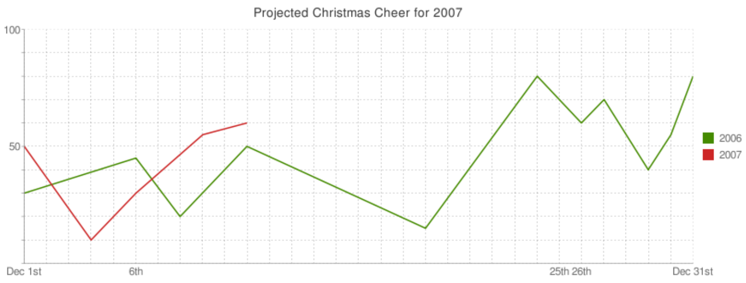


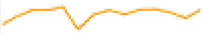
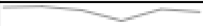
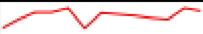
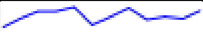
Figure 7: Projected Christmas cheer for 2006 and 2007.

DID SOMEONE MENTION EDWARD TUFTE?

Google Charts are a robust set of chart types that you can create easily and freely using their API. As you can see, you can graph just about anything you want using the line graph, bar charts, scatter plots, venn diagrams and pie charts. One type of chart conspicuously missing from the API is sparklines. Sparklines were proposed by Edward Tufte as “small, high resolution graphics embedded in a context of words, numbers, images”. They can be extremely useful, but can you create them in Google Charts?

The answer is: “Yes, but it’s an undocumented feature”. (The usual disclaimer about undocumented features applies.)

If we take our original line graph example, and change the value of the `cht` parameter from `lc` (line chart) to `lfi` (financial line chart) the axis-lines are removed. This allows you to make a chart — a sparkline — small enough to fit into a sentence. Google uses the `lfi` type all throughout the their financial site, but it's not yet part of the official API.

MerryChristmas		`http://chart.apis.google.com/ chart?cht=lfi&chs=100x15&chd=s:Mer`
24ways		`http://chart.apis.google.com/ chart?cht=lfi&chs=100x15&chd=s:24w`
HappyHolidays		`http://chart.apis.google.com/ chart?cht=lfi&chs=100x15&chd=s:Hap`
HappyNewYear		`http://chart.apis.google.com/ chart?cht=lfi&chs=100x15&chd=s:Hap`

SUMMARY

The new Google Charts API is a powerful method for creating charts and graphs of all types. If you apply a little bit of technical skill you can create pie charts, bar graphs, and even sparklines as and when you need them. Now you've finished ready the article I hope you waste no more time: go forth and chart!

FURTHER READING

- Google Charts API
- More on Google Charts
- How to handle negative numbers
- 12 Days of Christmas Pie Chart

[1] In order to remain within the 50,000 requests a day limit the Google Charts API imposes, chart images on this page have been cached and are being served from our own servers. But the URLs work – try them!

ABOUT THE AUTHOR

Brian Suda is a master informatician working to make the web a better place little by little everyday. Since discovering the Internet in the mid-90s, Brian Suda has spent a good portion of each day connected to it. His own little patch of Internet is http://suda.co.uk, where many of his past projects and crazy ideas can be found.

Photo: Jeremy Keith

12. Unobtrusively Mapping Microformats with jQuery

Simon Willison 24ways.org/200712

Microformats are everywhere. You can't shake an electronic stick these days without accidentally poking a microformat-enabled site, and many developers use microformats as a matter of course. And why not? After all, why invent your own class names when you can re-use pre-defined ones that give your site extra functionality for free?

Nevertheless, while it's good to know that users of tools such as Tails and Operator will derive added value from your shiny semantics, it's nice to be able to reuse that effort in your own code.

We're going to build a map of some of my favourite restaurants in Brighton. Fitting with the principles of unobtrusive JavaScript, we'll start with a semantically marked up list of restaurants, then use JavaScript to add the map, look up the restaurant locations and plot them as markers.

We'll be using a couple of powerful tools. The first is jQuery, a JavaScript library that is ideally suited for unobtrusive scripting. jQuery allows us to manipulate elements on the page based on their CSS selector, which makes it easy to extract information from microformats.

The second is Mapstraction, introduced here by Andrew Turner a few days ago. We'll be using Google Maps in the background, but Mapstraction makes it easy to change to a different provider if we want to later.

GETTING STARTED

We'll start off with a simple collection of microformatted restaurant details, representing my seven favourite restaurants in Brighton. The full, unstyled list can be seen in restaurants-plain.html. Each restaurant listing looks like this:

```
<li class="vcard">
  <h3><a class="fn org url"
href="http://www.riddleandfinns.co.uk/">Riddle &
Finns</a></h3>
  <div class="adr">
    <p class="street-address">12b Meeting House Lane</p>
    <p><span class="locality">Brighton</span>, <abbr
class="country-name" title="United Kingdom">UK</abbr></p>
    <p class="postal-code">BN1 1HB</p>
  </div>
  <p>Telephone: <span class="tel">+44 (0)1273 323
008</span></p>
```

```
  <p>E-mail: <a href="mailto:info@riddleandfinns.co.uk"
class="email">info@riddleandfinns.co.uk</a></p>
</li>
```

Since we're dealing with a list of restaurants, each hCard is marked up inside a list item. Each restaurant is an organisation; we signify this by placing the classes `fn` and `org` on the element surrounding the restaurant's name (according to the hCard spec, setting both `fn` and `org` to the same value signifies that the hCard represents an organisation rather than a person).

The address information itself is contained within a `div` of class `adr`. Note that the HTML `<address>` element is not suitable here for two reasons: firstly, it is intended to mark up contact details for the current document rather than generic addresses; secondly, address is an inline element and as such cannot contain the paragraphs elements used here for the address information.

A nice thing about microformats is that they provide us with automatic hooks for our styling. For the moment we'll just tidy up the whitespace a bit; for more advanced style tips consult John Allsop's guide from 24 ways 2006.

```
.vcard p {
  margin: 0;
}
.adr {
  margin-bottom: 0.5em;
}
```

To plot the restaurants on a map we'll need latitude and longitude for each one. We can find this out from their address using *geocoding*. Most mapping APIs include support for geocoding, which means we can pass the API an address and get back a latitude/longitude point. Mapstraction provides an abstraction layer around these APIs which can be included using the following script tag:

```
<script type="text/javascript"
src="http://mapstraction.com/src/
mapstraction-geocode.js"></script>
```

While we're at it, let's pull in the other external scripts we'll be using:

```
<script type="text/javascript"
src="jquery-1.2.1.js"></script>
<script src="http://maps.google.com/
maps?file=api&v=2&key=YOUR_KEY" type="text/
javascript"></script>
<script type="text/javascript"
src="http://mapstraction.com/src/
mapstraction.js"></script>
<script type="text/javascript"
src="http://mapstraction.com/src/
mapstraction-geocode.js"></script>
```

That's everything set up: let's write some JavaScript!

In jQuery, almost every operation starts with a call to the jQuery function. The function simulates method overloading to behave in different ways depending on the arguments passed to it. When writing unobtrusive JavaScript it's important to set up code to execute when the page has loaded to the point that the DOM is available to be manipulated. To do this with jQuery, pass a callback function to the jQuery function itself:

```
jQuery(function() {
  // This code will be executed when the DOM is ready
});
```

INITIALISING THE MAP

The first thing we need to do is initialise our map. Mapstraction needs a div with an explicit width, height and ID to show it where to put the map. Our document doesn't currently include this markup, but we can insert it with a single line of jQuery code:

```
jQuery(function() {
  // First create a div to host the map
  var themap = jQuery('<div id="themap"></div>').css({
    'width': '90%',
    'height': '400px'
  }).insertBefore('ul.restaurants');
});
```

While this is technically just a single line of JavaScript (with line-breaks added for readability) it's actually doing quite a lot of work. Let's break it down in to steps:

```
var themap = jQuery('<div id="themap"></div>')
```

Here's jQuery's method overloading in action: if you pass it a string that starts with a < it assumes that you wish to create a new HTML element. This provides us with a handy shortcut for the more verbose DOM equivalent:

```
var themap = document.createElement('div');
themap.id = 'themap';
```

Next we want to apply some CSS rules to the element. jQuery supports chaining, which means we can continue to call methods on the object returned by jQuery or any of its methods:

```
var themap = jQuery('<div id="themap"></div>').css({
  'width': '90%',
  'height': '400px'
})
```

Finally, we need to insert our new HTML element in to the page. jQuery provides a number of methods for element insertion, but in this case we want to position it directly before the `<ul>` we are using to contain our restaurants. jQuery's `insertBefore()` method takes a CSS selector indicating an element already on the page and places the current jQuery selection directly before that element in the DOM.

```
var themap = jQuery('<div id="themap"></div>').css({
  'width': '90%',
  'height': '400px'
}).insertBefore('ul.restaurants');
```

Finally, we need to initialise the map itself using Mapstraction. The Mapstraction constructor takes two arguments: the first is the ID of the element used to position the map; the second is the mapping provider to use (in this case `google`):

```
// Initialise the map
var mapstraction = new Mapstraction('themap','google');
```

We want the map to appear centred on Brighton, so we'll need to know the correct co-ordinates. We can use www.getlatlon.com to find both the co-ordinates and the initial map zoom level.

```
// Show map centred on Brighton
mapstraction.setCenterAndZoom(
  new LatLonPoint(50.82423734980143, -0.14007568359375),
  15 // Zoom level appropriate for Brighton city centre
);
```

We also want controls on the map to allow the user to zoom in and out and toggle between map and satellite view.

```
mapstraction.addControls({
  zoom: 'large',
  map_type: true
});
```

ADDING THE MARKERS

It's finally time to parse some microformats. Since we're using hCard, the information we want is wrapped in elements with the class `vcard`. We can use jQuery's CSS selector support to find them:

```
var vcards = jQuery('.vcard');
```

Now that we've found them, we need to create a marker for each one in turn. Rather than using a regular JavaScript for loop, we can instead use jQuery's `each()` method to execute a function against each of the hCards.

```
jQuery('.vcard').each(function() {
  // Do something with the hCard
});
```

Within the callback function, `this` is set to the current DOM element (in our case, the list item). If we want to call the magic jQuery methods on it we'll need to wrap it in another call to jQuery:

```
jQuery('.vcard').each(function() {
  var hcard = jQuery(this);
});
```

The Google maps geocoder seems to work best if you pass it the street address and a postcode. We can extract these using CSS selectors: this time, we'll use jQuery's `find()` method which searches within the current jQuery selection:

```
var streetaddress = hcard.find('.street-address').text();
var postcode = hcard.find('.postal-code').text();
```

The `text()` method extracts the text contents of the selected node, minus any HTML markup.

We've got the address; now we need to geocode it. Mapstraction's geocoding API requires us to first construct a `MapstractionGeocoder`, then use the `geocode()` method to pass it an address. Here's the code outline:

```
var geocoder = new MapstractionGeocoder(onComplete,
'google');
geocoder.geocode({'address': 'the address goes here');
```

The `onComplete` function is executed when the geocoding operation has been completed, and will be passed an object with the resulting point on the map. We just want to create a marker for the point:

```
var geocoder = new MapstractionGeocoder(function(result)
{
  var marker = new Marker(result.point);
  mapstraction.addMarker(marker);
}, 'google');
```

For our purposes, joining the street address and postcode with a comma to create the address should suffice:

```
geocoder.geocode({'address': streetaddress + ', ' +
postcode});
```

There's one last step: when the marker is clicked, we want to display details of the restaurant. We can do this with an info bubble, which can be configured by passing in a string of HTML. We'll construct that HTML using jQuery's `html()` method on our `hcard` object, which extracts the HTML contained within that DOM node as a string.

```
var marker = new Marker(result.point);
marker.setInfoBubble(
  '<div class="bubble">' + hcard.html() + '</div>'
);
mapstraction.addMarker(marker);
```

We've wrapped the bubble in a div with class `bubble` to make it easier to style. Google Maps can behave strangely if you don't provide an explicit width for your info bubbles, so we'll add that to our CSS now:

```
.bubble {
  width: 300px;
}
```

That's everything we need: let's combine our code together:

```
jQuery(function() {
  // First create a div to host the map
  var themap = jQuery('<div id="themap"></div>').css({
    'width': '90%',
    'height': '400px'
  }).insertBefore('ul.restaurants');
  // Now initialise the map
  var mapstraction = new Mapstraction('themap','google');
  mapstraction.addControls({
    zoom: 'large',
    map_type: true
  });
  // Show map centred on Brighton
  mapstraction.setCenterAndZoom(
    new LatLonPoint(50.82423734980143,
-0.14007568359375),
    15 // Zoom level appropriate for Brighton city centre
  );
  // Geocode each hcard and add a marker
  jQuery('.vcard').each(function() {
    var hcard = jQuery(this);
    var streetaddress =
hcard.find('.street-address').text();
    var postcode = hcard.find('.postal-code').text();
    var geocoder = new
MapstractionGeocoder(function(result) {
      var marker = new Marker(result.point);
      marker.setInfoBubble(
        '<div class="bubble">' + hcard.html() + '</div>'
      );
      mapstraction.addMarker(marker);
    }, 'google');
    geocoder.geocode({'address': streetaddress + ', ' +
postcode});
  });
});
```

Here's the finished code.

There's one last shortcut we can add: jQuery provides the `$` symbol as an alias for jQuery. We could just go through our code and replace every call to `jQuery()` with a call to `$()`, but this would cause incompatibilities if we ever attempted to use our script on a page that also includes the Prototype library. A more robust approach is to start our code with the following:

```
jQuery(function($) {
  // Within this function, $ now refers to jQuery
  // ...
});
```

jQuery cleverly passes itself as the first argument to any function registered to the DOM ready event, which means we can assign a local `$` variable shortcut without affecting the `$` symbol in the global scope. This makes it easy to use jQuery with other libraries.

LIMITATIONS OF GEOCODING

You may have noticed a discrepancy creep in to the last example: whereas my original list included seven restaurants, the geocoding example only shows five. This is because the Google Maps geocoder incorporates a rate limit: more than five lookups in a second and it starts returning error messages instead of regular results.

In addition to this problem, geocoding itself is an inexact science: while UK postcodes generally get you down to the correct street, figuring out the exact point on the street from the provided address usually isn't too accurate (although Google do a pretty good job).

Finally, there's the performance overhead. We're making five geocoding requests to Google for every page served, even though the restaurants themselves aren't likely to change location any time soon. Surely there's a better way of doing this?

Microformats to the rescue (again)! The geo microformat suggests simple classes for including latitude and longitude information in a page. We can add specific points for each restaurant using the following markup:

```
<li class="vcard">
  <h3 class="fn org">E-Kagen</h3>
  <div class="adr">
    <p class="street-address">22-23 Sydney Street</p>
    <p><span class="locality">Brighton</span>, <abbr
class="country-name" title="United Kingdom">UK</abbr></p>
    <p class="postal-code">BN1 4EN</p>
  </div>
  <p>Telephone: <span class="tel">+44 (0)1273 687
068</span></p>
  <p class="geo">Lat/Lon:
    <span class="latitude">50.827917</span>,
    <span class="longitude">-0.137764</span>
  </p>
</li>
```

As before, I used www.getlatlon.com to find the exact locations – I find satellite view is particularly useful for locating individual buildings.

Latitudes and longitudes are great for machines but not so useful for human beings. We could hide them entirely with `display: none`, but I prefer to merely de-emphasise them (someone might want them for their GPS unit):

```
.vcard .geo {
  margin-top: 0.5em;
  font-size: 0.85em;
  color: #ccc;
}
```

It's probably a good idea to hide them completely when they're displayed inside an info bubble:

```
.bubble .geo {
  display: none;
}
```

We can extract the co-ordinates in the same way we extracted the address. Since we're no longer geocoding anything our code becomes a lot simpler:

```
$('.vcard').each(function() {
  var hcard = $(this);
  var latitude = hcard.find('.geo .latitude').text();
  var longitude = hcard.find('.geo .longitude').text();
  var marker = new Marker(new LatLonPoint(latitude,
longitude));
  marker.setInfoBubble(
    '<div class="bubble">' + hcard.html() + '</div>'
  );
  mapstraction.addMarker(marker);
});
```

And here's the finished geo example.

FURTHER READING

We've only scratched the surface of what's possible with microformats, jQuery (or just regular JavaScript) and a bit of imagination. If this example has piqued your interest, the following links should give you some more food for thought.

- The hCard specification
- Notes on parsing hCards
- jQuery for JavaScript programmers – my extended tutorial on jQuery.
- Dann Webb's Sumo – a full JavaScript library for parsing microformats, based around some clever metaprogramming techniques.
- Jeremy Keith's Adactio Austin – the first place I saw using microformats to unobtrusively plot locations on a map. Makes clever use of hEvent as well.

