24

2011

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.

2011

In October, Steve Jobs died. The thorniest part of responsive web design, and an arena for many competing and dissenting voices was images. 24 ways tackled that and many other issues head on: conditional loading; front-end style guides; icon fonts; and the importance of side projects.

1. Creating Custom Font Stacks with Unicode-Range

Drew McLellan 24ways.org/201101

Any web designer or front-end developer worth their salt will be familiar with the CSS `@font-face` rule used for embedding fonts in a web page. We've all used it — either directly in our code ourselves, or via one of the web font services like Fontdeck, Typekit or Google Fonts.

If you're like me, however, you'll be used to just copying and pasting in a specific incantation of lines designed to get different formats of fonts working in different browsers, and may not have really explored all the capabilities of `@font-face` properties as defined by the spec.

One such property — the `unicode-range` descriptor — sounds pretty dull and is easily overlooked. It does, however, have some fairly interesting possibilities when put to use in creative ways.

UNICODE-RANGE

The `unicode-range` descriptor is designed to help when using fonts that don't have full coverage of the characters used in a page. By adding a `unicode-range` property to a `@font-face` rule it is possible to specify the range of characters the font covers.

```
@font-face {
    font-family: BBCBengali;
    src: url(fonts/BBCBengali.ttf) format("opentype");
    unicode-range: U+00-FF;
}
```

In this example, the font is to be used for characters in the range of `U+00` to `U+FF` which runs from the unexciting control characters at the start of the Unicode table (symbols like the exclamation mark start at `U+21`) right through to ÿ at `U+FF` – the extent of the Basic Latin character range.

By adding multiple `@font-face` rules for the same family but with different ranges, you can build up complete coverage of the characters your page uses by using different fonts.

When I say that it's possible to specify the range of characters the font *covers*, that's true, but what you're really doing with the `unicode-range` property is declaring which characters the font should be *used for*. This becomes interesting, because instead of merely working with the technical constraints of available characters in a given font, we can start picking and choosing characters to use and selectively mix fonts together.

THE BEST AVAILABLE AMPERSAND

A few years back, Dan Cederholm wrote a post encouraging designers to use the best available ampersand. Dan went on to outline how this can be achieved by wrapping our ampersands in a `<span>` element with a class applied:

```
<span class="amp">&</span>
```

A CSS rule can then be written to select the `<span>` and apply a different font:

```
span.amp {
    font-family: Baskerville, Palatino, "Book Antiqua",
serif;
}
```

That's a perfectly serviceable technique, but the drawbacks are clear — you have to add extra markup which is borderline presentational, and you also have to be *able* to add that markup, which isn't always possible when working with a CMS.

Perhaps we could do this with `unicode-range`.

A BETTER BEST AVAILABLE AMPERSAND

The Unicode code point for an ampersand is `U+26`, so the ampersand font stack above can be created like so:

```
@font-face {
    font-family: 'Ampersand';
    src: local('Baskerville'), local('Palatino'),
local('Book Antiqua');
    unicode-range: U+26;
}
```

What we've done here is specify a new family called Ampersand and created a font stack for it with the user's locally installed copies of Baskerville, Palatino or Book Antiqua. We've then limited it to a single character range — the ampersand. Of course, those don't need to be local fonts — they could be web font files, too. If you have a font with a really snazzy ampersand, go for your life.

We can then use that new family in a regular font stack.

```
h1 {
    font-family: Ampersand, Arial, sans-serif;
}
```

With this in place, any `<h1>` elements in our page will use the Ampersand family (Baskerville, Palatino or Book Antiqua) for ampersands, and Arial for all other characters. If the user doesn't have any of the Ampersand family fonts available, the ampersand will fall back to the next item in the font stack, Arial.

YOU DIDN'T THINK IT WAS THAT EASY, DID YOU?

Oh, if only it were so. The problem comes, as ever, with the issue of browser support. The `unicode-range` property has good support in WebKit browsers (like Safari and Chrome, and the browsers on most popular smartphone platforms) and in recent versions of Internet Explorer. The big stumbling block comes in the form of Firefox, which has no support at all.

If you're familiar with how CSS works when it comes to unsupported properties, you'll know that if a browser encounters a property it doesn't implement, it just skips that declaration and moves on to the next. That works perfectly for things like `border-radius` — if the browser can't round off the corners, the declaration is skipped and the user sees square corners instead. Perfect.

Less perfect when it comes to `unicode-range`, because if no range is specified then the default is that the font is applied for all characters — the whole range. If you're using a fancy font for flamboyant ampersands, you probably don't want that applied to *all* your text if `unicode-range` isn't supported. That would be bad. Really bad.

ENSURING GOOD FALLBACKS

As ever, the trick is to make sure that there's a sensible fallback in place if a browser doesn't have support for whatever technology you're trying to use. This is where being a super nerd about understanding the spec you're working with really pays off.

We can make use of the rules of the CSS cascade to make sure that if `unicode-range` isn't supported we get a sensible fallback font. What would be ideal is if we were able to follow up the `@font-face` rule with a second rule to override it if Unicode ranges aren't implemented.

```
@font-face {
    font-family: 'Ampersand';
    src: local('Baskerville'), local('Palatino'),
local('Book Antiqua');
    unicode-range: U+26;
}
@font-face {
    font-family: 'Ampersand';
    src: local('Arial');
}
```

In theory, this code *should* make sense for all browsers. For those that support `unicode-range` the two rules become cumulative. They specify different ranges for the same family, and in WebKit browsers this has the expected result of using Arial for most characters, but Baskerville and friends for the ampersand. For browsers that don't have support, the second rule should just supersede the first, setting the font to Arial.

Unfortunately, this code causes current versions of Firefox to freak out and use the *first* rule, applying Baskerville to the entire range. That's both unexpected and unfortunate. Bad Firefox. On your rug.