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

13. CSS for Accessibility

Ann McMeekin 24ways.org/200713

CSS is magical stuff. In the right hands, it can transform the plainest of (well-structured) documents into a visual feast. But it's not all fur coat and nae knickers (as my granny used to say). Here are some simple ways you can use CSS to improve the usability and accessibility of your site.

Even better, no sexy visuals will be harmed by the use of these techniques. Promise.

Nae knickers

This is less of an accessibility tip, and more of a reminder to check that you've got your body background colour specified.

If you're sitting there wondering why I'm mentioning this, because it's a really basic thing, then you might be as surprised as I was to discover that from a sample of over 200 sites checked last year, 35% of UK local authority websites were missing their body background colour.

Forgetting to specify your body background colour can lead to embarrassing gaps in coverage, which are not only unsightly, but can prevent your users reading the text on your site if they use a different operating system colour scheme.

All it needs is the following line to be added to your CSS file:

```
body {background-color: #fff;}
```

If you pair it with

```
color: #000;
```

… you'll be assured of maintaining contrast for any areas you inadvertently forget to specify, no matter what colour scheme your user needs or prefers.

Even better, if you've got standard reset CSS you use, make sure that default colours for background and text are specified in it, so you'll never be caught with your pants down. At the very least, you'll have a white background and black text that'll prompt you to change them to your chosen colours.

Elbow room

Paying attention to your typography is important, but it's not just about making it look nice.

Careful use of the line-height property can make your text more readable, which helps everyone, but is particularly helpful for those with dyslexia, who use screen magnification or simply find it uncomfortable to read lots of text online.

When lines of text are too close together, it can cause the eye to skip down lines when reading, making it difficult to keep track of what you're reading across.

So, a bit of room is good.

That said, when lines of text are too far apart, it can be just as bad, because the eye has to jump to find the next line. That not only breaks up the reading rhythm, but can make it much more difficult for those using Screen Magnification (especially at high levels of magnification) to find the beginning of the next line which follows on from the end of the line they've just read.

Using a line height of between 1.2 and 1.6 times normal can improve readability, and using unit-less line heights help take care of any pesky browser calculation problems.

For example:

```
p {
  font-family: "Lucida Grande", Lucida, Verdana, 
Helvetica, sans-serif;
  font-size: 1em;
  line-height: 1.3;
}
```

or if you want to use the shorthand version:

```
p {
  font: 1em/1.3 "Lucida Grande", Lucida, Verdana,
Helvetica, sans-serif;
}
```

View some examples of different line-heights, based on default text size of 100%/1em.

Further reading on Unitless line-heights from Eric Meyer.

Transformers: Initial case in disguise

Nobody wants to shout at their users, but there are some occasions when you might legitimately want to use uppercase on your site.

Avoid screen-reader pronunciation weirdness (where, for example, CONTACT US would be read out as Contact U S, which is not the same thing – unless you really are offering your users the chance to contact the United States) caused by using uppercase by using title case for your text and using the often neglected text-transform property to fake uppercase.

For example:

```
.uppercase {
  text-transform: uppercase
}
```

Don’t overdo it though, as uppercase text is harder to read than normal text, not to mention the whole SHOUTING thing.

Linky love

When it comes to accessibility, keyboard only users (which includes those who use voice recognition software) who can see just fine are often forgotten about in favour of screen reader users.

This Christmas, share the accessibility love and light up those links so all of your users can easily find their way around your site.

THE LINK OUTLINE

AKA: the focus ring, or that dotted box that goes around links to show users where they are on the site.

The techniques below are intended to supplement this, not take the place of it. You may think it’s ugly and want to get rid of it, especially since you’re going to the effort of tarting up your links.

Don’t.

Just don’t.

THE NON-UNDERLINED UNDERLINE

If you listen to Jacob Nielsen, every link on your site should be underlined so users know it’s a link.

You might disagree with him on this (I know I do), but if you are choosing to go with underlined links, in whatever state, then remove the default underline and replacing it with a border that’s a couple of pixels away from the text.

The underline is still there, but it’s no longer cutting off the bottom of letters with descenders (e.g., g and y) which makes it easier to read.

This is illustrated in Examples 1 and 2.

You can modify the three lines of code below to suit your own colour and border style preferences, and add it to whichever link state you like.

```
text-decoration: none;
border-bottom: 1px #000 solid;
padding-bottom: 2px;
```

STANDING OUT FROM THE CROWD

Whatever way you choose to do it, you should be making sure your links stand out from the crowd of normal text which surrounds them when in their default state, and especially in their hover or focus states.

A good way of doing this is to reverse the colours when on hover or focus.

WELL-FOCUSED

Everyone knows that you can use the `:hover` pseudo class to change the look of a link when you mouse over it, but, somewhat ironically, people who can see and use a mouse are the group who least need this extra visual clue, since the cursor handily (sorry) changes from an arrow to a hand.

So spare a thought for the non-pointing device users that visit your site and take the time to duplicate that hover look by using the `:focus` pseudo class.

Of course, the internets being what they are, it's not quite that simple, and predictably, Internet Explorer is the culprit once more with it's frustrating lack of support for `:focus`. Instead it applies the `:active` pseudo class whenever an anchor has focus.

What this means in practice is that if you want to make your links change on focus as well as on hover, you need to specify focus, hover **and** active.

Even better, since the look and feel necessarily has to be the same for the last three states, you can combine them into one rule.

So if you wanted to do a simple reverse of colours for a link, and put it together with the non-underline underlines from before, the code might look like this:

```
a:link {
  background: #fff;
  color: #000;
  font-weight: bold;
  text-decoration: none;
  border-bottom: 1px #000 solid;
  padding-bottom: 2px;
}
a:visited {
  background: #fff;
  color: #800080;
  font-weight: bold;
  text-decoration: none;
  border-bottom: 1px #000 solid;
  padding-bottom: 2px;
}
a:focus, a:hover, a:active {
  background: #000;
  color: #fff;
  font-weight: bold;
  text-decoration: none;
  border-bottom: 1px #000 solid;
  padding-bottom: 2px;
}
```

Example 3 shows what this looks like in practice.

To take this example to it’s natural conclusion, you can add an `id` of `current` (or something similar) in appropriate places in your navigation, specify a full set of link styles for current, and have a navigation which, at a glance, lets users know which page or section they’re currently in.

Example navigation using location indicators.

and the source code

Conclusion

All the examples here are intended to illustrate the concepts, and should not be taken as the absolute best way to format links or style navigation bars – that’s up to you and whatever visual design you’re using at the time.

They’re also not the only things you should be doing to make your site accessible.

Above all, remember that accessibility is for life, not just for Christmas.

ABOUT THE AUTHOR

Ann McMeekin is passionate about accessibility and good design, whether on the web or in the real world, and doesn't believe that one has to be sacrificed to achieve the other. This is something she's argued for several years, both in her work (currently as a Web Accessibility Consultant for the RNIB, and previously as a web designer) and to anyone who'll sit still long enough to listen. She blogs for herself at http://www.pixeldiva.co.uk and for work at http://www.rnib.org.uk/wacblog.

14. Underpants Over My Trousers

Andrew Clarke 24ways.org/200714

With Christmas approaching faster than a speeding bullet, this is the perfect time for you to think about that last minute present to buy for the web geek in your life. If you’re stuck for ideas for that special someone, forget about that svelte iPhone case carved from solid mahogany and head instead to your nearest comic-book shop and pick up a selection of comics or graphic novels. (I’ll be using some of my personal favourite comic books as examples throughout).

Trust me, whether your nearest and dearest has been reading comics for a while or has never peered inside this four-colour world, they’ll thank-you for it.

Aside from indulging their superhero fantasies, comic books can provide web designers with a rich vein of inspiring ideas and material to help them create shirt button popping, trouser bursting work for the web. I know from my own personal experience, that looking at aspects of comic book design, layout and conventions and thinking about the ways that they can inform web design has taken my design work in often-unexpected directions.

There are far too many fascinating facets of comic book design that provide web designers with inspiration to cover in the time that it takes to pull your underpants over your trousers. So I'm going to concentrate on one muscle bound aspect of comic design, one that will make you think differently about how you lay out the content of your pages in panels.

A SUITCASE FULL OF KRYPTONITE

Now, to the uninitiated onlooker, the panels of a comic book may appear to perform a similar function to still frames from a movie. But inside the pages of a comic, panels must work harder to help the reader understand the timing of a story. It is this method for conveying narrative timing to a reader that I believe can be highly useful to designers who work on the web as timing, drama and suspense are as important in the web world as they are in worlds occupied by costumed crime fighters and superheroes.

I'd like you to start by closing your eyes and thinking about your own process for laying out panels of content on a page. OK, you'll actually be better off with your eyes open if you're going to carry on reading.

I'll bet you a suitcase full of Kryptonite that you often, if not always, structure your page layouts, and decide on the dimensions of those panels according to either:

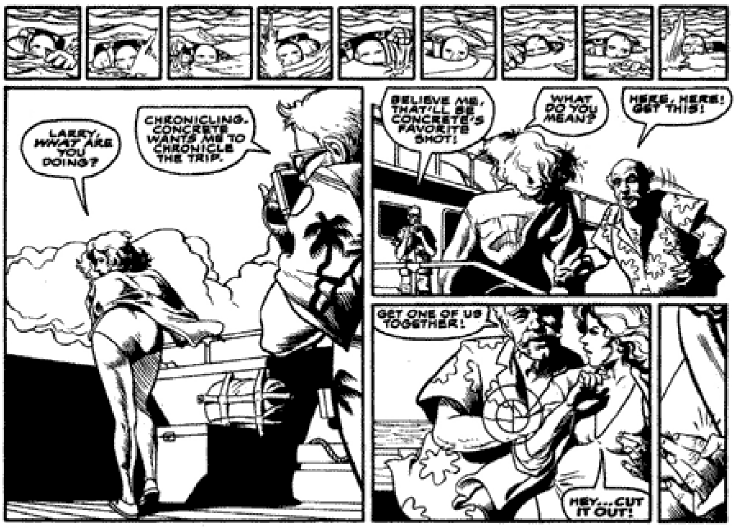
- The base grid that you are working to
- The Golden Ratio or another mathematical schema

More likely, I bet that you decide on the size and the number of your panels based on the amount of content that will be going into them. From today, I'd like you to think about taking a different approach. This approach not only addresses horizontal and vertical space, but also adds the dimension of time to your designs.

SLOWING DOWN THE ACTION

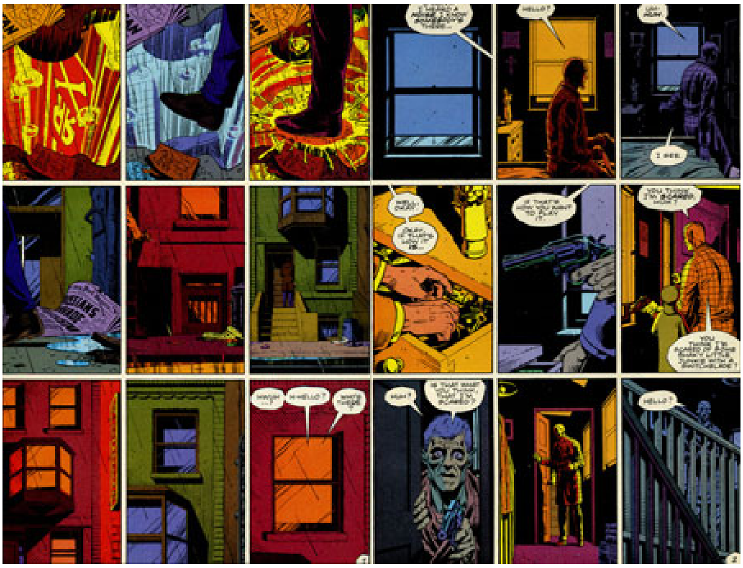
A comic book panel not only acts as a container for its content but also indicates to a reader how much time passes within the panel and as a result, how much time the reader should focus their attention on that one panel.

Smaller panels create swift eye movement and shorter bursts of attention. Larger panels give the perception of more time elapsing in the story and subconsciously demands that a reader devotes more time to focus on it.


LARRY, WHAT ARE YOU DOING?
CHRONICLING. CONCRETE WANTS ME TO CHRONICLE THE TRIP.
BELIEVE ME, THAT'LL BE CONCRETE'S FAVORITE SHOT!
WHAT DO YOU MEAN?
HERE, HERE! GET THIS!
GET ONE OF US TOGETHER!
HEY... CUT IT OUT!
Concrete by Paul Chadwick (Dark Horse Comics)

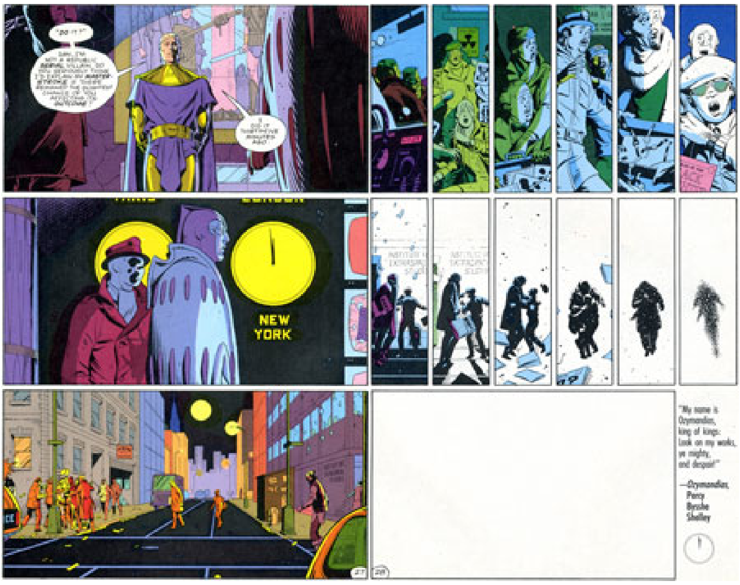
This use of panel dimensions to control timing can also be useful for web designers in designing the reading/user experience. Imagine a page full of information about a product or service. You'll naturally want the reader to focus for longer on the key benefits of your offering rather than perhaps its technical specifications.

Now take a look at this spread of pages from Watchmen by Alan Moore and Dave Gibbons.


Watchmen by Alan Moore and Dave Gibbons (Diamond Comic Distributors 2004)

Throughout this series of (originally) twelve editions, artist Dave Gibbons stuck rigidly to his 3×3 panels per page design and deviated from it only for dramatic moments within the narrative.

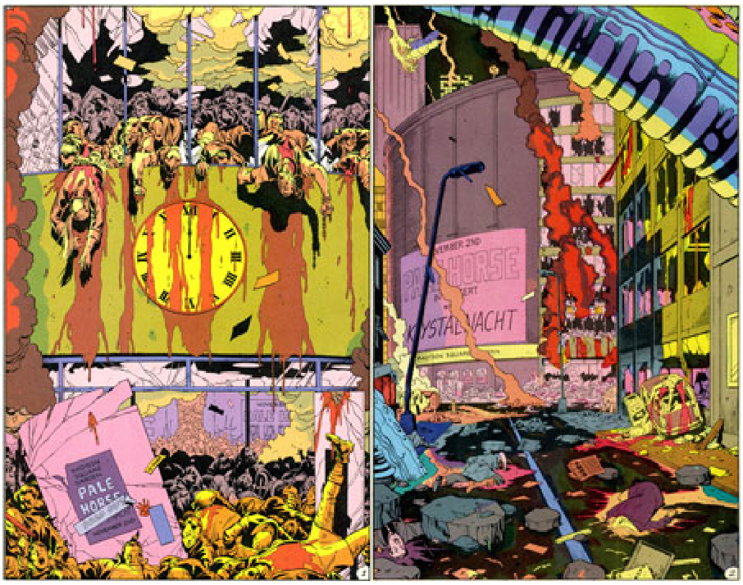
In particular during the last few pages of chapter eleven, Gibbons adds weight to the impending doom by slowing down the action by using larger panels and forces the reader to think longer about what was coming next. The action then speeds up through twelve smaller panels until the final panel: nothing more than white space and yet one of the most iconic and thought provoking in the entire twelve book series.


NEW YORK
"My name is Ozymandias, king of kings: Look on my works, ye mighty, and despair!"
—Ozymandias, Percy Bysshe Shelley

Watchmen by Alan Moore and Dave Gibbons (Diamond Comic Distributors 2004)

On the web it is common for clients to ask designers to fill every pixel of screen space with content, perhaps not understanding the drama that can be added by nothing more than white space.

In the final chapter, Gibbons emphasises the carnage that has taken place (unseen between chapters eleven and twelve) by presenting the reader with six full pages containing only single, large panels.