If that doesn't work, what can we do? Well, we know that if given a `unicode-range` Firefox will ignore the range and apply the font to all characters. That's really what we're trying to achieve. So what if we specified a range for the fallback font, but made sure it only covers some obscure high-value Unicode character we're never going to use in our page? Then it wouldn't affect the outcome for browsers that *do* support ranges.

```
@font-face {
    font-family: 'Ampersand';
    src: local('Baskerville'), local('Palatino'),
local('Book Antiqua');
```

```
    unicode-range: U+26;
}
@font-face {
    /* Ampersand fallback font */
    font-family: 'Ampersand';
    src: local('Arial');
    unicode-range: U+270C;
}
```

By specifying a range on the fallback font, Firefox appears to correctly override the first based on the cascade sort order. Browsers that do support ranges take the second rule in addition, and apply Arial for that obscure character we're not using in any of our pages — `U+270C`.

So we get our nice ampersands in browsers that support `unicode-range` and, thanks to our styling of an obscure Unicode character, the font falls back to a perfectly acceptable Arial in browsers that do not offer support. Perfect!

That obscure character, my friends, is what Unicode defines as the VICTORY HAND.

SO, HOW CAN WE USE THIS?

Ampersands are a neat trick, and it works well in browsers that support ranges, but that's not really the point of all this. Styling ampersands is fun, but they're only really scratching the surface. Consider more involved examples, such as substituting a different font for numerals, or symbols, or even caps. Things certainly begin to get a bit more interesting.

How do you know what the codes are for different characters? Richard Ishida has a handy online conversion tool available where you can type in the characters and get the Unicode code points out the other end.

Of course, the fact remains that browser support for `unicode-range` is currently limited, so any application needs to have fallbacks that you're still happy for a significant proportion of your visitors to see. In some cases, such as dedicated pages for mobile devices in an HTML-based phone app, this is immediately useful as support in WebKit browsers is already very good. In other cases, you'll have to use your own best judgement based on your needs and audience.

One thing to keep in mind is that if you're using web fonts, the entire font will be downloaded even if only one character is used. That said, the font shouldn't be downloaded if none of the characters within the Unicode range are present in a given page.

As ever, there are pros and cons to using `unicode-range` as well as varied but increasing support in browsers. It remains a useful tool to understand and have in your toolkit for when the right moment comes along.

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. Conditional Loading for Responsive Designs

Jeremy Keith 24ways.org/201102

On the eighteenth day of last year's 24 ways, Paul Hammond wrote a great article called *Speed Up Your Site with Delayed Content*. He outlined a technique for loading some content — like profile avatars — after the initial page load. This gives you a nice performance boost.

There's another situation where this kind of delayed loading could be really handy: mobile-first responsive design.

Responsive design combines three techniques:

- a fluid grid
- flexible images
- media queries

At first, responsive design was applied to existing desktop-centric websites to allow the layout to adapt to smaller screen sizes. But more recently it has been combined with another innovative approach called mobile first.

Rather then starting with the big, bloated desktop site and then scaling down for smaller devices, it makes more sense to start with the constraints of the small screen and then scale up for larger viewports. Using this approach, your layout grid, your large images and your media queries are applied *on top* of the pre-existing small-screen design. It's taking progressive enhancement to the next level.

One of the great advantages of the mobile-first approach is that it forces you to really focus on the core content of your page. It might be more accurate to think of this as a content-first approach. You don't have the luxury of sidebars or multiple columns to fill up with content that's just nice to have rather than essential.

But what happens when you apply your media queries for larger viewports and you *do* have sidebars and multiple columns? Well, you can load in that nice-to-have content using the same kind of Ajax functionality that Paul described in his article last year. The difference is that you first run a quick test to see if the viewport is wide enough to accommodate the subsidiary content. This is conditional delayed loading.

Consider this situation: I've published an article about cats and I'd like to include relevant cat-related news items in the sidebar …but only if there's enough room on the screen for a sidebar.

I'm going to use Google's News API to return the search results. This is the ideal time to use delayed loading: I don't want a third-party service slowing down the rendering of my page so I'm going to fire off the request after my document has loaded.

Here's an example of the kind of Ajax function that I would write:

```
var searchNews = function(searchterm) {
  var elem = document.createElement('script');
  elem.src = 'http://ajax.googleapis.com/ajax/services/
search/news?v=1.0&q='+searchterm+'&callback=displayNews';

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

I've provided a callback function called `displayNews` that takes the JSON result of that Ajax request and adds it an element on the page with the ID `newsresults`:

```
var displayNews = function(news) {
  var html = '',
  items = news.responseData.results,
  total = items.length;
  if (total>0) {
    for (var i=0; i<total; i++) {
      var item = items[i];
      html+= '<article>';
      html+= '<a href="'+item.unescapedUrl+'">';
      html+= '<h3>'+item.titleNoFormatting+'</h3>';
      html+= '</a>';
      html+= '<p>';
      html+= item.content;
      html+= '</p>';
      html+= '</article>';
    }
    document.getElementById('newsresults').innerHTML =
html;
  }
};
```

Now, I can call that function at the bottom of my document:

```
<script>
    searchNews('cats');
</script>
```

If I only want to run that search when there's room for a sidebar, I can wrap it in an `if` statement:

```
<script>
if (document.documentElement.clientWidth > 640) {
    searchNews('cats');
}
</script>
```

If the browser is wider than 640 pixels, that will fire off a search for news stories about cats and put the results into the `newsresults` element in my markup:

```
<div id="newsresults">
    <!-- search results go here -->
</div>
```

This works pretty well but I'm making an assumption that people with small-screen devices wouldn't be interested in seeing that nice-to-have content. You know what they say about assumptions: they make an ass out of you and umptions. I should really try to give everyone at least the option to get to that extra content:

```
<div id="newsresults">
    <a href="http://www.google.com/
search?q=cats&tbm=nws">Search Google News</a>
</div>
```

See the result

Visitors with small-screen devices will see that link to the search results; visitors with larger screens will get the search results directly.

I've been concentrating on HTML and JavaScript, but this technique has consequences for content strategy and information architecture. Instead of thinking about possible page content in a binary way as either 'on the page' or 'not on the page', conditional loading introduces a third 'it's complicated' option.

This was just a simple example but I hope it illustrates that conditional loading could become an important part of the content-first responsive design approach.

ABOUT THE AUTHOR

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

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

3. Subliminal User Experience

Chris Sealey 24ways.org/201103

The term 'user experience' is often used vaguely to quantify common elements of the interaction design process: wireframing, sitemapping and so on. UX undoubtedly involves all of these principles to some degree, but there *really* is a lot more to it than that.

Good UX is characterized by providing the user with constant feedback as they step through your interface. It means thinking about and providing fallbacks and error resolutions in even the rarest of scenarios. It's about omitting clutter to make way for the necessary, and using the most fundamental of design tools to influence a user's path. It means making no assumptions, designing right down to the most distinct details and going one step further every single time. In many cases, good UX is completely subliminal.

There are simple tools and subtleties we can build into our products to enhance the overall experience but, in order to do so, we really have to step beyond where we usually draw the line on what to design.

The purpose of this article is not to provide technical how-tos, as the functionality is, in most cases, quite simple and could be implemented in a myriad of ways. Rather, it will present a handful of ideas for enhancing the experience of an interface at a deeper level of design without relying on the container.

We'll cover three elements that should get you thinking in the right mindset:

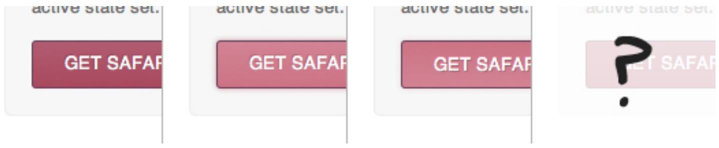
1. progress activity and post-active states
2. pseudo-class preloading
3. buttons and their (mis)behaviour

PROGRESS ACTIVITY AND THE POST-ACTIVE STATE

We've long established that we can't control the devices our products are viewed on, which browser they'll run in or what connection speed will be used to access them. We accept this all as factual, so why is it so often left to the browser to provide feedback to the user when an event is triggered or an error encountered? The browser isn't part of the interface — it's merely a container. A simple, visual recognition of your users' activity may be all it takes to make or break the product.

Let's begin with a commonly overlooked case: progress activity.

A user moves their cursor over a hyperlink or button, which is clearly defined as one by the visual language of your content. Upon doing so, they trigger the `:hover` state to confirm this element is indeed interactive. So far, so good. What happens next is where it starts to fall apart: the user hits this link, presumably triggering an `:active` state, which is then returned to the normal state upon release. And then what?



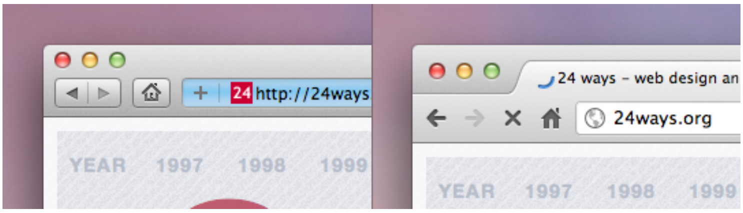
From this point on, your user is in limbo. The link has fallen back to either its regular or `:visited` state. You've effectively abandoned them and are relying entirely on the browser they're using to communicate that something is happening. This poses quite a few problems:

- The user may lose focus of what they were doing.
- There is little consistency between progress indication in browsers.

- The user may not even notice that their action has been acknowledged.

How many times have one or more of these events happened to you due to a lack of communication from the interface?

Think about the differences between Safari and Chrome in this area — two browsers that, when compared to each other, are relatively similar in nature, though this basic feature differs in execution.



Like all aspects of designing the user experience, there is no one true way to fix this problem, but we can introduce details that many users will unconsciously appreciate.

Consider the basic loading indicator. It's nothing new — in fact, some would argue it's quite a cliché. However, whether using a spinning wheel or a progress bar, a gif or JavaScript, or something more sophisticated, these simple tools create an illusion of movement, progress and activity. Depending on the implementation, progress indication graphics can significantly increase a user's perception of the speed in which an event is taking place.

Combine this with a cursor change and a lock over the element to prevent double-clicking or reloading, and your chances of keeping your user's valuable attention have significantly increased.

Demo: Progress activity and the post-active state

This same logic applies to all aspects of defaulting in a browser, from micro-elements like this up to something as simple as a 404 page. The difference in a user's reaction to hitting the default Apache 404 and a hand-crafted, branded page are phenomenal and there are no prizes for guessing which one they're more likely to exit from.

PSEUDO-CLASS PRELOADING

Another detail that it pays well to look after is the use and abuse of the `:hover` element and, more importantly, the content revealed by it. Chances are you're using the `:hover` pseudo-class somewhere in almost every screen you create. If content is being revealed on `:hover` and that content takes some time to load, there will inevitably be a delay the first time it is initiated. It appears tacky and half-finished when a tooltip or drop-down loads instantly, only to have its background or supporting elements follow through a second or two later. So, let's preload the elements we know we'll need.

A very simple application of this would be to load each file into the default state of a visible element and offset them by a large number. This ensures our elements have loaded and are ready if and when they need to be displayed.

```
element {
    background: url(path/to/image.jpg) -9999em -9999em
no-repeat;
    }
element .tooltip {
    display: none;
    }
element:hover .tooltip {
    display: block;
    background: url(path/to/image.jpg) 0 0;
    }
```

Background images are just one example. Of course, the same logic can apply to any form of revealed content. Using a sprite graphic can also be a clever — albeit tedious — method for achieving the same goal, so if you're using a sprite, preloading in this way may not be necessary

The differences between preloading and not can only be visualized properly with an actual demonstration.

Demo: Preloading revealed content

BUTTONS AND THEIR (MIS)BEHAVIOUR

Almost all of the time, a button serves just one purpose: to be clicked (or tapped). When a button's pressed, therefore, if anything other than triggering the desired event occurs, a user naturally becomes frustrated. I often get funny looks when talking about this, but designing the details of a button is something I consider essential.

It goes without saying that a button should always visually recognise `:hover` and `:active` states. We can take that one step further and disable some actions that get in the way of pressing the button.

It's rare that a user would ever want to select and use the text on a button, so let's cleanly disable it:

```
element {
    -moz-user-select: -moz-none;
    -webkit-user-select: none;
    user-select: none;
    }
```

If the button is image-based or contains an image, we could also disable user dragging to make sure the image element stays locked to the button:

```
element {
    -moz-user-drag: -moz-none;
    -webkit-user-drag: none;
    user-drag: none;
    }
```

Demo: A more usable button

Disabling global features like this should be done with utmost caution as it's very easy to cross the line between enhancement and friction. Cases where this is acceptable are very rare, but it's a good trick to keep in mind nevertheless. Both Apple's iCloud and Metalab's Flow applications use these tools appropriately and to great extent.

You could argue that the visual feedback of having the text selected or image dragged when a user mis-hits the button is actually a positive effect, informing the user that their desired action did not work. However, covering for human error should be a designer's job, not that of our users. We can (almost) ensure it *does* work for them by accommodating for errors like this in most cases.

FINAL THOUGHTS

Designing to this level of detail can seem obsessive, but as a designer and user of many interfaces and applications, I believe it can be the difference between a good user experience and a **great** one.

The samples you've just seen are only a fraction of the detail we can design for. Keep in mind that the demonstrations, code and methods above outline *just one* way to do this. You may not agree with all of these processes or have the time and desire to consider them, but one fact remains: it’s not the technology, or the *way* it’s done that’s important — it’s the logic and the concept of designing **everything**.

ABOUT THE AUTHOR

Chris Sealey is a UI designer and front-end developer with a passion for obsessively detailed graphics, clean-cut code and copious amounts of caffeine. Having worked on the web for roughly a quarter of its lifespan, he lives in Sydney and is employed full-time as a web designer/developer for Holy Cow! Design. He occasionally writes, designs and codes under the moniker of 51bits where his work, ramblings and pet projects reside.

4. Adaptive Images for Responsive Designs

Matt Wilcox 24ways.org/201104

So you’ve been building some responsive designs and you’ve been working through your checklist of things to do:

- You started with the content and designed around it, with mobile in mind first.
- You’ve gone liquid and there’s nary a px value in sight; % is your weapon of choice now.
- You’ve baked in a few media queries to adapt your layout and tweak your design at different window widths.
- You’ve made your images scale to the container width using the fluid Image technique.
- You’ve even done the same for your videos using a nifty bit of JavaScript.

You’ve done a good job so pat yourself on the back. But there’s still a problem and it’s as tricky as it is important: image resolutions.

HTML HAS AN `<IMG>` PROBLEM

CSS is great at adapting a website design to different window sizes – it allows you not only to tweak layout but also to send rescaled versions of the design's images. And you want to do that because, after all, a smartphone does not need a 1,900-pixel background image[1].

HTML is less great. In the same way that you don't want CSS background images to be larger than required, you don't want that happening with `<img>`s either. A smartphone only needs a small image but desktop users need a large one. Unfortunately `<img>`s can't adapt like CSS, so what do we do?

Well, you could just use a high resolution image and the fluid image technique would scale it down to fit the viewport; but that's sending an image five or six times the file size that's really needed, which makes it slow to download and unpleasant to use. Smartphones are pretty impressive devices – my ancient iPhone 3G is more powerful in every way than my first proper computer – but they're still terribly slow in comparison to today's desktop machines. Sending a massive image means it has to be manipulated in memory and redrawn as you scroll. You'll find phones rapidly run out of RAM and slow to a crawl.

Well, OK. You went mobile first with everything else so why not put in mobile resolution `<img>`s too? Because even though mobile devices are rapidly gaining share in your analytics stats, they're still not likely to be the major share of your user base. I don't think desktop users would be happy with pokey little mobile resolution images, do you? What we need are adaptive images.

ADAPTIVE IMAGE TECHNIQUES

There are a number of possible solutions, each with pros and cons, and it's not as simple to find a graceful solution as you might expect.

Your first thought might be to use JavaScript to trawl through the markup and rewrite the source attribute. That'll get you the right end result, but it'll have done it in a way you absolutely don't want. That's because of the way browsers load resources. It starts to load the HTML and builds the page on-the-fly; as soon as it finds an `<img>` element it immediately asks the server for that image. After the HTML has finished loading, the JavaScript will run, change the `src` attribute, and then the browser will request that new image too. Not instead of, but as well as. Not good: that's added more bloat instead of cutting it.

Plain JavaScript is out then, which is a problem, because what other tools do we have to work with as web designers? Let's ignore that for now and I'll outline another issue with the concept of serving different resolution images for different window widths: a basic file management problem. To request a different image, that image has to exist on the server. How's it going to get there? That's not a trivial problem, especially if you have non-technical users that update content through a CMS. Let's say you solve that – do you plan on a simple binary switch: big image|little image? Is that really efficient or future-proof? What happens if you have an archive of existing content that needs to behave this way? Can you apply such a solution to existing content or markup?

There's a detailed round-up of potential techniques for solving the adaptive images problem over at the Cloud Four blog if you fancy a dig around exploring all the options currently available. But I'm here to show you what I think is the most flexible and easy to implement solution, so here we are.

ADAPTIVE IMAGES

Adaptive Images aims to mitigate most of the issues surrounding the problems of bringing the venerable `<img>` tag into the 21st century. And it works by leaving that tag completely alone – just add that desktop resolution image into the markup as you've been doing for years now. We'll fix it using secret magic techniques and bottled pixie dreams. Well, fine: with one .htaccess file, one small PHP file and one line of JavaScript. But you're killing the mystique with that kind of talk.

So, what does this solution do?

- It allows `<img>`s to adapt to the same break points you use in your media queries, giving granular control in the same way you get with your CSS.
- It installs on your server in five minutes or less and after that is automatic and you don't need to do anything.
- It generates its own rescaled images on the server and doesn't require markup changes, so you can apply it to existing web content.
- If you wish, it will make all of your images go mobile-first (just in case that's what you want if JavaScript and cookies aren't available).

Sound good? I hope so. Here's what you do.

SETTING UP AND ROLLING OUT

I'll assume you have some basic server knowledge along with that wealth of front-end wisdom exploding out of your head: that you know not to overwrite any existing .htaccess file for example, and how to set file permissions on your server. Feeling up to it? Excellent.

1. Download the latest version of Adaptive Images either from the website or from the GitHub repository.

2. Upload the included .htaccess and adaptive-images.php files into the root folder of your website.
3. Create a directory called ai-cache and make sure the server can write to it (`CHMOD 755` should do it).
4. Add the following line of JavaScript into the `<head>` of your site:

```
<script>document.cookie='resolution='+Math.max(screen.width,screen.hei
path=/‘;</script>
```

That’s it, unless you want to tweak the default settings. You likely do, but essentially you’re already up and running.

HOW IT WORKS

Adaptive Images does a number of things depending on the scenario the script has to handle, but here’s a basic overview of what it does when you load a page running it:

1. A session cookie is written with the value of the visitor’s screen size in pixels.
2. The HTML encounters an `<img>` tag and sends a request to the server for that image. It also sends the cookie, because that’s how browsers work.
3. Apache sits on the server and receives the request for the image. Apache then has a look in the .htaccess file to see if there are any special instructions for files in the requested URL.

4. There are! The .htaccess says “Hey, server! Any request you get for a JPG, GIF or PNG file just send to the adaptive-images.php file instead.”
5. The PHP file then does some intelligent thinking which can cover a number of scenarios, but I’ll illustrate one path that can happen:

- The PHP file looks for the cookie and finds out that the user has a maximum screen width of 480px.
- The PHP has a look at the available media query sizes that were configured and decides which one matches the user’s device.
- It then has a look inside the /ai-cache/480/ folder to see if a rescaled image already exists there.
- We’ll pretend it doesn’t – the PHP then goes to the actual requested URI and finds that the original file does exist.
- It has a look to see how wide that image is. If it’s already smaller than the user’s screen width it sends it along and stops there. But, let’s pretend the image is 1,000px wide.
- The PHP then resizes the image and saves it into the /ai-cache/480 folder ready for the next time someone needs it.

It also does a few other things when needs arise, for example:

- It sends images with a cache header field that tells proxies not to cache the image, while telling browsers they should. This avoids problems with proxy servers and network caching systems grabbing the wrong image and storing it.
- It handles cases where there isn't a cookie set, and you can choose whether to then send the mobile version or the largest configured media query size.
- It compares timestamps between the source image and the generated cache image – to ensure that if the source image gets updated, the old cached file won't be sent.

CUSTOMIZING

There are a few options you can customize if you don't like the default values. By looking in the PHP's configuration section at the top of the file, you can:

- Set the resolution breakpoints to match your media query break points.
- Change the name and location of the ai-cache folder.
- Change the quality level any generated JPG images are saved at.
- Have it perform a subtle sharpen on rescaled images to help keep detail.
- Toggle whether you want it to compare the files in the cache folder with the source ones or not.
- Set how long the browser should cache the images for.

- Switch between a mobile-first or desktop-first approach when a cookie isn't found.

More importantly, you probably want to omit a few folders from the AI behaviour. You don't need or want it resizing the images you're using in your CSS, for example. That's fine – just open up the .htaccess file and follow the instructions to list any directories you want AI to ignore. Or, if you're a dab hand at `RewriteRules` you can remove the exclamation mark at the start of the rule and it'll only apply AI behaviour to a given list of folders.

CAVEATS

As I mentioned, I think this is one of the most flexible, future-proof, retrofittable and easy to use solutions available today. But, there are problems with this approach as there are with all of the ones I've seen so far.

This is a PHP solution

I wish I was smarter and knew some fancy modern languages the cool kids discuss at parties, but I don't. So, you need PHP on your server. That said, Adaptive Images has a Creative Commons licence[2] and I would welcome anyone to contribute a port of the code[3].

Content delivery networks

Adaptive Images relies on the server being able to: intercept requests for images; do some logic; and send one of a given number of responses. Content delivery networks are generally dumb caches, and they won't allow that to happen. Adaptive Images will not work if you're using a CDN to deliver your website.

A minor but interesting cookie issue.

As Yoav Weiss pointed out in his article Preloaders, cookies and race conditions, there is no way to guarantee that a cookie will be set before images are requested – even though the JavaScript that sets the cookie is loaded by the browser before it finds any `<img>` tags. That could mean images being requested without a cookie being available. Adaptive Images has a two-fold mechanism to avoid this being a problem:

1. The `$mobile_first` toggle allows you to choose what to send to a browser if a cookie isn't set. If `FALSE` then it will send the highest configured resolution; if `TRUE` it will send the lowest.
2. Even if set to `TRUE`, Adaptive Images checks the User Agent String. If it discovers the user is on a desktop environment, it will override `$mobile_first` and set it to `FALSE`.

This means that if `$mobile_first` is set to `TRUE` and the user was unlucky (their browser didn't write the cookie fast enough), mobile devices will be supplied with the smallest image, and desktop devices will get the largest.

The best way to get a cookie written is to use JavaScript as I've explained above, because it's the fastest way. However, for those that want it, there is a JavaScript-free method which uses CSS and a bogus PHP 'image' to set the cookie. A word of caution: because it requests an external file, this method is slower than the JavaScript one, and it is very likely that the cookie won't be set until after images have been requested.

THE FUTURE

For today, this is a pretty good solution. It works, and as it doesn't interfere with your markup or source material in any way, the process is non-destructive. If a future solution is superior, you can just remove the Adaptive Images files and you're good to go – you'd never know AI had been there.

However, this isn't really a long-term solution, not least because of the intermittent problem of the cookie and image request race condition. What we really need are a number of standardized ways to handle this in the future.

First, we could do with browsers sending far more information about the user's environment along with each HTTP request (device size, connection speed, pixel density, etc.), because the way things work now is no longer fit for purpose. The web now is a much broader entity used on far more diverse devices than when these technologies were dreamed up, and we absolutely require the server to have better knowledge about device capabilities than is currently possible. Relying on cookies to do this job doesn't cut it, and the User Agent String is a complete mess incapable of fulfilling the various purposes we are forced to hijack it for.

Secondly, we need a W3C-backed markup level solution to supply semantically different content at different resolutions, not just rescaled versions of the same content as Adaptive Images does.

I hope you've found this interesting and will find Adaptive Images useful.

FOOTNOTES

[1] While I'm talking about preventing smartphones from downloading resources they don't need: you should be careful of your media query construction if you want to stop WebKit downloading all the images in all of the CSS files.

[2] Adaptive Images has a very broad Creative Commons licence and I warmly welcome feedback and community contributions via the GitHub repository.

[3] There is a ColdFusion port of an older version of Adaptive Images. I do not have anything to do with ported versions of Adaptive Images.

ABOUT THE AUTHOR

Matt Wilcox has been a web developer for seven years and spent the last six specialising in front-end coding and design at a couple of small agencies.
He loves bouldering, photography, and a good debate. He is continually re-starting his attempts to learn Japanese.

He has a personal website at mattwilcox.net and, after five years, a burning desire to get it re-designed.

5. Collaborative Development for a Responsively Designed Web