Watchmen by Alan Moore and Dave Gibbons (Diamond Comic Distributors 2004)

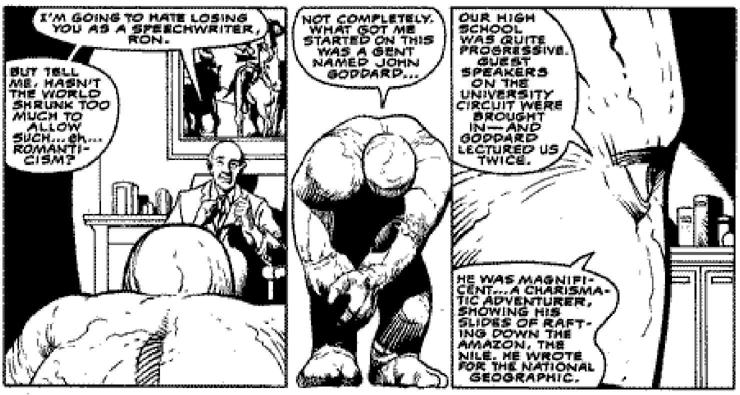
This drama, created by the artist's use of panel dimensions to control timing, is a technique that web designers can also usefully employ when emphasising important areas of content.

Think back for a moment to the home page of Apple Inc., during the launch of their iconic iPhone, where the page contained nothing more than a large image and the phrase "Say hello to iPhone". Rather than fill the page with sales messages, Apple's designers allowed the space itself to tell the story and created a real sense of suspense and expectation among their readers.

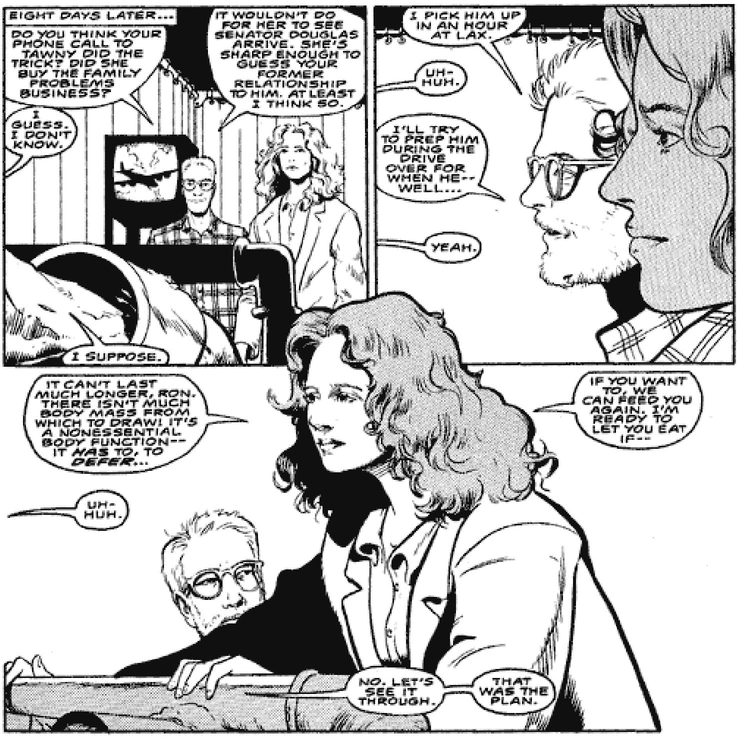
BORDERS

Whereas on the web, panel borders are commonly used to add emphasis to particular areas of content, in comic books they take on a different and sometimes opposite role.

In the examples so far, borders have contained all of the action. Removing a border can have the opposite effect to what you might at first think. Rather than taking emphasis away from their content, in comics, borderless panels allow the reader's eyes to linger for longer on the content adding even stronger emphasis.


I'M GOING TO HATE LOSING YOU AS A SPEECHWRITER, RON.
BUT TELL ME, HASN'T THE WORLD SHRUNK TOO MUCH TO ALLOW SUCH... eh... ROMANTICISM?
NOT COMPLETELY. WHAT GOT ME STARTED ON THIS WAS A GENT NAMED JOHN GODDARD...
OUR HIGH SCHOOL WAS QUITE PROGRESSIVE. GUEST SPEAKERS ON THE UNIVERSITY CIRCUIT WERE BROUGHT IN—AND GODDARD LECTURED US TWICE.
HE WAS MAGNIFICENT... A CHARISMATIC ADVENTURER, SHOWING HIS SLIDES OF RAFTING DOWN THE AMAZON, THE NILE. HE WROTE FOR THE NATIONAL GEOGRAPHIC.
Concrete by Paul Chadwick (Dark Horse Comics)

This effect is amplified when the borderless content is allowed to bleed to the edges of a page. Because the content is no longer confined, except by the edges of the page (both comic and web) the reader's eye is left to wander out into open space.


EIGHT DAYS LATER...
DO YOU THINK YOUR PHONE CALL TO TAWNY DID THE TRICK? DID SHE BUY THE FAMILY PROBLEMS BUSINESS?
I GUESS. I DON'T KNOW.
IT WOULDN'T DO FOR HER TO SEE SENATOR DOUGLAS ARRIVE. SHE'S SHARP ENOUGH TO GUESS YOUR FORMER RELATIONSHIP TO HIM. AT LEAST I THINK SO.
I SUPPOSE.
I PICK HIM UP IN AN HOUR AT LAX.
UH-HUH.
I'LL TRY TO PREP HIM DURING THE DRIVE OVER FOR WHEN HE-- WELL....
YEAH.
IT CAN'T LAST MUCH LONGER, RON. THERE ISN'T MUCH BODY MASS FROM WHICH TO DRAW! IT'S A NONESSENTIAL BODY FUNCTION-- IT *HAS* TO, TO *DEFER*...
UH-HUH.
IF YOU WANT TO, WE CAN FEED YOU AGAIN. I'M READY TO LET YOU EAT IF--
NO. LET'S SEE IT THROUGH.
THAT WAS THE PLAN.

Concrete by Paul Chadwick (Dark Horse Comics)

This type of open, borderless content panel can be highly useful in placing emphasis on the most important content on a page in exactly the very opposite way that we commonly employ on the web today.

So why is time an important dimension to think about when designing your web pages? On one level, we are often already concerned with the short attention spans of visitors to our pages and should work hard to allow them to quickly and easily find and read the content that both they and we think is important. Learning lessons from comic book timing can only help us improve that experience.

On another: timing, suspense and drama are already everyday parts of the web browsing experience. Will a reader see what they expect when they click from one page to the next? Or are they in for a surprise?

Most importantly, I believe that the web, like comics, is about story telling: often the story of the experiences that a customer will have when they use our product or service or interact with our organisation. It is this element of story telling than can be greatly improved by learning from comics.

It is exactly this kind of learning and adapting from older, more established and at first glance unrelated media that you will find can make a real distinctive difference to the design work that you create.

FILL YOUR STOCKINGS

If you're a visual designer or developer and are not a regular reader of comics, from the moment that you pick up your first title, I know that you will find them inspiring.

I will be writing more, and speaking about comic design applied to the web at several (to be announced) events this coming year. I hope you'll be slipping your underpants over your trousers and joining me then. In the meantime, here is some further reading to pick up on your next visit to a comic book or regular bookshop and slip into your stockings:

- **Comics and Sequential Art** by Will Eisner (Northern Light Books 2001)
- **Understanding Comics: The Invisible Art** by Scott McCloud (Harper Collins 1994)

Have a happy superhero season.

(I would like to thank all of the talented artists, writers and publishers whose work I have used as examples in this article and the hundreds more who inspire me every day with their tall tales and talent.)

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.

15. Conditional Love

Ethan Marcotte 24ways.org/200715

“Browser.” The four-letter word of web design.

I mean, let’s face it: on the good days, when things *just work* in your target browsers, it’s marvelous. The air smells sweeter, birds’ songs sound more melodious, and both your design and your code are looking sharp.

But on the less-than-good days (which is, frankly, most of them), you’re compelled to tie up all your browsers in a sack, heave them into the nearest river, and start designing all-imagemap websites. We all play favorites, after all: some will swear by Firefox, Opera fans are allegedly legion, and others still will frown upon anything less than the latest WebKit nightly.

Thankfully, we do have an out for those little inconsistencies that crop up when dealing with cross-browser testing: CSS patches.

SPARE THE ROD, HACK THE BROWSER

Before committing browsercide over some rendering bug, a designer will typically reach for a snippet of CSS fix the faulty browser. Historically referred to as "hacks," I prefer Dan Cederholm's more client-friendly alternative, "patches".

But whatever you call them, CSS patches all work along the same principle: supply the proper property value to the good browsers, while giving ~~higher maintenance~~ other browsers an incorrect value that their ~~frustrating~~ idiosyncratic rendering engine can understand.

Traditionally, this has been done either by exploiting incomplete CSS support:

```
#content {
  height: 1%;   // Let's force hasLayout for old
versions of IE.
  line-height: 1.6;
  padding: 1em;
}
html>body #content {
  height: auto; // Modern browsers get a proper height
value.
}
```

or by exploiting bugs in their rendering engine to deliver alternate style rules:

```
#content p {
  font-size: .8em;
  /* Hide from Mac IE5 \*/
  font-size: .9em;
  /* End hiding from Mac IE5 */
}
```

We've even used these exploits to serve up whole stylesheets altogether:

```
@import url("core.css");
@media tty {
  i{content:"\";/*" "*/}} @import 'windows-ie5.css';
/*";}
}/* */
```

The list goes on, and on, and on. For every browser, for every bug, there's a patch available to fix some rendering bug.

But after some time working with standards-based layouts, I've found that CSS patches, as we've traditionally used them, become increasingly difficult to maintain. As stylesheets are modified over the course of a site's lifetime, inline fixes we've written may become obsolete, making them difficult to find, update, or prune out of our CSS. A good patch requires a constant gardener to ensure that it adds more than just bloat to a stylesheet, and inline patches can be very hard to weed out of a decently sized CSS file.

GIVING THE KIDS SEPARATE ROOMS

Since I joined Airbag Industries earlier this year, every project we’ve worked on has this in the head of its templates:

```
<link rel="stylesheet" href="-/css/screen/main.css"
type="text/css" media="screen, projection" />
<!--[if lt IE 7]>
<link rel="stylesheet" href="-/css/screen/patches/
win-ie-old.css" type="text/css" media="screen,
projection" />
<![endif]-->
<!--[if gte IE 7]>
<link rel="stylesheet" href="-/css/screen/patches/
win-ie7-up.css" type="text/css" media="screen,
projection" />
<![endif]-->
```

The first element is, simply enough, a `link` element that points to the project’s main CSS file. No patches, no hacks: just pure, modern browser-friendly style rules. Which, nine times out of ten, will net you a design that looks like spilled eggnog in various versions of Internet Explorer.

But don’t reach for the mulled wine quite yet. Immediately after, we’ve got a brace of conditional comments wrapped around two other `link` elements. These odd-looking comments allow us to selectively serve up additional stylesheets *just* to the version of IE that needs them. We’ve got one for IE 6 and below:

```
<!--[if lt IE 7]>
<link rel="stylesheet" href="-/css/screen/patches/
win-ie-old.css" type="text/css" media="screen,
projection" />
<![endif]-->
```

And another for IE7 and above:

```
<!--[if gte IE 7]>
<link rel="stylesheet" href="-/css/screen/patches/
win-ie7-up.css" type="text/css" media="screen,
projection" />
<![endif]-->
```

Microsoft's conditional comments aren't exactly new, but they can be a valuable alternative to cooking CSS patches directly into a master stylesheet. And though they're not a W3C-approved markup structure, I think they're just brilliant because they innovate within the spec: non-IE devices will assume that the comments are just that, and ignore the markup altogether.

This does, of course, mean that there's a little extra markup in the `head` of our documents. But this approach can seriously cut down on the unnecessary patches served up to the browsers that don't need them. Namely, we no longer have to write rules like this in our main stylesheet:

```
#content {
  height: 1%;  // Let's force hasLayout for old versions
of IE.
```

```
  line-height: 1.6;
  padding: 1em;
}
html>body #content {
  height: auto;  // Modern browsers get a proper height
value.
}
```

Rather, we can simply write an un-patched rule in our core stylesheet:

```
#content {
  line-height: 1.6;
  padding: 1em;
}
```

And now, our patch for older versions of IE goes in—you guessed it—the stylesheet for older versions of IE:

```
#content {
  height: 1%;
}
```

The `hasLayout` patch is applied, our design's repaired, and—most importantly—the patch is only seen by the browser that needs it. The "good" browsers don't have to incur any added stylesheet weight from our IE patches, and Internet Explorer gets the conditional love it deserves.

Most importantly, this "compartmentalized" approach to CSS patching makes it much easier for me to patch and maintain the fixes applied to a particular browser. If I need to track down a bug for IE7, I don't need to scroll through dozens or hundreds of rules in my core stylesheet: instead, I just open the considerably slimmer IE7-specific patch file, make my edits, and move right along.

EVEN GOOD CHILDREN MISBEHAVE

While IE may occupy the bulk of our debugging time, there's no denying that other popular, modern browsers will occasionally disagree on how certain bits of CSS should be rendered. But without something as, well, pimp as conditional comments at our disposal, how do we bring the so-called "good browsers" back in line with our design?

Assuming you're loving the "one patch file per browser" model as much as I do, there's just one alternative: JavaScript.

```
function isSaf() {
  var isSaf = (document.childNodes && !document.all &&
!navigator.taintEnabled && !navigator.accentColorName) ?
true : false;
  return isSaf;
}
function isOp() {
  var isOp = (window.opera) ? true : false;
  return isOp;
}
```

Instead of relying on dotcom-era tactics of parsing the browser's user-agent string, we're testing here for support for various DOM objects, whose presence or absence we can use to reasonably infer the browser we're looking at. So running the `isOp()` function, for example, will test for Opera's proprietary `window.opera` object, and thereby accurately tell you if your user's running Norway's finest browser.

With scripts such as `isOp()` and `isSaf()` in place, you can then reasonably test which browser's viewing your content, and insert additional `link` elements as needed.

```
function loadPatches(dir) {
  if (document.getElementsByTagName() &&
document.createElement()) {
    var head = document.getElementsByTagName("head")[0];
    if (head) {
      var css = new Array();
      if (isSaf()) {
        css.push("saf.css");
      } else if (isOp()) {
        css.push("opera.css");
      }
      if (css.length) {
        var link = document.createElement("link");
        link.setAttribute("rel", "stylesheet");
        link.setAttribute("type", "text/css");
        link.setAttribute("media", "screen, projection");
        for (var i = 0; i < css.length; i++) {
          var tag = link.cloneNode(true);
          tag.setAttribute("href", dir + css[0]);
          head.appendChild(tag);
        }
      }
    }
  }
}
```

Here, we're testing the results of `isSaf()` and `isOp()`, one after the other. For each function that returns `true`, then the name of a new stylesheet is added to the oh-so-cleverly named `css` array. Then, for each entry in `css`, we create a new `link` element, point it at our patch file, and insert it into the `head` of our template.

Fire it up using your favorite onload or DOMContentLoaded function, and you're good to go.

Scripteat Emptor

At this point, some of the audience's more conscientious 'scripters may be preparing to lob figgy pudding at this author's head. And that's perfectly understandable; relying on JavaScript to patch CSS chafes a bit against the normally clean separation we have between our pages' content, presentation, and behavior layers.

And beyond the philosophical concerns, this approach comes with a few technical caveats attached:

BROWSER DETECTION? SO UN-133T.

Browser detection is *not* something I'd typically recommend. Whenever possible, a proper DOM script should check for the support of a given object or method, rather than the device with which your users view your content.

IT'S JAVASCRIPT, SO DON'T COUNT ON IT BEING AVAILABLE.

According to one site, roughly four percent of Internet users don't have JavaScript enabled. Your site's stats might be higher or lower than this number, but still: don't expect that every member of your audience will see these additional stylesheets, and ensure that your content's still accessible with JS turned off.

BE A CONSTANT GARDENER.

The sample `isSaf()` and `isOp()` functions I've written will tell you if the user's browser is Safari or Opera. As a result, stylesheets written to patch issues in an old browser may break when later releases repair the relevant CSS bugs.

You can, of course, add logic to these simple little scripts to serve up version-specific stylesheets, but that way madness may lie. In any event, test your work vigorously, and keep testing it when new versions of the targeted browsers come out. Make sure that a patch written today doesn't become a bug tomorrow.

Patching Firefox, Opera, and Safari isn't something I've had to do frequently: still, there have been occasions where the above script's come in handy. Between conditional comments, careful CSS auditing, and some judicious JavaScript, browser-based bugs can be handled with near-surgical precision.

So pass the 'nog. It's patchin' time.

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.

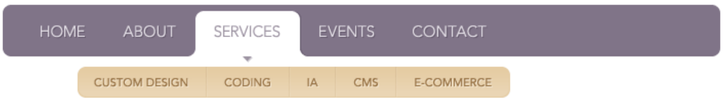
He swears profusely on Twitter, and would like to be an unstoppable robot ninja when he grows up. Beep.

Photo: Brian Warren

16. Get In Shape

Dave Shea 24ways.org/200716

Pop quiz: what's wrong with the following navigation?



Maybe nothing. But then again, maybe there's something bugging you about the way it comes together, something you can't quite put your finger on. It seems well-designed, but it also seems a little… off.

The design decisions that led to this eventual form were no doubt well-considered:

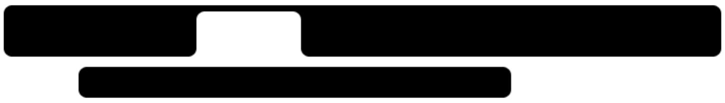
- *Client*: The top level needs to have a "current page" status indicator of some sort.
- *Designer*: How about a white tab?
- *Client*: Great! The second level needs to show up underneath the first level though…
- *Designer*: Okay, but that white tab I just added makes it hard to visually connect the bottom nav to the top.
- *Client*: Too late, we've seen the white tab and we love it. Try and make it work.

- *Designer*: Right. So I placed the second level in its own box.
- *Client*: Hmm. They seem too separated. I can't tell that the yellow nav is the second level of the first.
- *Designer*: How about an indicator arrow?
- *Client*: Brilliant!

The problem is that the end result feels awkward and forced. During the design process, little decisions were made that ultimately affect the overall shape of the navigation. What started out as a neatly contained rounded rectangle ended up as an ambiguous double shape that looks funny, though it's often hard to pinpoint precisely *why*.

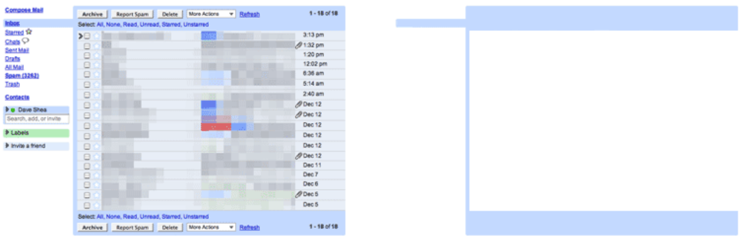
THE SHAPE OF THINGS

Well the why in this case is because seemingly unrelated elements in a design still end up visually interacting. Adding a new item to a page impacts everything surrounding it. In this navigation example, we're looking at two individual objects that are close enough to each other that they form a relationship; if we reduce them to strictly their outlines, it's a little easier to see that this particular combination registers oddly.

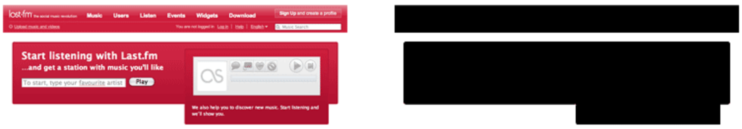


The two shapes float with nothing really grounding them. If they were connected, perhaps it would be a different story. The white tab divides the top shape in half, leaving a gap in the middle of it. There's very little balance in this pairing because the overall shape of the navigation wasn't considered during the design process.

Here's another example: Gmail. Great email client, but did you ever closely look at what's going on in that left hand navigation? The continuous blue bar around the message area spills out into the navigation. If we remove all text, we're left with this odd configuration:



Though the reasoning for anchoring the navigation highlight against the message area might be sound, the result is an irregular shape that doesn't correspond with anything in reality. You may never consciously notice it, but once you do it's hard to miss. One other example courtesy of last.fm:


last.fm
Music
Users
Listen
Events
Widgets
Download
Sign Up and create a profile
Start listening with Last.fm
...and get a station with music you'll like
To start, type your favourite artist
Play
We also help you to discover new music. Start listening and we'll show you.

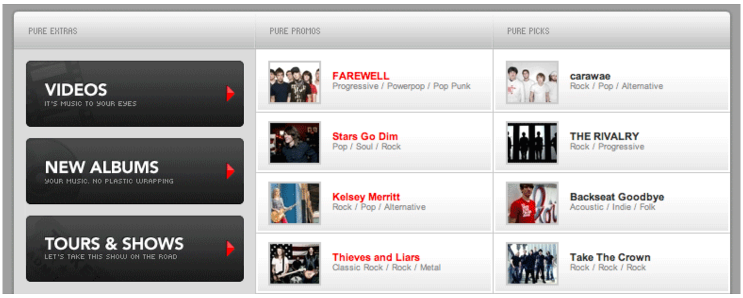
The two header areas are the same shade of pink so they appear to be closely connected. When reduced to their outlines it’s easy to see that this combination is off-balance: the edges don’t align, the sharp corners of the top shape aren’t consistent with the rounded corners of the bottom, and the part jutting out on the right of the bottom one seems fairly random. The result is a duo of oddly mis-matched shapes.

DESIGN STRATEGIES

Our minds tend to pick out familiar patterns. A clever designer can exploit this by creating references in his or her work to shapes and combinations with which viewers are already familiar. There are a few simple ideas that can be employed to help you achieve this: consistency, balance, and completion.

Consistency

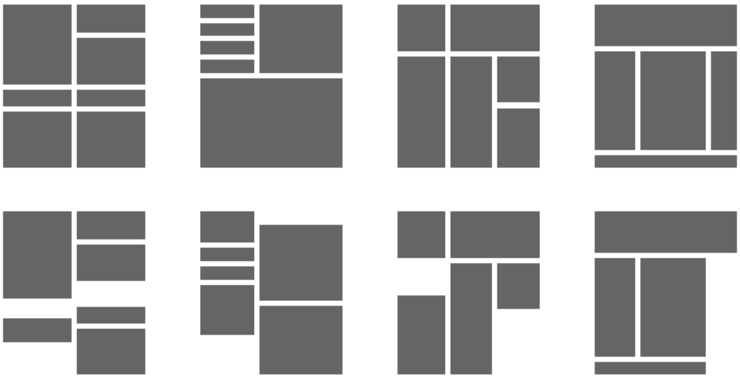
A fairly simple way to unify the various disparate shapes on a page is by designing them with a certain amount of internal consistency. You don’t need to apply an identical size, colour, border, or corner treatment to every single shape; devolving a design into boring repetition isn’t what we’re after here. But it certainly doesn’t hurt to apply a set of common rules to most shapes within your work.



Consider purevolume and its multiple rounded-corner panels. From the bottom of the site's main navigation to the grey "Extras" panels halfway down the page (shown above), multiple shapes use a common border radius for unity. Different colours, different sizes, different content, but the consistent outlines create a strong sense of similarity. Not that every shape on the site follows this rule; they break the pattern right at the top with a darker sharp-cornered header, and again with the thumbnails below. But the design remains unified, nonetheless.

Balance

Arguably the biggest problem with the last.fm example earlier is one of balance. The area poking out of the bottom shape created a fairly obvious imbalance for no apparent reason. The right hand side is visually emphasized due to the greater area of pink coverage, but with the white gap left beside it, the emphasis seems unwarranted. It's possible to create tension in your design by mismatching shapes and throwing off the balance, but when that happens unintentionally it can look like a mistake.

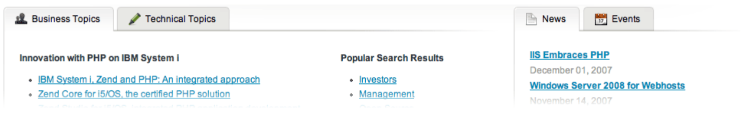


Above are a few examples of design elements in balanced and unbalanced configurations. The examples in the top row are undeniably more pleasing to the eye than those in the bottom row. If these were fleshed out into full designs, those derived from the templates in the top row would naturally result in stronger work.

Take a look at the header on 9Rules for a study in well-considered balance. On the left you'll see a couple of paragraphs of text, on the right you have floating navigational items, and both flank the site's logo. This unusual layout combines multiple design elements that look nothing alike, and places them together in a way that anchors each so that no one weighs down the header.

Completion

And finally we come to the idea of completion. Shapes don't necessarily need hard outlines to be read visually as shapes, which can be exploited for various purposes. Notice how Zend's mid-page "Business Topics" and "News" items (below) fade out to the right and bottom, but the placement of two of these side-by-side creates an impression of two panels rather than three disparate floating columns. By allowing the viewer's eye to complete the shapes, they've lightened up the design of the page and removed inessential lines. In a busy design this technique could prove quite handy.


Business Topics | Technical Topics

Innovation with PHP on IBM System i

- IBM System i, Zend and PHP: An integrated approach
- Zend Core for i5/OS, the certified PHP solution

Popular Search Results

- Investors
- Management

News | Events

IIS Embraces PHP
December 01, 2007
Windows Server 2008 for Webhosts
November 14, 2007

Along the same lines, the individual shapes within your design may also be combined visually to form outlines of larger shapes. The differently-coloured header and main content/sidebar shapes on Veerle's blog come together to form a single central panel, further emphasized by the slight drop shadow to the right.

IMPLEMENTATION

Studying how shape can be used effectively in design is simply a starting point. As with all things design-related, there are no hard and fast rules here; ultimately you may choose to bring these principles into your work more often, or break them for effect. But understanding how shapes interact within a page, and how that effect is ultimately perceived by viewers, is a key design principle you can use to impress your friends.

ABOUT THE AUTHOR

Dave Shea is the Founder of Bright Creative, and co-organizer of Web Directions North. He blogs sometimes at Mezzoblue and Flickrs a bit more often than that. Oh, and there's other stuff too.

17. Increase Your Font Stacks With Font Matrix

Richard Rutter 24ways.org/200717

Web pages built in plain old HTML and CSS are displayed using only the fonts installed on users' computers (`@font-face` implementations excepted). To enable this, CSS provides the font-family property for specifying fonts in order of preference (often known as a font stack). For example:

```
h1 {font-family: 'Egyptienne F', Cambria, Georgia, serif}
```

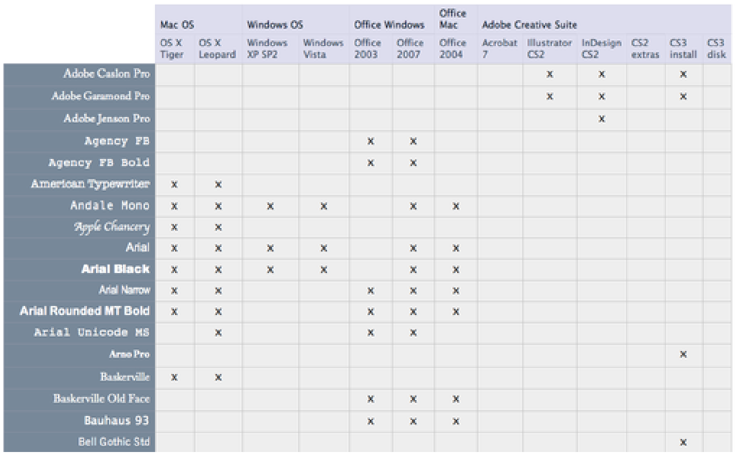
So in the above rule, headings will be displayed in Egyptienne F. If Egyptienne F is not available then Cambria will be used, failing that Georgia or the final fallback default serif font. This everyday bit of CSS will be common knowledge among all 24 ways readers.

It is also a commonly held belief that the only fonts we can rely on being installed on users' computers are the core web fonts of Arial, Times New Roman, Verdana, Georgia and friends. But is that really true?

If you look in the fonts folder of your computer, or even your Mum's computer, then you are likely to find a whole load of fonts besides the core ones. This is because many software packages automatically install extra typefaces. For example, Office 2003 installs over 100 additional fonts. Admittedly not all of these fonts are particularly refined, and not all are suitable for the Web. However they still do increase your options.

THE MATRIX

I have put together a matrix of (western) fonts showing which are installed with Mac and Windows operating systems, which are installed with various versions of Microsoft Office, and which are installed with Adobe Creative Suite.



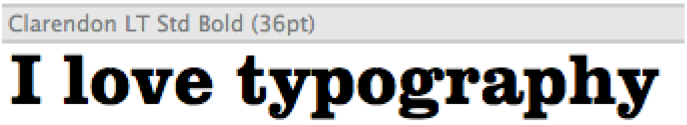
	Mac OS		Windows OS		Office Windows		Office Mac	Adobe Creative Suite					
	OS X Tiger	OS X Leopard	Windows XP SP2	Windows Vista	Office 2003	Office 2007	Office 2004	Acrobat 7	Illustrator CS2	InDesign CS2	CS2 extras	CS3 install	CS3 disk
Adobe Caslon Pro									x	x		x	
Adobe Garamond Pro									x	x		x	
Adobe Jenson Pro										x			
Agency FB					x	x							
Agency FB Bold					x	x							
American Typewriter	x	x											
Andale Mono	x	x	x	x		x	x						
Apple Chancery	x	x											
Arial	x	x	x	x		x	x						
Arial Black	x	x	x	x		x	x						
Arial Narrow	x	x			x	x	x						
Arial Rounded MT Bold	x	x			x	x	x						
Arial Unicode MS		x			x	x							
Arno Pro												x	
Baskerville	x	x											
Baskerville Old Face					x	x	x						
Bauhaus 93					x	x	x						
Bell Gothic Std												x	

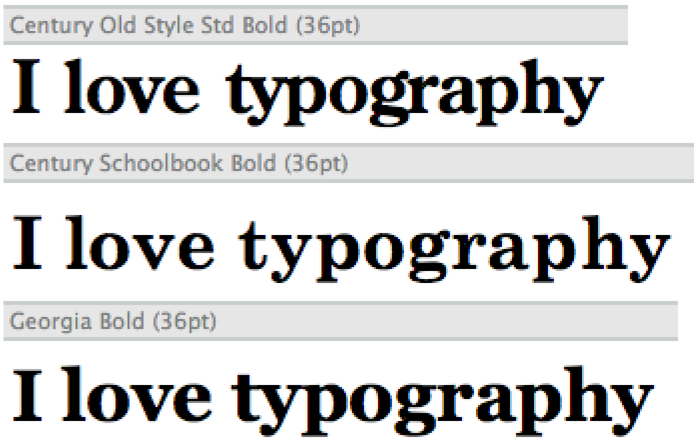
The matrix is available for download as an Excel file and as a CSV. There are no readily available statistics regarding the penetration of Office or Creative Suite, but you can probably take an educated guess based on your knowledge of your readers.

The idea of the matrix is that use can use it to help construct your font stack. First of all pick the font you'd really like for your text – this doesn't have to be in the matrix. Then pick the generic family (serif, sans-serif, cursive, fantasy or monospace) and a font from each of the operating systems. Then pick any suitable fonts from the Office and Creative Suite lists.

For example, you may decide your headings should be in the increasingly ubiquitous Clarendon. This is a serif type face. At OS-level the most similar is arguably Georgia. Adobe CS2 comes with Century Old Style which has a similar feel. Century Schoolbook is similar too, and is installed with all versions of Office. Based on this your font stack becomes:

```
font-family: 'Clarendon Std', 'Century Old Style
Std', 'Century Schoolbook', Georgia, serif
```





Note the ‘Std’ suffix indicating a ‘standard’ OpenType file, which will normally be your best bet for more esoteric fonts.

I’m not suggesting the process of choosing suitable fonts is an easy one. Firstly there are nearly two hundred fonts in the matrix, so learning what each font looks like is tricky and potentially time consuming (if you haven’t got all the fonts installed on a machine to hand you’ll be doing a lot of Googling for previews). And it’s not just as simple as choosing fonts that look similar or have related typographic backgrounds, they need to have similar metrics as well, This is especially true in terms of x-height which gives an indication of how big or small a font looks.

OVER TO YOU

The main point of all this is that there are potentially more fonts to consider than is generally accepted, so branch out a little (carefully and tastefully) and bring a little variety to sites out there. If you come up with any novel font stacks based on this approach, please do blog them (tagged as per the footer) and at some point they could all be combined in one place for everyone to consider.

APPENDIX

What about Linux?

The only operating systems in the matrix are those from Microsoft and Apple. For completeness, Linux operating systems should be included too, although these are many and varied and very much in a minority, so I omitted them for time being. For the record, some Linux distributions come packaged with Microsoft's core fonts. Others use the Vera family, and others use the Liberation family which comprises fonts metrically identical to Times New Roman and Arial.

Sources

The sources of font information for the matrix are as follows:

- Windows XP SP2

- Windows Vista
- Office 2003
- Office 2007
- Mac OSX Tiger
- Mac OSX Leopard (scroll down two thirds)
- Office 2004 (Mac) by inspecting my Microsoft Office 2004/Office/Fonts folder
- Office 2008 (Mac) is expected to be as Office 2004 with the addition of the Vista ClearType fonts
- Creative Suite 2 (see pdf link in first comment)
- Creative Suite 3

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.

18. Keeping JavaScript Dependencies At Bay

Christian Heilmann 24ways.org/200718

As we are writing more and more complex JavaScript applications we run into issues that have hitherto (god I love that word) not been an issue. The first decision we have to make is what to do when planning our app: one big massive JS file or a lot of smaller, specialised files separated by task.

Personally, I tend to favour the latter, mainly because it allows you to work on components in parallel with other developers without lots of clashes in your version control. It also means that your application will be more lightweight as you only include components on demand.

STARTING WITH A GLOBAL OBJECT

This is why it is a good plan to start your app with one single object that also becomes the namespace for the whole application, say for example `myAwesomeApp`:

```
var myAwesomeApp = {};
```

You can nest any necessary components into this one and also make sure that you check for dependencies like DOM support right up front.

ADDING THE COMPONENTS

The other thing to add to this main object is a components object, which defines all the components that are there and their file names.

```
var myAwesomeApp = {
  components :{
    formcheck:{
      url:'formcheck.js',
      loaded:false
    },
    dynamicnav:{
      url:'dynamicnav.js',
      loaded:false
    },
    gallery:{
      url:'gallery.js',
      loaded:false
    },
    lightbox:{
      url:'lightbox.js',
      loaded:false
    }
  }
};
```

Technically you can also omit the loaded properties, but it is cleaner this way. The next thing to add is an `addComponent` function that can load your components on demand by adding new `SCRIPT` elements to the head of the documents when they are needed.

```
var myAwesomeApp = {
  components :{
    formcheck:{
      url:'formcheck.js',
      loaded:false
    },
    dynamicnav:{
      url:'dynamicnav.js',
      loaded:false
    },
    gallery:{
      url:'gallery.js',
      loaded:false
    },
    lightbox:{
      url:'lightbox.js',
      loaded:false
    }
  },
  addComponent:function(component){
    var c = this.components[component];
    if(c && c.loaded === false){
      var s = document.createElement('script');
      s.setAttribute('type', 'text/javascript');
      s.setAttribute('src',c.url);

document.getElementsByTagName('head')[0].appendChild(s);
```

```
    }
  }
};
```

This allows you to add new components on the fly when they are not defined:

```
if(!myAwesomeApp.components.gallery.loaded){
  myAwesomeApp.addComponent('gallery');
};
```

VERIFYING THAT COMPONENTS HAVE BEEN LOADED

However, this is not safe as the file might not be available. To make the dynamic adding of components safer each of the components should have a callback at the end of them that notifies the main object that they indeed have been loaded:

```
var myAwesomeApp = {
  components :{
    formcheck:{
      url:'formcheck.js',
      loaded:false
    },
    dynamicnav:{
      url:'dynamicnav.js',
      loaded:false
    },
    gallery:{
      url:'gallery.js',
      loaded:false
    },
    lightbox:{
      url:'lightbox.js',
      loaded:false
    }
  },
  addComponent:function(component){
    var c = this.components[component];
    if(c && c.loaded === false){
      var s = document.createElement('script');
      s.setAttribute('type', 'text/javascript');
      s.setAttribute('src',c.url);

document.getElementsByTagName('head')[0].appendChild(s);
    }
  },
  componentAvailable:function(component){
    this.components[component].loaded = true;
  }
}
```

For example the `gallery.js` file should call this notification as a last line:

```
myAwesomeApp.gallery = function(){
  // [... other code ...]
}();
myAwesomeApp.componentAvailable('gallery');
```

TELLING THE IMPLEMENTERS WHEN COMPONENTS ARE AVAILABLE

The last thing to add (actually as a courtesy measure for debugging and implementers) is to offer a listener function that gets notified when the component has been loaded:

```
var myAwesomeApp = {
  components :{
    formcheck:{
      url:'formcheck.js',
      loaded:false
    },
    dynamicnav:{
      url:'dynamicnav.js',
      loaded:false
    },
    gallery:{
      url:'gallery.js',
      loaded:false
    },
    lightbox:{
      url:'lightbox.js',
      loaded:false
    }
  },
  addComponent:function(component){
    var c = this.components[component];
    if(c && c.loaded === false){
      var s = document.createElement('script');
      s.setAttribute('type', 'text/javascript');
      s.setAttribute('src',c.url);
document.getElementsByTagName('head')[0].appendChild(s);
    }
  },
  componentAvailable:function(component){
    this.components[component].loaded = true;
    if(this.listener){
      this.listener(component);
    };
  }
};
```

This allows you to write a main listener function that acts when certain components have been loaded, for example:

```
myAwesomeApp.listener = function(component){
  if(component === 'gallery'){
     showGallery();
  }
};
```

EXTENDING WITH OTHER COMPONENTS

As the main object is public, other developers can extend the components object with own components and use the listener function to load dependent components. Say you have a bespoke component with data and labels in extra files:

```
myAwesomeApp.listener = function(component){
  if(component === 'bespokecomponent'){
    myAwesomeApp.addComponent('bespokelabels');
  };
  if(component === 'bespokelabels'){
    myAwesomeApp.addComponent('bespokedata');
  };
  if(component === 'bespokedata'){
    myAwesomeApp,bespokecomponent.init();
  };
};
myAwesomeApp.components.bespokecomponent = {
  url:'bespoke.js',
  loaded:false
};
myAwesomeApp.components.bespokelabels = {
  url:'bespokelabels.js',
  loaded:false
};
myAwesomeApp.components.bespokedata = {
  url:'bespokedata.js',
  loaded:false
};
myAwesomeApp.addComponent('bespokecomponent');
```

Following this practice you can write pretty complex apps and still have full control over what is available when. You can also extend this to allow for CSS files to be added on demand.

INFLUENCES

If you like this idea and wondered if someone already uses it, take a look at the Yahoo! User Interface library, and especially at the YAHOO_config option of the global YAHOO.js object.

AVAILABLE IN GERMAN
webkrauts.de

ABOUT THE AUTHOR

Christian Heilmann grew up in Germany and, after a year working for the red cross, spent a year as a radio producer. From 1997 onwards he worked for several agencies in Munich as a web developer. In 2000 he moved to the States to work for Etoys and, after the .com crash, he moved to the UK where he lead the web development department at Agilisys. In April 2006 he joined Yahoo! UK as a web developer and moved on to be the Lead Developer Evangelist for the Yahoo Developer Network. In December 2010 he moved on to Mozilla as Principal Developer Evangelist for HTML5 and the Open Web. He publishes an almost daily blog at http://wait-till-i.com and runs an article repository at http://icant.co.uk. He also authored Beginning JavaScript with DOM Scripting and Ajax: From Novice to Professional.

19. Christmas Is In The AIR

Jonathan Snook 24ways.org/200719

That's right, Christmas is coming up fast and there's plenty of things to do. Get the tree and lights up, get the turkey, buy presents and who know what else. And what about Santa? He's got a list. I'm pretty sure he's checking it twice.

Sure, we could use an existing list making web site or even a desktop widget. But we're geeks! What's the fun in that? Let's build our own to-do list application and do it with Adobe AIR!

WHAT'S ADOBE AIR?

Adobe AIR, formerly codenamed Apollo, is a runtime environment that runs on both Windows and OSX (with Linux support to follow). This runtime environment lets you build desktop applications using Adobe technologies like Flash and Flex. Oh, and HTML. That's right, you web standards lovin' maniac. You can build desktop applications that can run cross-platform using the trio of technologies, HTML, CSS and JavaScript.

If you've tried developing with AIR before, you'll need to get re-familiarized with the latest beta release as many things have changed since the last one (such as the API and restrictions within the sandbox.)

TO GET STARTED

To get started in building an AIR application, you'll need two basic things:

1. The AIR runtime. The runtime is needed to run any AIR-based application.
2. The SDK. The software development kit gives you all the pieces to test your application. Unzip the SDK into any folder you wish.

You'll also want to get your hands on the JavaScript API documentation which you'll no doubt find yourself getting into before too long. (You can download it, too.)

Also of interest, some development environments have support for AIR built right in. Aptana doesn't have support for beta 3 yet but I suspect it'll be available shortly.

Within the SDK, there are two main tools that we'll use: one to test the application (ADL) and another to build a distributable package of our application (ADT). I'll get into this some more when we get to that stage of development.

BUILDING OUR TO-DO LIST APPLICATION

The first step to building an application within AIR is to create an XML file that defines our default application settings. I call mine `application.xml`, mostly because Aptana does that by default when creating a new AIR project. It makes sense though and I've stuck with it. Included in the templates folder of the SDK is an example XML file that you can use.

The first key part to this after specifying things like the application ID, version, and filename, is to specify what the default content should be within the content tags. Enter in the name of the HTML file you wish to load. Within this HTML file will be our application.

```
<content>ui.html</content>
```

Create a new HTML document and name it `ui.html` and place it in the same directory as the `application.xml` file. The first thing you'll want to do is copy over the `AIRAliases.js` file from the frameworks folder of the SDK and add a link to it within your HTML document.

```
<script type="text/javascript"
src="AIRAliases.js"></script>
```

The aliases create shorthand links to all of the Flash-based APIs.

Now is probably a good time to explain how to debug your application.

Debugging our application

So, with our XML file created and HTML file started, let's try testing our 'application'. We'll need the ADL application located in BIN folder of the SDK and tell it to run the `application.xml` file.

```
/path/to/adl /path/to/application.xml
```

You can also just drag the XML file onto ADL and it'll accomplish the same thing. If you just did that and noticed that your blank application didn't load, you'd be correct. It's running but isn't visible. Which at this point means you'll have to shut down the ADL process. Sorry about that!

Changing the visibility

You have two ways to make your application visible. You can do it automatically by setting the placing `true` in the `visible` tag within the `application.xml` file.

```
<visible>true</visible>
```

The other way is to do it programmatically from within your application. You'd want to do it this way if you had other startup tasks to perform before showing the interface. To turn the UI on programmatically, simple set the `visible` property of `nativeWindow` to `true`.

```
<script type="text/javascript">
  nativeWindow.visible = true;
</script>
```

Sandbox Security

Now that we have an application that we can see when we start it, it's time to build the to-do list application. In doing so, you'd probably think that using a JavaScript library is a really good idea — and it can be but there are some limitations within AIR that have to be considered.

An HTML document, by default, runs within the application sandbox. You have full access to the AIR APIs but once the `onload` event of the window has fired, you'll have a limited ability to make use of `eval` and other dynamic script injection approaches. This limits the ability of external sources from gaining access to everything the AIR API offers, such as database and local file system access. You'll still be able to make use of `eval` for evaluating JSON responses, which is probably the most important if you wish to consume JSON-based services.

If you wish to create a greater wall of security between AIR and your HTML document loading in external resources, you can create a child sandbox. We won't need to worry about it for our application so I won't go any further into it but definitely keep this in mind.

Finally, our application

Getting tired of all this preamble? Let's actually build our to-do list application. I'll use jQuery because it's small and should suit our needs nicely. Let's begin with some structure:

```
<body>
  <input type="text" id="text" value="">
  <input type="button" id="add" value="Add">
  <ul id="list"></ul>
</body>
```

Now we need to wire up that button to actually add a new item to our to-do list.

```
<script type="text/javascript">
$(document).ready(function(){
  // make sure the application is visible
  nativeWindow.visible = true;
  $('#add').click(function(){
    var t = $('#text').val();
    if(t)
    {
      // use DOM methods to create the new list item
      var li = document.createElement('li');
      // the extra space at the end creates a buffer
between the text
      // and the delete link we're about to add
      li.appendChild(document.createTextNode(t + ' '));
      // create the delete link
      var del = document.createElement('a');
      // this makes it a true link. I feel dirty doing
this.
      del.setAttribute('href', '#');
      del.addEventListener('click', function(evt){

this.parentNode.parentNode.removeChild(this.parentNode);
      });
      del.appendChild(document.createTextNode('[del]'));
      li.appendChild(del);
      // append everything to the list
      $('#list').append(li);
      //reset the text box
      $('#text').val('');
    }
  })
});
</script>
```

And just like that, we've got a to-do list! That's it! Just never close your application and you'll remember everything. Okay, that's not very practical. You need to have some way of storing your to-do items until the next time you open up the application.

STORING DATA

You've essentially got 4 different ways that you can store data:

- Using the local database. AIR comes with SQLLite built in. That means you can create tables and insert, update and select data from that database just like on a web server.
- Using the file system. You can also create files on the local machine. You have access to a few folders on the local system such as the documents folder and the desktop.
- Using `EcryptedLocalStore`. I like using the `EcryptedLocalStore` because it allows you to easily save key/value pairs and have that information encrypted. All this within just a couple lines of code.
- Sending the data to a remote API. Our to-do list could sync up with Remember the Milk, for example.

To demonstrate some persistence, we'll use the file system to store our files. In addition, we'll let the user specify where the file should be saved. This way, we can create multiple to-do lists, keeping them separate and organized.

The application is now broken down into 4 basic tasks:

1. Load data from the file system.
2. Perform any interface bindings.

3. Manage creating and deleting items from the list.
4. Save any changes to the list back to the file system.

Loading in data from the file system

When the application starts up, we'll prompt the user to select a file or specify a new to-do list. Within AIR, there are 3 main file objects: `File`, `FileMode`, and `FileStream`. `File` handles file and path names, `FileMode` is used as a parameter for the `FileStream` to specify whether the file should be read-only or for write access. The `FileStream` object handles all the read/write activity.

The `File` object has a number of shortcuts to default paths like the documents folder, the desktop, or even the application store. In this case, we'll specify the documents folder as the default location and then use the `browseForSave` method to prompt the user to specify a new or existing file. If the user specifies an existing file, they'll be asked whether they want to overwrite it.

```
var store = air.File.documentsDirectory;
var fileStream = new air.FileStream();
store.browseForSave(”Choose To-do List”);
```

Then we add an event listener for when the user has selected a file. When the file is selected, we check to see if the file exists and if it does, read in the contents, splitting the file on new lines and creating our list items within the interface.

```
store.addEventListener(air.Event.SELECT, fileSelected);
function fileSelected()
{
  air.trace(store.nativePath);
  // load in any stored data
  var byteData = new air.ByteArray();
  if(store.exists)
  {
    fileStream.open(store, air.FileMode.READ);
    fileStream.readBytes(byteData, 0, store.size);
    fileStream.close();
```

if(byteData.length > 0) { var s = byteData.readUTFBytes(byteData.length); oldlist = s.split("\r\n");

// create todolist items for(var i=0; i < oldlist.length; i++) { createItem(oldlist[i], (new Date()).getTime() + i); } } }
}

Perform Interface Bindings

This is similar to before where we set the click event on the Add button but we've moved the code to save the list into a separate function.

```
$('#add').click(function(){
    var t = $('#text').val();
    if(t){
      // create an ID using the time
      createItem(t, (new Date()).getTime() );
    }
})
```

Manage creating and deleting items from the list

The list management is now in its own function, similar to before but with some extra information to identify list items and with calls to save our list after each change.

```
function createItem(t, id)
{
  if(t.length == 0) return;
  // add it to the todo list
  todolist[id] = t;
  // use DOM methods to create the new list item
  var li = document.createElement('li');
  // the extra space at the end creates a buffer between
the text
  // and the delete link we're about to add
  li.appendChild(document.createTextNode(t + ' '));
  // create the delete link
  var del = document.createElement('a');
  // this makes it a true link. I feel dirty doing this.
  del.setAttribute('href', '#');
  del.addEventListener('click', function(evt){
    var id = this.id.substr(1);
    delete todolist[id]; // remove the item from the list

this.parentNode.parentNode.removeChild(this.parentNode);
    saveList();
  });
  del.appendChild(document.createTextNode('[del]'));
  del.id = 'd' + id;
  li.appendChild(del);
  // append everything to the list
  $('#list').append(li);
  //reset the text box
  $('#text').val('');
  saveList();
}
```

Save changes to the file system

Any time a change is made to the list, we update the file. The file will always reflect the current state of the list and we'll never have to click a save button. It just iterates through the list, adding a new line to each one.

```
function saveList(){
  if(store.isDirectory) return;
  var packet = '';
  for(var i in todolist)
  {
    packet += todolist[i] + '\r\n';
  }
  var bytes = new air.ByteArray();
  bytes.writeUTFBytes(packet);
  fileStream.open(store, air.FileMode.WRITE);
  fileStream.writeBytes(bytes, 0, bytes.length);
  fileStream.close();
}
```

One important thing to mention here is that we check if the store is a directory first. The reason we do this goes back to our `browseForSave` call. If the user cancels the dialog without selecting a file first, then the store points to the `documentsDirectory` that we set it to initially. Since we haven't specified a file, there's no place to save the list.

Hopefully by this point, you've been thinking of some cool ways to pimp out your list. Now we need to package this up so that we can let other people use it, too.

CREATING A PACKAGE

Now that we've created our application, we need to package it up so that we can distribute it. This is a two step process. The first step is to create a code signing certificate (or you can pay for one from Thawte which will help authenticate you as an AIR application developer).

To create a self-signed certificate, run the following command. This will create a PFX file that you'll use to sign your application.

```
adt -certificate -cn todo24ways 1024-RSA todo24ways.pfx
mypassword
```

After you've done that, you'll need to create the package with the certificate

```
adt -package -storetype pkcs12 -keystore todo24ways.pfx
todo24ways.air application.xml .
```

The important part to mention here is the period at the end of the command. We're telling it to package up all files in the current directory.

After that, just run the AIR file, which will install your application and run it.

IMPORTANT THINGS TO REMEMBER ABOUT AIR

When developing an HTML application, the rendering engine is Webkit. You'll thank your lucky stars that you aren't struggling with cross-browser issues. (My personal favourites are multiple backgrounds and border radius!)

Be mindful of memory leaks. Things like Ajax calls and event binding can cause applications to slowly leak memory over time. Web pages are normally short lived but desktop applications are often open for hours, if not days, and you may find your little desktop application taking up more memory than anything else on your machine!

The WebKit runtime itself can also be a memory hog, usually taking about 15MB just for itself. If you create multiple HTML windows, it'll add another 15MB to your memory footprint. Our little to-do list application shouldn't be much of a concern, though.

The other important thing to remember is that you're still essentially running within a Flash environment. While you probably won't notice this working in small applications, the moment you need to move to multiple windows or need to accomplish stuff beyond what HTML and JavaScript can give you, the need to understand some of the Flash-based elements will become more important.

Lastly, the other thing to remember is that HTML links will load within the AIR application. If you want a link to open in the users web browser, you'll need to capture that event and handle it on your own. The following code takes the `HREF` from a clicked link and opens it in the default web browser.

```
air.navigateToURL(new air.URLRequest(this.href));
```

ONLY THE BEGINNING

Of course, this is only the beginning of what you can do with Adobe AIR. You don't have the same level of control as building a native desktop application, such as being able to launch other applications, but you do have more control than what you could have within a web application. Check out the Adobe AIR Developer Center for HTML and Ajax for tutorials and other resources.

Now, go forth and create your desktop applications and hopefully you finish all your shopping before Christmas!

Download the example files.

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. Diagnostic Styling

Eric Meyer 24ways.org/200720

We’re all used to using CSS to make our designs live and breathe, but there’s another way to use CSS: to find out where our markup might be choking on missing accessibility features, targetless links, and just plain missing content.

Note: the techniques discussed here mostly work in Firefox, Safari, and Opera, but not Internet Explorer. I’ll explain why that’s not really a problem near the end of the article — and no, the reason is *not* “everyone should just ignore IE anyway”.

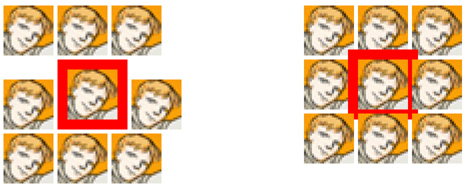
BASIC DIAGNOSTICS

To pick a simple example, suppose you want to call out all holdover `font` and `center` elements in a site. Simple: you just add the following to your styles.

```
font, center {outline: 5px solid red;}
```

You could take it further and add in a nice lime background or some such, but big thick red outlines should suffice. Now you'll be able to see the offenders wherever as you move through the site. (Of course, if you do this on your public server, everyone else will see the outlines too. So this is probably best done on a development server or local copy of the site.)

Not everyone may be familiar with outlines, which were introduced in CSS2, so a word on those before we move on. Outlines are much like borders, except outlines don't affect layout. Eh? Here's a comparison.



On the left, you have a border. On the right, an outline. The border takes up layout space, pushing other content around and generally being a nuisance. The outline, on the other hand, just draws into quietly into place. In most current browsers, it will overdraw any content already onscreen, and will be overdrawn by any content placed later — which is why it overlaps the images above it, and is overlapped by those below it.

Okay, so we can outline deprecated elements like `font` and `center`. Is that all? Oh no.

ATTRIBUTION

Let's suppose you also want to find any instances of inline style — that is, use of the `style` attribute on elements in the markup. This is generally discouraged (outside of HTML e-mails, which I'm not going to get anywhere near), as it's just another side of the same coin of using `font`: baking the presentation into the document structure instead of putting it somewhere more manageable. So:

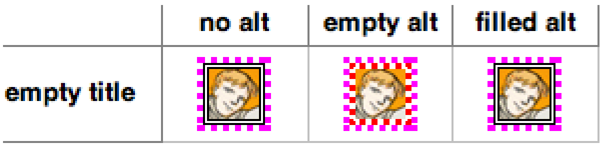
```
*[style], font, center {outline: 5px solid red;}
```

Adding that attribute selector to the rule's grouped selector means that we'll now be outlining any element with a `style` attribute.

There's a lot more that attribute selectors will let use diagnose. For example, we can highlight any images that have empty `alt` or `title` text.

```
img[alt=""] {border: 3px dotted red;}
img[title=""] {outline: 3px dotted fuchsia;}
```

Now, you may wonder why one of these rules calls for a border, and the other for an outline. That's because I want them to "add together" — that is, if I have an image which possesses both `alt` and `title`, and the values of both are empty, then I want it to be doubly marked.



	no alt	empty alt	filled alt
empty title			

See how the middle image there has both red and fuchsia dots running around it? (And am I the only one who sorely misses the *actual circular dots* drawn by IE5/Mac?) That's due to its markup, which we can see here in a fragment showing the whole table row.

```
<tr>
<th scope="row">empty title</th>
<td><img src="comic.gif" title="" /></td>
<td><img src="comic.gif" title="" alt="" /></td>
<td><img src="comic.gif" title="" alt="comical" /></td>
</tr>
```

Right, that's all well and good, but it misses a rather more serious situation: the selector `img[alt=""]` won't match an `img` element that doesn't even have an `alt` attribute. How to tackle this problem?

NOT A PROBLEM

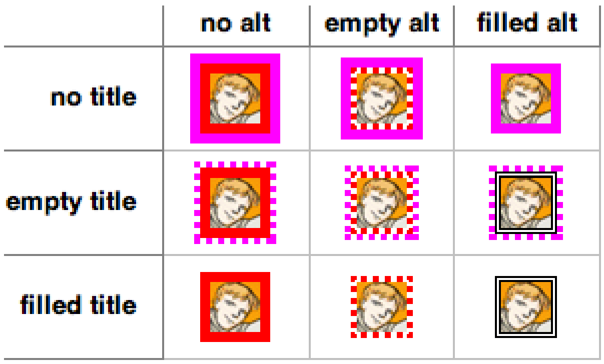
Well, if you want to select something based on a negative, you need a negative selector.

```
img:not([alt]) {border: 5px solid red;}
```

This is really quite a break from the rest of CSS selection, which is all positive: “select anything that has these characteristics”. With `:not()`, we have the ability to say (in supporting browsers) “select anything that *hasn’t* these characteristics”. In the above example, only `img` elements that do `not` have an `alt` attribute will be selected. So we expand our list of image-related rules to read:

```
img[alt=""] {border: 3px dotted red;}
img[title=""] {outline: 3px dotted fuchsia;}
img:not([alt]) {border: 5px solid red;}
img:not([title]) {outline: 5px solid fuchsia;}
```

With the following results:



We could expand this general idea to pick up tables who lack a summary, or have an empty `summary` attribute.

```
table[summary=""] {outline: 3px dotted red;}
table:not([summary]) {outline: 5px solid red;}
```

When it comes to selecting header cells that lack the proper scope, however, we have a trickier situation. Finding headers with no `scope` attribute is easy enough, but what about those that have a `scope` attribute with an incorrect value?

In this case, we actually need to pull an on-off maneuver. This has us setting *all* `th` elements to have a highlight style, and then turn it off for the elements that meet our criteria.

```
th {border: 2px solid red;}
th[scope="col"], th[scope="row"] {border: none;}
```

This was necessary because of the way CSS selectors work. For example, consider this:

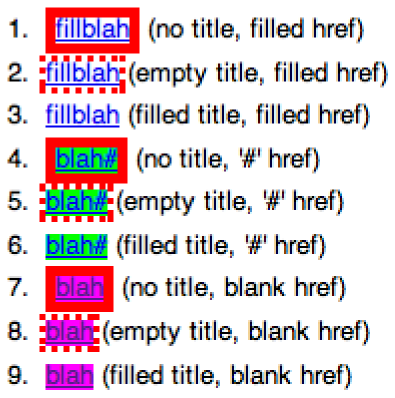
```
th:not([scope="col"]), th:not([scope="row"]) {border:
2px solid red;}
```

That would select…**all `th` elements, regardless of their attrributes**. That's because every `th` element doesn't have a scope of `col`, doesn't have a scope of `row`, or doesn't have either. There's no escaping this selector o' doom!

This limitation arises because `:not()` is limited to containing a single "thing" within its parentheses. You can't, for example, say "select all elements except those that are images which descend from list items". Reportedly, this limitation was imposed to make browser implementation of `:not()` easier.

Still, we can make good use of `:not()` in the service of further diagnosing. Calling out links in trouble is a breeze:

```
a[href]:not([title]) {border: 5px solid red;}
a[title=""] {outline: 3px dotted red;}
a[href="#"] {background: lime;}
a[href=""] {background: fuchsia;}
```



1. fillblah (no title, filled href)
2. fillblah (empty title, filled href)
3. fillblah (filled title, filled href)
4. blah# (no title, '#' href)
5. blah# (empty title, '#' href)
6. blah# (filled title, '#' href)
7. blah (no title, blank href)
8. blah (empty title, blank href)
9. blah (filled title, blank href)

Here we have a set that will call our attention to links missing `title` information, as well as links that have no valid target, whether through a missing URL or a JavaScript-driven page where there are no link fallbacks in the case of missing or disabled JavaScript (`href="#"`).

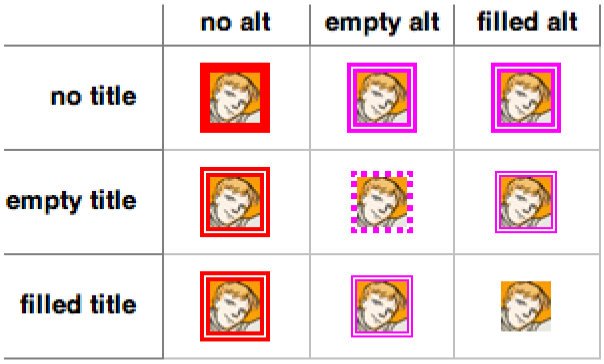
AND WHAT ABOUT IE?

As I said at the beginning, much of what I covered here doesn't work in Internet Explorer, most particularly `:not()` and `outline`. *(Oh, so basically everything? -Ed.)*

I can't do much about the latter. For the former, however, it's possible to hack your way around the problem by doing some layered on-off stuff. For example, for images, you replace the previously-shown rules with the following:

```
img {border: 5px solid red;}
img[alt][title] {border-width: 0;}
img[alt] {border-color: fuchsia;}
img[alt], img[title] {border-style: double;}
img[alt=""][title],
img[alt][title=""] {border-width: 3px;}
img[alt=""][title=""] {border-style: dotted;}
```

It won't have exactly the same set of effects, given the inability to use both borders and outlines, but will still highlight troublesome images.



It's also the case that IE6 and earlier lack support for even attribute selectors, whereas IE7 added pretty much all the attribute selector types there are, so the previous code block won't have any effect previous to IE7.

In a broader sense, though, these kinds of styles probably aren't going to be used in the wild, as it were. Diagnostic styles are something only you see as you work on a site, so you can make sure to use a browser that supports outlines and `:not()` when you're diagnosing. The fact that IE users won't see these styles is irrelevant since users of any browser probably won't be seeing these styles.

Personally, I always develop in Firefox anyway, thanks to its ability to become a full-featured IDE through the addition of extensions like Firebug and the Web Developer Toolbar.

YEAH, ABOUT THAT…

It's true that much of what I describe in this article is available in the WDT. I feel there are two advantages to writing your own set of diagnostic styles.

1. When you write your own styles, you can define exactly what the visual results will be, and how they will interact. The WDT doesn't let you make its outlines thicker or change their colors.

2. You can combine a bunch of diagnostics into a single set of rules and add it to your site's style sheet during the diagnostic portion, thus ensuring they persist as you surf around. This can be done in the WDT, but it isn't as easy (and, at least for me, not as reliable).

It's also true that a markup validator will catch many of the errors I mentioned, such as missing `alt` and `summary` attributes. For some, that's sufficient. But it won't catch everything diagnostic styles can, like empty `alt` values or untargeted links, which are perfectly valid, syntactically speaking.

DIAGNOSIS COMPLETE?

I hope this has been a fun look at the concept of diagnostic styling as well as a quick introduction into possibly new concepts like `:not()` and outlines. This isn't all there is to say, of course: there is plenty more that could be added to a diagnostic style sheet. And everyone's diagnostics will be different, tuned to meet each person's unique situation.

Mostly, though, I hope this small exploration triggers some creative thinking about the use of CSS to do more than just lay out pages and colorize text. Given the familiarity we acquire with CSS, it only makes sense to use it wherever it might be useful, and setting up visible diagnostic flags is just one more place for it to help us.

ABOUT THE AUTHOR

Eric Meyer has been working with the web since late 1993 and is an internationally recognized expert on the subjects of HTML, CSS, and Web standards. A widely read author, he is the principal consultant for Complex Spiral Consulting, which counts among its clients America On-Line; Apple Computer, Inc.; Wells Fargo Bank; and Macromedia. You can find more detailed information on Eric's personal Web page at http://www.meyerweb.com/eric/.

21. Mobile 2.0

Brian Fling 24ways.org/200721

THINKING 2.0

As web geeks, we have a thick skin towards jargon. We all know that "Web 2.0" has been done to death. At Blue Flavor we even have a jargon bucket to penalize those who utter such painfully overused jargon with a cash deposit. But Web 2.0 is a term that has lodged itself into the conscience of the masses. This is actually a good thing.

The 2.0 suffix was able to succinctly summarize all that was wrong with the Web during the dot-com era as well as the next evolution of an evolving media. While the core technologies actually stayed basically the same, the principles, concepts, interactions and contexts were radically different.

With that in mind, this Christmas I want to introduce to you the concept of Mobile 2.0. While not exactly a new concept in the mobile community, it is relatively unknown in the web community. And since the foundation of Mobile 2.0 is the web, I figured it was about time for you to get to know each other.

IT'S THE CARRIERS' WORLD. WE JUST LIVE IN IT.

Before getting into Mobile 2.0, I thought first I should introduce you to its older brother. You know the kind, the kid with emotional problems that likes to beat up on you and your friends for absolutely no reason. That is the mobile of today.

The mobile ecosystem is a very complicated space often and incorrectly compared to the Web. If the Web was a freewheeling hippie — believing in freedom of information and the unity of man through communities — then Mobile is the cutthroat capitalist — out to pillage and plunder for the sake of the almighty dollar. Where the Web is relatively easy to publish to and ultimately make a buck, Mobile is wrought with layers of complexity, politics and obstacles.

I can think of no better way to summarize these challenges than the testimony of Jason Devitt to the United States Congress in what is now being referred to as the "iPhone Hearing." Jason is the co-founder and CEO of SkyDeck a new wireless startup and former CEO of Vindigo an early pioneer in mobile content.

As Jason points out, the mobile ecosystem is a closed door environment controlled by the carriers, forcing the independent publisher to compete or succumb to the will of corporate behemoths.

But that is all about to change.

INTRODUCING MOBILE 2.0

Mobile 2.0 is term used by the mobile community to describe the current revolution happening in mobile. It describes the convergence of mobile and web services, adding portability, ubiquitous connectivity and location-aware services to add physical context to information found on the Web.

It's an important term that looks toward the future. Allowing us to imagine the possibilities that mobile technology has long promised but has yet to deliver. It imagines a world where developers can publish mobile content without the current constraints of the mobile ecosystem.

Like the transition from Web 1.0 to 2.0, it signifies the shift away from corporate or brand-centered experiences to user-centered experiences. A focus on richer interactions, driven by user goals. Moving away from proprietary technologies to more open and standard ones, more akin to the Web. And most importantly (from our perspective as web geeks) a shift away from kludgy one-off mobile applications toward using the Web as a platform for content and services.

This means the world of the Web and the world of Mobile are coming together faster than you can say ARPU (Average Revenue Per User, a staple mobile term to you webbies). And this couldn't come at a better time. The importance of understanding and addressing user context is quickly becoming a crucial consideration to every interactive experience as the number of ways we access information on the Web increases.

Mobile enables the power of the Web, the collective information of millions of people, inherit payment channels and access to just about every other mass media to literally be overlaid on top of the physical world, in context to the person viewing it.

Anyone who can’t imagine how the influence of mobile technology can’t transform how we perform even the simplest of daily tasks needs to get away from their desktop and see the new evolution of information.

THE INSTIGATORS

But what will make Mobile 2.0 move from idillic concept to a hardened market reality in 2008 will be four key technologies. Its my guess that you know each them already.

1. Opera

Opera is like the little train that could. They have been a driving force on moving the Web as we know it on to mobile handsets. Opera technology has proven itself to be highly adaptable, finding itself preloaded on over 40 million handsets, available on televisions sets through Nintendo Wii or via the Nintendo DS.

2. WebKit

Many were surprised when Apple chose to use KHTML instead of Gecko (the guts of Firefox) to power their Safari rendering engine. But WebKit has quickly evolved to be a powerful and flexible browser in the mobile context. WebKit has been in Nokia smartphones for a few years now, is the technology behind Mobile Safari in the iPhone and the iPod Touch and is the default web technology in Google's open mobile platform effort, Android.

3. The iPhone

The iPhone has finally brought the concepts and principles of Mobile 2.0 into the forefront of consumers minds and therefore developers' minds as well. Over 500 web applications have been written specifically for the iPhone since its launch. It's completely unheard of to see so many applications built for the mobile context in such a short period of time.

4. CSS & Javascript

Web 2.0 could not exist without the rich interactions offered by CSS and Javascript, and Mobile 2.0 is no different. CSS and Javascript support across multiple phones historically has been, well… to put it positively… utter crap.

Javascript finally allows developers to create interesting interactions that support user goals and the mobile context. Specially, AJAX allows us to finally shed the days of bloated Java applications and focus on portable and flexible web applications. While CSS — namely CSS3 — allows us to create designs that are as beautiful as they are economical with bandwidth and load times.

With Leaflets, a collection of iPhone optimized web apps we created, we heavily relied on CSS3 to cache and reuse design elements over and over, minimizing download times while providing an elegant and user-centered design.

IN CONCLUSION

It is the combination of all these instigators that is significantly decreasing the bar to mobile publishing. The market as Jason Devitt describes it, will begin to fade into the background. And maybe the world of mobile will finally start looking more like the Web that we all know and love.

So after the merriment and celebration of the holiday is over and you look toward the new year to refresh and renew, I hope that you take a seriously consider the mobile medium.

By this time next year, it is predicted that one-third of humanity will be using mobile devices to access the Web.

ABOUT THE AUTHOR

Brian Fling has been a leader in the web and mobile user experience. He has worked with several Fortune 500 companies to help design and develop their online experiences. Brian is a frequent speaker and author on the issues on mobile design, the mobile web and mobile user experience.

He co-created a series of iPhone web applications called Leaflets to showcase the concepts of "Mobile 2.0" just two weeks after the iPhone launched. Brian co-authored the dotMobi Mobile Web Developers Guide, the first free publication to cover mobile web design and development from start to finish. He runs one of the largest online communities focused on mobile design. He is currently writing O'Reilly Media's first book mobile, Mobile Design and Development.

Today Brian runs a small studio called Fling Media with his wife Cyndi.

22. How Media Studies Can Massage Your Message

Molly Holzschlag 24ways.org/200722

A young web designer once told his teacher 'just get to the meat already.' He was frustrated with what he called the 'history lessons and name-dropping' aspects of his formal college course. He just wanted to learn how to build Web sites, not examine the reasons why.

Technique and theory are both integrated and necessary portions of a strong education. The student's perspective has direct value, but also holds a distinct sorrow: Knowing the how without the why creates a serious problem. Without these surrounding insights we cannot tap into the influence of the history and evolved knowledge that came before. We cannot properly analyze, criticize, evaluate and innovate beyond the scope of technique.

History holds the key to transcending former mistakes. Philosophy encourages us to look at different points of view. Studying media and social history empowers us as Web workers by bringing together myriad aspects of humanity in direct relation to the environment of society and technology. Having an understanding of media, communities, communication arts as well as logic, language and computer savvy are all core skills of the best of web designers in today's medium.

CONTROLLING THE MESSAGE

> 'The computer can't tell you the emotional story. It can give you the exact mathematical design, but what's missing is the eyebrows.' – Frank Zappa

Media is meant to express an idea. The great media theorist Marshall McLuhan suggests that not only is media interesting because it's about the expression of ideas, but that the media itself actually shapes the way a given idea is perceived. This is what McLuhan meant when he uttered those famous words: 'The medium is the message.'

If instead of actually serving a steak to a vegetarian friend, what might a painting of the steak mean instead? Or a sculpture of a cow? Depending upon the form of media in question, the message is altered.

Figure 1

Must we know the history of cows to appreciate the steak on our plate? Perhaps not, but if we begin to examine how that meat came to be on the plate, the social, cultural and ideological associations of that cow, we begin to understand the complexity of both medium and message. A piece of steak on my plate makes me happy. A vegetarian friend from India might disagree and even find that that serving her a steak was very insensitive.

Takeaway: Getting the message right involves understanding that message in order to direct it to your audience accordingly.

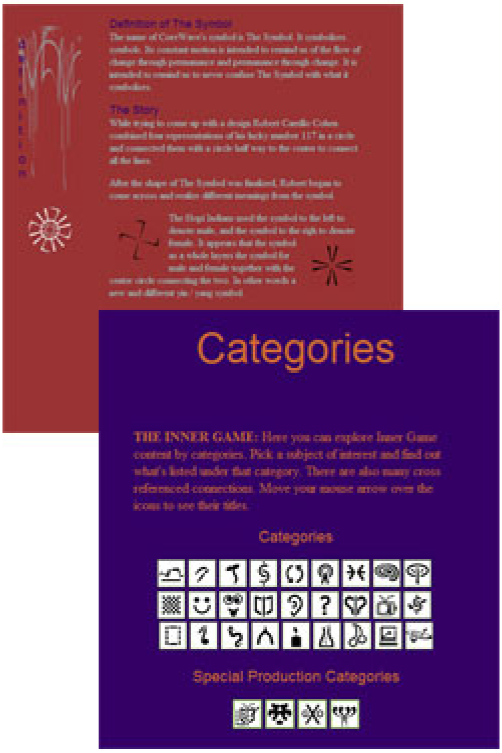
A SEPARATE PIECE

If we revisit the student who only wants technique, while he might become extremely adept at the rendering of an idea, without an understanding of the medium, how is he to have greater control over how that idea is perceived? Ultimately, his creativity is limited and his perspective narrowed, and the teacher has done her student a disservice without challenging him, particularly in a scholastic environment, to think in liberal, creative and ultimately innovative terms.

For many years, web pundits including myself have promoted the idea of separation as a core concept within creating effective front-end media for the Web. By this, we've meant literal separation of the technologies and documents: Markup for content; CSS for presentation; DOM Scripting for behavior. While the message of separation was an important part of understanding and teaching best practices for manageable, scalable sites, that separation is really just a separation of pieces, not of entire disciplines.

For in fact, the medium of the Web is an integrated one. That means each part of the desired message must be supported by the media silos within a given site. The visual designer must study the color, space, shape and placement of visual objects (including type) into a meaningful expression. The underlying markup is ideally written as semantically as possible, promote the meaning of the content it describes. Any interaction and functionality must make the process of the medium support, not take away from, the meaning of the site or Web application.

EXAMINATION: THE GLASS BEAD GAME



Definition of The Symbol

The name of CoreWeb's symbol is The Symbol. It symbolizes symbols. Its constant motion is intended to remind us of the flow of change through permanence and permanence through change. It is intended to remind us to never confuse The Symbol with what it symbolizes.

The Story

While trying to come up with a design Robert Carillo Cohen combined four representations of his lucky number 117 in a circle and connected them with a circle half way to the center to connect all the lines.

After the shape of The Symbol was finalized, Robert began to come across and realize different meanings from the symbol.

The Hopi Indians used the symbol to the left to denote male, and the symbol to the right to denote female. It appears that the symbol as a whole layers the symbol for male and female together with the center circle connecting the two. In other words a new and different yin / yang symbol.

Categories

THE INNER GAME: Here you can explore Inner Game content by categories. Pick a subject of interest and find out what's listed under that category. There are also many cross referenced connections. Move your mouse arrow over the icons to see their titles.

Categories

Special Production Categories

Figure 2

Figure 2 shows two screenshots of CoreWave's historic 'Glass Bead Game.' Fashioned after Herman Hesse's novel of the same name, the game was an exploration of how ideas are connected to other ideas via multiple forms of media. It was created for the Web in 1996 using server-side randomization with .htmlx files in order to allow players to see how random associations are in fact not random at all.

Takeaway: We can use the medium itself to explore creative ideas, to link us from one idea to the next, and to help us better express those ideas to our audiences.

COMPUTERS AND HUMAN INTERACTION

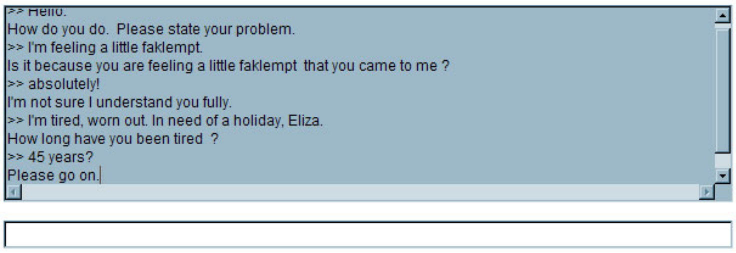
Since our medium involves computers and human interaction, it does us well to look to foundations of computers and reason. Not long ago I was chatting with Jared Spool on IM about this and that, and he asked me 'So how do you feel about that?' This caused me no end of laughter and I instantly quipped back 'It's okay by me ELIZA.' We both enjoyed the joke, but then I tried to share it with another dare I say younger colleague, and the reference was lost.

Raise your hand if you got the reference! Some of you will, but many people who come to the Web medium do not get the benefit of such historical references because we are not formally educated in them. Joseph Weizenbaum created the ELIZA program, which emulates a Rogerian Therapist, in 1966. It was an early study of computers and natural human language. I was a little over 2 years old, how about you?

CONVERSATION WITH ELIZA

There are fortunately a number of ELIZA emulators on the Web. I found one at http://www.chayden.net/eliza/Eliza.html that actually contains the source code (in Java) that makes up the ELIZA script.

Figure 3 shows a screen shot of the interaction. ELIZA first welcomes me, says ‘Hello, How do you do. Please state your problem’ and we continue in a short loop of conversation, the computer using cues from my answers to create new questions and leading fragments of conversation.



```
>> Hello.
How do you do.  Please state your problem.
>> I'm feeling a little faklempt.
Is it because you are feeling a little faklempt  that you came to me ?
>> absolutely!
I'm not sure I understand you fully.
>> I'm tired, worn out. In need of a holiday, Eliza.
How long have you been tired  ?
>> 45 years?
Please go on.
```

Figure 3

Albeit a very limited demonstration of how humans could interact with a computer in 1966, it’s amusing to play with now and compare it to something as richly interactive as the Microsoft Surface (Figure 4). Here, we see clear Lucite blocks that display projected video. Each side of the block has a different view of the video, so not only does one have to match up the images as they are moving, but do so in the proper directionality.

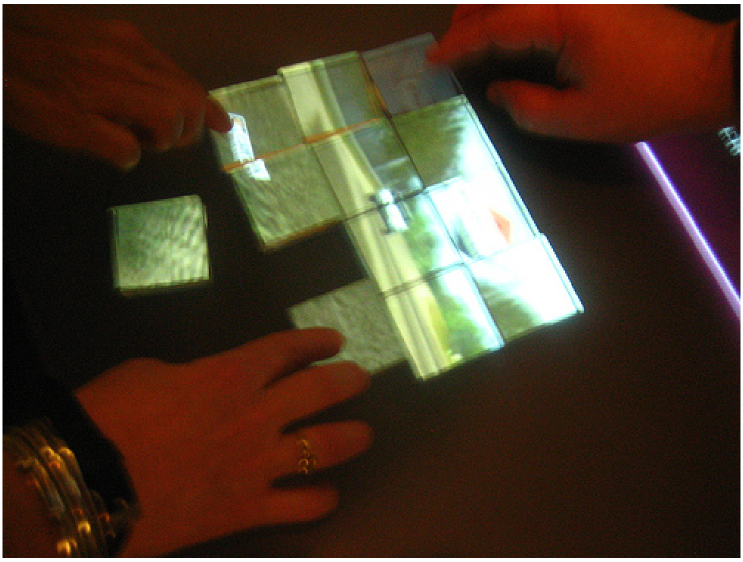


Figure 4

Takeway: the better we know our environment, the more we can alter it to emulate, expand and even supersede our message.

LEVERAGING HOLIDAY CHEER

Since most of us at least have a few days off for the holidays now that Christmas is upon us, now's a perfect time to reflect on ones' environment and examine the messages within it. Convince your spouse to find you a few audio books for stocking stuffers. Find interactive games to play with your kids and observe them, and yourself, during the interaction. Pour a nice egg-nog and sit down with a copy of Marshall McLuhan's 'The Medium is the Massage.' Leverage that holiday cheer and here's to a prosperous, interactive new year.

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

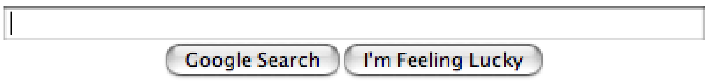
23. A Gift Idea For Your Users: Respect, Yo

Brian Oberkirch 24ways.org/200723

If, indeed, it *is* the thought that counts, maybe we should pledge to make more thoughtful design decisions. In addition to wowing people who use the Web sites we build with novel features, nuanced aesthetics and the new new thing, maybe we should also thread some subtle things throughout our work that let folks know: *hey, I'm feeling ya. We're simpatico. I hear you loud and clear.*

It's not just holiday spirit that moves me to talk this way. As good as people are, we need more than the horizon of karma to overcome that invisible demon, inertia. Makers of the Web, respectful design practices aren't just the right thing, they are good for business. Even if your site is the one and only place to get experience x, y or zed, you don't rub someone's face in it. You keep it free flowing, you honor the possible back and forth of a healthy transaction, you are Johnny Appleseed with the humane design cues. You make it clear that you are in it for the long haul.

A peek back:



Think back to what search (and strategy) was like before Google launched a super clean page with “I’m Feeling Lucky” button. Aggregation was the order of the day (just go back and review all the ‘strategic alliances’ that were announced daily.) Yet the GOOG comes along with this zen layout (nope, we’re not going to try to make you look at one of our media properties) and a bold, brash, teleport-me-straight-to-the-first-search-result button. It could have been titled “We’re Feeling Cocky”. These were radical design decisions that reset how people thought about search services. Oh, you mean I can just find what I want and get on with it?

It’s maybe even more impressive today, after the GOOG has figured out how to monetize attention better than anyone. “I’m Feeling Lucky” is still there. No doubt, it costs the company millions. But by leaving a little money on the table, they keep the basic bargain they started to strike in 1997. We’re going to get you where you want to go as quickly as possible.

Where are the places we might make the same kind of impact in our work? Here are a few ideas:

Respect People's Time

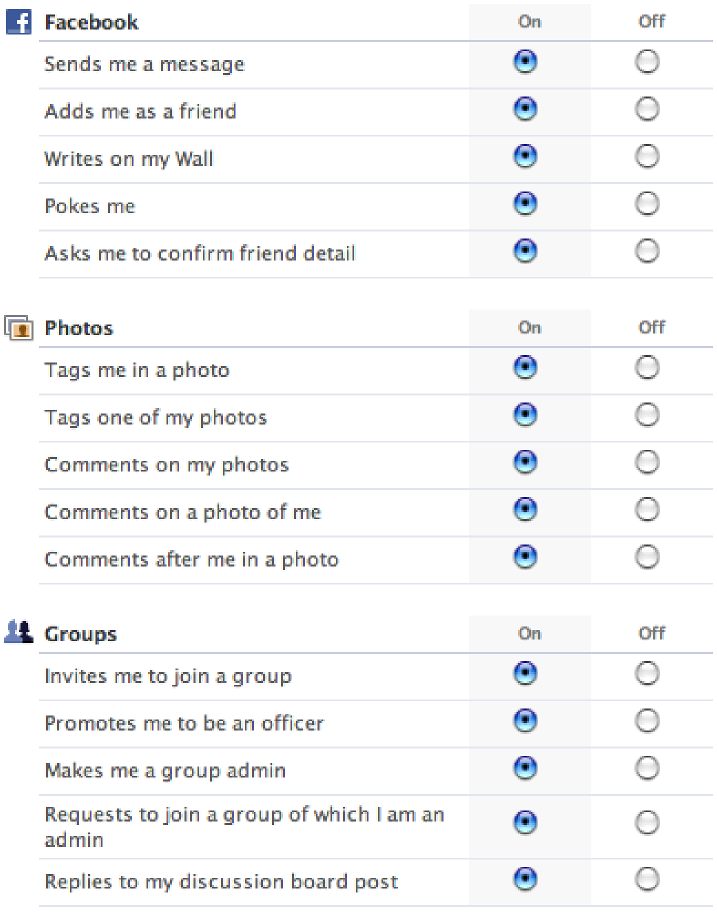
As more services become more integrated with our lives, this will only become more important. How can you make it clear that you respect the time a user has granted you?

USER-ORIENTED DEFAULTS

Default design may be the psionic tool in your belt. Unseen, yet pow-er-ful. Look at your defaults. Who are they set up to benefit? Are you depending on users not checking off boxes so you can feel ok about sending them email they really don't want? Are you depending on users not checking off boxes so you tilt privacy values in ways most beneficial for your SERPs? Are you making it a little too easy for 3rd party applications to run buckwild through your system?

There's being right and then there's being awesome. Design to the spirit of the agreement and not the letter.

See this?



Facebook	On	Off
Sends me a message	●	○
Adds me as a friend	●	○
Writes on my Wall	●	○
Pokes me	●	○
Asks me to confirm friend detail	●	○

Photos	On	Off
Tags me in a photo	●	○
Tags one of my photos	●	○
Comments on my photos	●	○
Comments on a photo of me	●	○
Comments after me in a photo	●	○

Groups	On	Off
Invites me to join a group	●	○
Promotes me to be an officer	●	○
Makes me a group admin	●	○
Requests to join a group of which I am an admin	●	○
Replies to my discussion board post	●	○

Make sure that's *really* the experience you think people want. Whenever I see a service that defaults to **not** opting me in their newsletter just because I bought a t shirt from them, you can be sure that I trust them that much more. And they are likely to see me again.

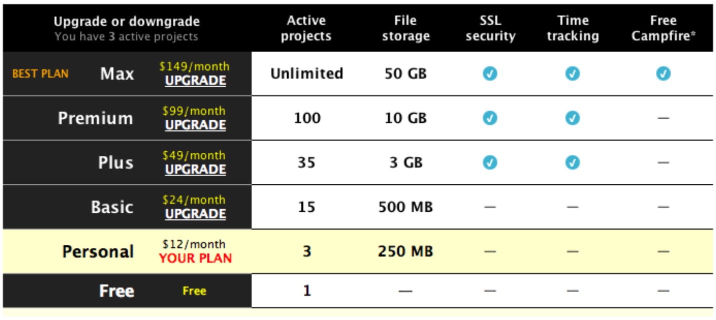
REDUCE, REUSE

It’s likely that people using your service will have data and profile credentials elsewhere. You should really think hard about how you can let them repurpose some of that work within your system. Can you let them reduce the number of logins/passwords they have to manage by supporting OpenID? Can you let them reuse profile information from another service by slurping in (or even subscribing) to hCards? Can you give them a leg up by reusing a friends list they make available to you? (Note: please avoid the anti-pattern of inviting your user to upload all her credential data from 3rd party sites into your system.)

This is a much larger issue, and if you’d like to get involved, have a look at the wiki, and dive in.

MAKE IT SIMPLE TO LEAVE

Oh, this drives me bonkers. Again, the more simple you make it to increase or decrease involvement in your site, or to just opt-out altogether, the better. This example from Basecamp is instructive:

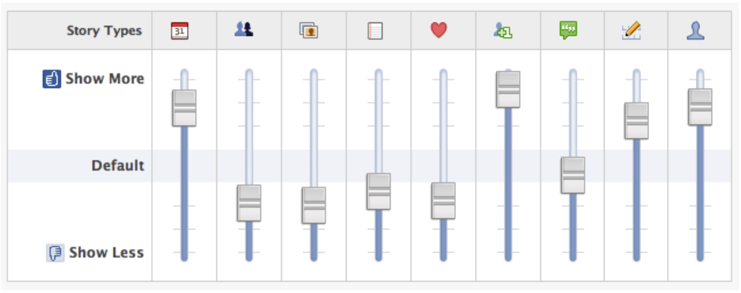


Upgrade or downgrade You have 3 active projects			Active projects	File storage	SSL security	Time tracking	Free Campfire*
BEST PLAN	Max	$149/month UPGRADE	Unlimited	50 GB	✓	✓	✓
	Premium	$99/month UPGRADE	100	10 GB	✓	✓	–
	Plus	$49/month UPGRADE	35	3 GB	✓	✓	–
	Basic	$24/month UPGRADE	15	500 MB	–	–	–
	Personal	$12/month YOUR PLAN	3	250 MB	–	–	–
	Free	Free	1	—	–	–	–

At a glance, I can see what the implications are of choosing a different type of account. I can also move between account levels with one click. Finally, I can cancel the service easily. No hoop jumping. Also, it should be simple for users to take data with them or delete it.

Let Them Have Fun

Don't overlook opportunities for pleasure. Even the most mundane tasks can be made more enjoyable. Check out one of my favorite pieces of interaction design from this past year:



Holy knob fiddling, Batman! What a great way to get people to play with preference settings: an equalizer metaphor. Those of a certain age will recall how fun it was to make patterns with your uncle's stereo EQ. I think this is a delightful way to encourage people to optimize their own experience of the news feed feature. Given the killer nature of this feature, it was important for Facebook to make it easy to fine tune.

I'd also point you to Flickr's Talk Like A Pirate Day Easter egg as another example of design that delights. What a huge amount of work for a one-off, totally optional way to experience the site. And what fun. And how true to its brand persona. Brill.

Anti-hype

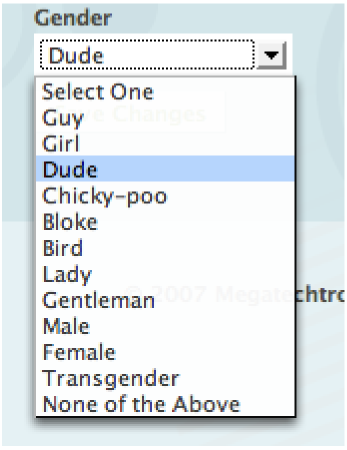
Don't talk so much. Rather, ship and sample. Release code, tell the right users. See what happens. Make changes. Extend the circle a bit by showing some other folks. Repeat.

The more you hype coming features, the more you talk about what isn't yet, the more you build unrealistic expectations. Your genius can't possibly match our collective dreaming. Disappointment is inevitable. Yet, if you craft the right melody and make it simple for people to hum your tune, it will spread. Give it time. Listen.

Speak the Language of the Tribe

It’s respectful to speak in a human way. Not that you have to get all zOMGWTFBBQ!!1 in your messaging. People respond when you speak to them in a way that sounds natural. Natural will, of course, vary according to context. Again, listen in and people will signal the speech that works in that group for those tasks. Reveal those cues in your interface work and you’ll have powerful proof that actual people are working on your Web site.

This example of Pownce‘s gender selector is the kind of thing I’m talking about:



Now, this doesn’t mean you should mimic the lingo on some cool kidz site. Your service doesn’t need to have a massage when it’s down. Think about what works for you and your tribe. Excellent advice here from Feedburner’s

Dick Costolo on finding a voice for your service. Also, Mule Design's Erika Hall has an excellent talk on improving your word fu.

Pass the mic, yo

Here is a crazy idea: you could ask your users what they want. Maybe you could even use this input to figure out what they *really* want. Tools abound. Comments, wikis, forums, surveys. Embed the sexy new Get Satisfaction widget and have a dynamic FAQ running.

The point is that you make it clear to people that they have a means of shaping the service with you. And you must showcase in some way that you are listening, evaluating and taking action against some of that user input.

You get my drift. There are any number of ways we can show respect to those who gift us with their time, data, feedback, attention, evangelism, money. Big things are in the offing. I can feel the love already.

ABOUT THE AUTHOR

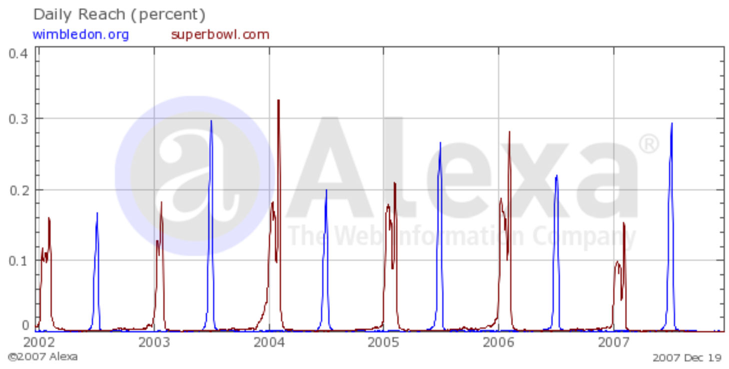
Brian Oberkirch yammers a lot. He used to teach literature. And do radio news. And write newspaper articles. Now he helps people make Web stuff and talks up things like microformats and other open design approaches that should make life better for everyone. He carries on at brianoberkirch.com.

24. Performance On A Shoe String

Drew McLellan 24ways.org/200724

Back in the summer, I happened to notice the official Wimbledon All England Tennis Club site had jumped to the top of Alexa's Movers & Shakers list — a list that tracks sites that have had the biggest upturn or downturn in traffic. The lawn tennis championships were underway, and so traffic had leapt from almost nothing to crazy-busy in a no time at all.

Many sites have similar peaks in traffic, especially when they're based around scheduled events. No one cares about the site for most of the year, and then all of a sudden – wham! – things start getting warm in the data centre. Whilst the thought of chestnuts roasting on an open server has a certain appeal, it's less attractive if you care about your site being available to visitors. Take a look at this Alexa traffic graph showing traffic patterns for superbowl.com at the beginning of each year, and wimbledon.org in the month of July.



Traffic graph from Alexa.com

Whilst not on the same scale or with such dramatic peaks, we have a similar pattern of traffic here at 24ways.org. Over the last three years we've seen a dramatic pick up in traffic over the month of December (as would be expected) and then a much lower, although steady load throughout the year. What we do have, however, is the luxury of knowing when the peaks will be. For a normal site, be that a blog, small scale web app, or even a small corporate site, you often just cannot predict when you might get slashdotted, end up on the front page of Digg or linked to from a similarly high-profile site. You just don't know when the peaks will be.

If you're a big commercial enterprise like the Super Bowl, scaling up for that traffic is simply a cost of doing business. But for most of us, we can't afford to have massive capacity sat there unused for 90% of the year. What you have to do instead is work out how to deal with as much traffic as possible with the modest resources you have.

In this article I'm going to talk about some of the things we've learned about keeping 24 ways running throughout December, whilst not spending a fortune on hosting we don't need for 11 months of each year. We've not always got it right, but we've learned a lot along the way.

THE PROBLEM

To know how to deal with high traffic, you need to have a basic idea of what happens when a request comes into a web server. 24 ways is hosted on a single small *virtual dedicated* server with a great little hosting company in the UK. We run Apache with PHP and MySQL all on that one server. When a request comes in a new Apache process is started to deal with the request (or assigned if there's one available not doing anything). Each process takes a bunch of memory, so there's a finite number of processes that you can run, and therefore a finite number of pages you can serve at once before your server runs out of memory.

With our budget based on whatever is left over after beer, we need to get best performance we can out of the resources available. As the goal is to serve as many pages as quickly as possible, there are several approaches we can take:

1. Reducing the amount of memory needed by each Apache process
2. Reducing the amount of time each process is needed
3. Reducing the number of requests made to the server

Yahoo! have published some information on what they call Exceptional Performance, which is well worth reading, and compliments many of my examples here. The Yahoo! guidelines very much look at things from a user perspective, which is always important.

SERVER TWEAKING

If you're in the position of being able to change your server configuration (our set-up gives us root access to what is effectively a virtual machine) there are some basic steps you can take to maximise the available memory and reduce the memory footprint. Without getting too boring and technical (whole books have been written on this) there are a couple of things to watch out for.

Firstly, check what processes you have running that you might not need. Every megabyte of memory that you free up might equate to several thousand extra requests being served each day, so take a look at `top` and see what's using up your resources. Quite often a machine configured as a web server will have some kind of mail server running by default. If your site doesn't use mail (ours doesn't) make sure it's shut down and not using resources.

Secondly, have a look at your Apache configuration and particularly what modules are loaded. The method for doing this varies between versions of Apache, but again, every module loaded increases the amount of memory that each Apache process requires and therefore limits the number of simultaneous requests you can deal with.

The final thing to check is that Apache isn't configured to start more servers than you have memory for. This is usually done by setting the `MaxClients` directive. When that limit is reached, your site is going to stop responding to further requests. However, if all else goes well that threshold won't be reached, and if it does it will at least stop the weight of the traffic taking the entire server down to a point where you can't even log in to sort it out.

Those are the main tidbits I've found useful for this site, although it's worth repeating that entire books have been written on this subject alone.

CACHING

Although the site is generated with PHP and MySQL, the majority of pages served don't come from the database. The process of compiling a page on-the-fly involves quite a few trips to the database for content, templates, configuration settings and so on, and so can be slow and require a lot of CPU. Unless a new article or comment is published, the site doesn't actually change between requests and so it makes sense to generate each page once, save it to a file and then just serve all following requests from that file.

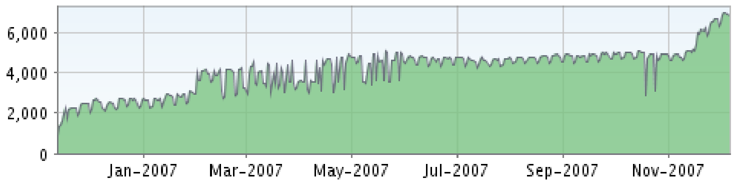
We use QuickCache (or rather a plugin based on it) for this. The plugin integrates with our publishing system (Textpattern) to make sure the cache is cleared when something on the site changes. A similar plugin called WP-Cache is available for WordPress, but of course this could be done any number of ways, and with any back-end technology.

The important principal here is to reduce the time it takes to serve a page by compiling the page once and serving that cached result to subsequent visitors. Keep away from your database if you can.

OUTSOURCE YOUR FEEDS

We get around 36,000 requests for our feed each day. That really only works out at about 7,000 subscribers averaging five-and-a-bit requests a day, but it's still 36,000 requests we could easily do without. Each request uses resources and particularly during December, all those requests can add up.

The simple solution here was to switch our feed over to using FeedBurner. We publish the address of the FeedBurner version of our feed here, so those 36,000 requests a day hit FeedBurner's servers rather than ours. In addition, we get pretty graphs showing how the subscriber-base is building.



OFF-LOAD BIG FILES

Larger files like images or downloads pose a problem not in bandwidth, but in the time it takes them to transfer. A typical page request is very quick, a few seconds at the most, resulting in the connection being freed up promptly. Anything that keeps a connection open for a long time is going to start killing performance very quickly.

This year, we started serving most of the images for articles from a subdomain – media.24ways.org. Rather than pointing to our own server, this subdomain points to an Amazon S3 account where the files are held. It's easy to pigeon-hole S3 as merely an online backup solution, and whilst not a fully fledged CDN, S3 works very nicely for serving larger media files. The roughly 20GB of files served this month have cost around $5 in Amazon S3 charges. That's so affordable it may not be worth even taking the files back off S3 once December has passed.

I found this article on Scalable Media Hosting with Amazon S3 to be really useful in getting started. I upload the files via a Firefox plugin (mentioned in the article) and then edit the ACL to allow public access to the files. The way S3 enables you to point DNS directly at it means that you're not tied to always using the service, and that it can be transparent to your users.

If your site uses photographs, consider uploading them to a service like Flickr and serving them directly from there. Many photo sharing sites are happy for you to link to images in this way, but do check the acceptable use policies in case you need to provide a credit or link back.

OFF-LOAD SMALL FILES

You'll have noticed the pattern by now – get rid of as much traffic as possible. When an article has a lot of comments and each of those comments has an avatar along with it, a great many requests are needed to fetch each of those images. In 2006 we started using Gravatar for avatars, but their servers were slow and were holding up page loads. To get around this we started caching the images on our server, but along with that came the burden of furnishing all the image requests.

Earlier this year Gravatar changed hands and is now run by the same team behind WordPress.com. Those guys clearly know what they're doing when it comes to high performance, so this year we went back to serving avatars directly from them.

If your site uses avatars, it really makes sense to use a service like Gravatar where your users probably already have an account, and where the image requests are going to be dealt with for you.

KNOW WHAT YOU'RE PAYING FOR

The server account we use for 24 ways was opened in November 2005. When we first hit the front page of Digg in December of that year, we upgraded the server with a bit more memory, but other than that we were still running on that 2005 spec for two years. Of course, the world of technology has moved on in those years, prices have dropped and specs have improved. For the same amount we were paying for that 2005 spec server, we could have an account with twice the CPU, memory and disk space.

So in November of this year I took out a new account and transferred the site from the old server to the new. In that single step we were prepared for dealing with twice the amount of traffic, and because of a special offer at the time I didn't even have to pay the setup cost on the new server. So it really pays to know what you're paying for and keep an eye out of ways you can make improvements without needing to spend more money.

FURTHER STEPS

There's nearly always more that can be done. For example, there are some media files (particularly for older articles) that are not on S3. We also serve our CSS directly and it's not minified or compressed. But by tackling the big problems first we've managed to reduce load on the server and at the same time make sure that the load being placed on the server can be dealt with in the most frugal way.

Over the last 24 days we've served up articles to more than 350,000 visitors without breaking a sweat. On a busy day, that's been nearly 20,000 visitors in just 24 hours. While in the grand scheme of things that's not a huge amount of traffic, it can be a lot if you're not prepared for it. However, with a little planning for the peaks you can help ensure that when the traffic arrives you're ready to capitalise on it.

Of course, people only visit 24 ways for the wealth of knowledge and experience that's tied up in the articles here. Therefore I'd like to take the opportunity to thank all our authors this year who have given their time as a gift to the community, and to wish you all a very happy Christmas.

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.