Paul Lloyd 24ways.org/201105

In responsive web design we've found a technique that allows us to design for the web as a medium in its own right: one that presents a fluid, adaptable and ever changing canvas.

Until this point, we gave little thought to the environment in which users will experience our work, caring more about the aggregate than the individual. The applications we use encourage rigid layouts, whilst linear processes focus on clients signing off paintings of websites that have little regard for behaviour and interactions. The handover of pristine, pixel-perfect creations to developers isn't dissimilar to farting before exiting a crowded lift, leaving front-end developers scratching their heads as they fill in the inevitable gaps. If you haven't already, I recommend reading Drew's checklist of things to consider before handing over a design.

Somehow, this broken methodology has survived for the last fifteen years or so. Even the advent of web standards has had little impact. Now, as we face an onslaught of different devices, the true universality of the web can no longer be ignored.

Responsive web design is just the thin end of the wedge. Largely concerned with layout, its underlying philosophy could ignite a trend towards interfaces that adapt to any number of different variables: input methods, bandwidth availability, user preference – you name it!

With such adaptability, a collaborative and iterative process is required. Ethan Marcotte, who worked with the team behind the responsive redesign of the Boston Globe website, talked about such an approach in his book:

> The responsive projects I've worked on have had a lot of success combining design and development into one hybrid phase, bringing the two teams into one highly collaborative group.

Whilst their process still involved the creation of desktop-centric mock-ups, these were presented to the entire team early on, where questions about how pages might adapt and behave at different sizes were asked. Mock-ups were quickly converted into HTML prototypes, meaning further decisions could be based on usage rather than guesswork (and endless hours spent in Photoshop).

Regardless of the exact process, it's clear that the relationship between our two disciplines is more crucial than ever. Yet, historically, it seems a wedge has been driven between us – perhaps a result of segregation and waterfall-style processes – resulting in animosity.

So how can we improve this relationship? Ultimately, we'll need to adapt, but even within existing workflows we can start to overlap. Simply adjusting our attitude can effect change, and bring design and development teams closer together.

> Good design is constant contact.
>
> *Mark Otto*

The way we work needs to be more open and inclusive. For example, ensuring members of the development team attend initial kick-off meetings and design workshops will not only ensure technical concerns are raised, but mean that those implementing our designs better understand the problems we're trying to solve.

It can also be useful at this stage to explain how you work and the sort of deliverables you expect to produce. This will give developers a chance to make recommendations on how these can be optimized for their own needs.

You may even find opportunities to share the load. On a recent project I worked on, our development partners offered to produce the interactive prototypes needed for user testing. This allowed us to concentrate on refining the experience, whilst they were able to get a head start on building the product.

While developers should be involved at the beginning of projects, it’s also important that designers are able to review and contribute to a product as it’s being built. Any handover should be done in person, and ideally you’ll have a day set aside to do so. Having additional budget available for follow-up design reviews is also recommended. Learning how to use version control tools like Subversion or Git will allow you to work within the same environment as developers, and allow you to contribute code or graphic assets directly to a project if needed.

Don’t underestimate the benefits of designer and developer sitting next to each other. Subtle nuances can be explored far more easily than if they were conducted over email or phone. As Ethan writes, “‘Design’ is the means, not merely the end; the path we walk over the course of a project, the choices we make”.

It’s from collaboration like this that I’ve become fond of producing visual style guides. These demonstrate typographic treatments for common markup and patterns (blockquotes, lists, pagination, basic form controls and so on). Thinking in terms of components rather than individual pages not only fits in better with how a developer will implement a site, but can also ensure your design works as a coherent whole.

Despite the amount of research and design produced, when it comes to the crunch, there will always be a need for compromise. As the old saying goes, ‘fast, cheap and good – pick two.’ It’s important that you know which pieces are crucial to a design and which areas can allow for movement. Pick your battles wisely. Having an agreed set of design principles can be useful when making such decisions, as they help everyone focus on the goals of the project.

> The best compromises are reached when both sides understand the issues of the other.
>
> *Richard Rutter*

Ultimately, better collaboration comes through a shared understanding of the different competencies required to build a website. Instead of viewing ourselves in terms of discrete roles, we should instead look to emphasize our range of abilities, and work with others whose skills are complementary.

Perhaps somebody who actively seeks to broaden their knowledge is the mark of a professional. Seek these people out.

The best developers I've worked with have a respect for design, probably having attempted to do some themselves! Having wrangled with a few MySQL databases myself, I certainly believe the obverse is true. While knowing HTML won't necessarily make you a better designer, it will help you understand the issues being faced by a front-end developer and, more importantly, allow you to offer solutions or alternative approaches.

So take a moment to think about how you work with developers and how you could improve your relationship with them. What are you doing to ease the path towards our collaborative future?

ABOUT THE AUTHOR

Paul Robert Lloyd is interaction designer at the Guardian. Prior to this he was a senior designer at Clearleft, where he worked for clients such as NBCUniversal, Channel 4, Mozilla and UNICEF UK.

When not working on side projects (he is currently digitizing George Bradshaw's railway guide), Paul can be found writing about design, travel and more on his blog or blathering on Twitter.

6. Defending the Perimeter Against Web Widgets

Rich Thornett 24ways.org/201106

On July 14, 1789, citizens of Paris stormed the Bastille, igniting a revolution that toppled the French monarchy. On July 14 of this year, there was a less dramatic (though more tweeted) takedown: The Deck network, which delivers advertising to some of the most popular web design and culture destinations, was down for about thirty minutes. During this period, most partner sites running ads from The Deck could not be viewed as result.

A few partners were unaffected (aside from not having an ad to display). Fortunately, Dribbble, was one of them. In this article, I'll discuss outages like this and how to defend against them. But first, a few qualifiers: The Deck has been rock solid – this is the only downtime we've witnessed since joining in June. More importantly, the issues in play are applicable to any web widget you might add to your site to display third-party content.

DOWN AND OUT

Your defense is only as good as its weakest link. Web pages are filled with links, some of which threaten the ability of your page to load quickly and correctly. If you want your site to work when external resources fail, you need to identify the weak links on your site. In this article, we'll talk about web widgets as a point of failure and defensive JavaScript techniques for handling them.

WIDGETS 101

Imagine a widget that prints out a *Pun of the Day* on your site. A simple technique for both widget provider and consumer is for the provider to expose a URL:

```
http://widgetjonesdiary.com/punoftheday.js
```

which returns a JavaScript file like this:

```
document.write("<h2>The Pun of the Day</h2><p>Where do
frogs go for beers after work? Hoppy hour!</p>");
```

The call to `document.write()` injects the string passed into the document where it is called. So to display the widget on your page, simply add an external script tag where you want it to appear:

```
<div class="punoftheday">
  <script src="http://widgetjonesdiary.com/
punoftheday.js"></script>
  <!-- Content appears here as output of script above -->
</div>
```

This approach is incredibly easy for both provider and consumer. But there are implications…

DOCUMENT.WRITE()… OR WRONG?

As in the example above, scripts may perform a `document.write()` to inject HTML. Page rendering halts while a script is processed so any output can be inlined into the document. Therefore, page rendering speed depends on how fast the script returns the data. If an external JavaScript widget hangs, so does the page content that follows. It was this scenario that briefly stalled partner sites of The Deck last summer.

THE ELEGANT SOLUTION

To make our web widget more robust, calls to `document.write()` should be avoided. This can be achieved with a technique called JSONP (AKA JSON with padding). In our example, instead of writing inline with `document.write()`, a JSONP script passes content to a callback function:

```
publishPun("<h2>Pun of the Day</h2><p>Where do frogs go
for beers after work? Hoppy hour!</p>");
```

Then, it's up to the widget consumer to implement a callback function responsible for displaying the content. Here's a simple example where our callback uses jQuery to write the content into a target `<div>`:

```
<!-- Where widget content should appear -->
<div class="punoftheday"></div>

…
```

View Example 1

Even if the widget content appears at the top of the page, our script can be included at the bottom so it's non-blocking: a slow response leaves page rendering unaffected. It simply invokes the callback which, in turn, writes the widget content to its display destination.

THE HACK

But what to do if your provider doesn't support JSONP? This was our case with The Deck. Returning to our example, I'm reminded of computer scientist David

Wheeler's statement, "All problems in computer science can be solved by another level of indirection… Except for the problem of too many layers of indirection."

In our case, the indirection is to *move* the widget content into position *after* writing it to the page. This allows us to place the widget `<script>` tag at the bottom of the page so rendering won't be blocked, but still display the widget in the target. The strategy:

1. Load widget content into a hidden `<div>` at the bottom of the page.
2. Move the loaded content from the hidden `<div>` to its display location.

and the code:

```
<!-- Where widget content should appear -->
<div class="punoftheday"></div>
```

…

View Example 2

After the external punoftheday.js script has processed, the *rendered* HTML will look as follows:

```
<div class="loading-dock hidden">
  <script src="http://widgetjonesdiary.com/
punoftheday.js"></script>
  <h2>Pun of the Day</h2>
  <p>Where do frogs go for beers after work? Hoppy
hour!</p>
</div>
```

The ‘loading-dock’ `<div>` now includes the widget content, albeit hidden from view (if we’ve styled the ‘hidden’ class with `display: none`). There’s just one more step: move the content to its display destination. This line of jQuery (from above) does the trick:

```
$('.punoftheday').append($('.loading-dock').children(':gt(0)'));
```

This selects all child elements in the ‘loading-doc’ `<div>` *except* the first – the widget `<script>` tag which generated it – and moves it to the display destination. Worth noting is the `:gt(0)` jQuery selector extension, which allows us to exclude the first (in a 0-based array) child element – the widget `<script>` tag – from selection.

Since all of this happens at the bottom of the page, just before the `</body>` tag, no rendering has to wait on the external widget script. The only thing that fails if our widget hangs is… the widget itself. Our weakest link has been strengthened and so has our site. DE-FENSE!

ABOUT THE AUTHOR

Rich Thornett wanted to play pro basketball when he grew up, but found himself trapped in the body of a software developer. After working on his game for over a decade at software shops large and small, he created Dribbble, a show and tell site for designers. What was once a side project is now a small company where he serves as lead developer and product designer.

He lives in Salem, MA with his witty, wonderful wife and two kids, who are lost in bonkers. If you believe the pun is mightier than the sword, you can follow him on Twitter.

7. Front-end Style Guides

Anna Debenham 24ways.org/201107

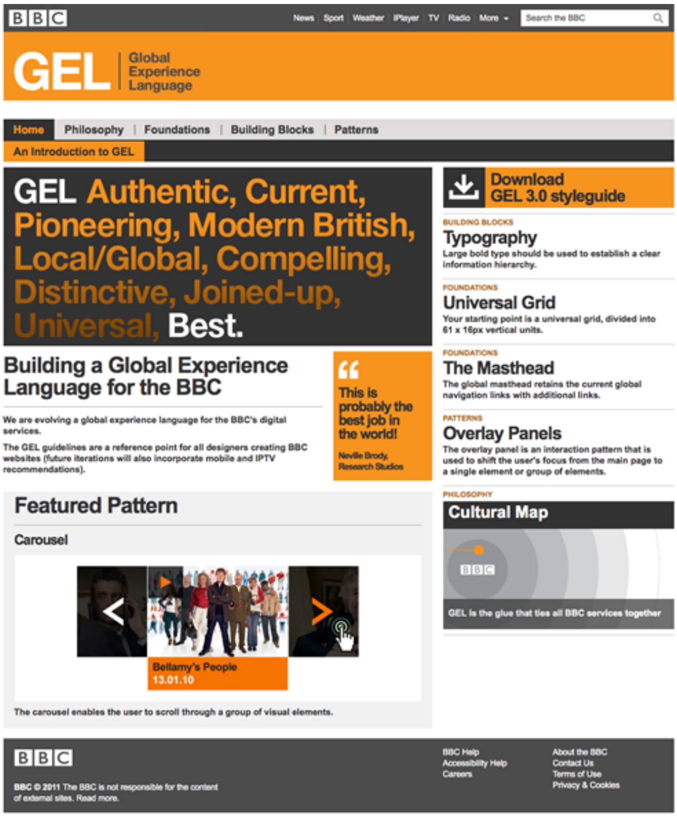
We all know that feeling: some time after we launch a site, new designers and developers come in and make adjustments. They add styles that don't fit with the content, use typefaces that make us cringe, or chuck in bloated code. But if we didn't leave behind any documentation, we can't really blame them for messing up our hard work.

To counter this problem, graphic designers are often commissioned to produce style guides as part of a rebranding project. A style guide provides details such as how much white space should surround a logo, which typefaces and colours a brand uses, along with when and where it is appropriate to use them.

DESIGN GUIDELINES

Some design guidelines focus on visual branding and identity. The UK National Health Service (NHS) refer to theirs as "brand guidelines". They help any designer create something such as a trustworthy leaflet for an NHS doctor's surgery. Similarly, Transport for London's "design standards" ensure the correct logos and typefaces are used in communications, and that they comply with the Disability Discrimination Act.

Some guidelines go further, encompassing a whole experience, from the visual branding to the messaging, and the icon sets used. The BBC calls its guidelines a "Global Experience Language" or GEL. It's essential for maintaining coherence across multiple sites under the same BBC brand.



GEL Authentic, Current, Pioneering, Modern British, Local/Global, Compelling, Distinctive, Joined-up, Universal, Best.

Building a Global Experience Language for the BBC

We are evolving a global experience language for the BBC's digital services.

The GEL guidelines are a reference point for all designers creating BBC websites (future iterations will also incorporate mobile and IPTV recommendations).

“This is probably the best job in the world!” Neville Brody, Research Studios

Featured Pattern

Carousel

The carousel enables the user to scroll through a group of visual elements.

BUILDING BLOCKS
Typography
Large bold type should be used to establish a clear information hierarchy.

FOUNDATIONS
Universal Grid
Your starting point is a universal grid, divided into 61 x 16px vertical units.

FOUNDATIONS
The Masthead
The global masthead retains the current global navigation links with additional links.

PATTERNS
Overlay Panels
The overlay panel is an interaction pattern that is used to shift the user's focus from the main page to a single element or group of elements.

PHILOSOPHY
Cultural Map
GEL is the glue that ties all BBC services together

BBC © 2011 The BBC is not responsible for the content of external sites. Read more.

The BBC's Global Experience Language.

Design guidelines may be brief and loose to promote creativity, like Mozilla's “brand toolkit”, or be precise and run to many pages to encourage greater conformity, such as Apple's “Human Interface Guidelines”.

Whatever name or form they're given, documenting reusable styles is invaluable when maintaining a brand identity over time, particularly when more than one person (who may not be a designer) is producing material.

CODE STANDARDS DOCUMENTS

We can make a similar argument for code. For example, in open source projects, where hundreds of developers are submitting code, it makes sense to set some standards. Drupal and Wordpress have written standards that make editing code less confusing for users, and more maintainable for contributors.

Each community has nuances: Drupal requests that developers indent with two spaces, while Wordpress stipulates a tab. Whatever the rules, good code standards documents also explain *why* they make their recommendations.

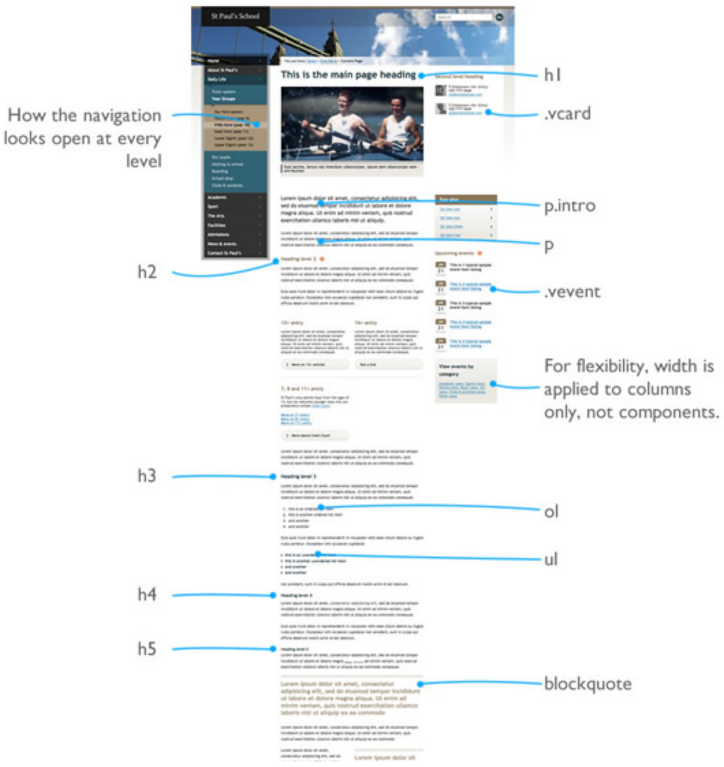
THE FRONT-END DEVELOPER'S STYLE GUIDE

Design style guides and code standards documents have been a successful way of ensuring brand and code consistency, but in between the code and the design examples, web-based style guides are emerging. These are maintained by front-end developers, and are more dynamic than visual design guidelines, documenting every component and its code on the site in one place.

Here are a few examples I've seen in the wild:

Natalie Downe's pattern portfolio

Natalie created the pattern portfolio system while working at Clearleft. The phrase describes a single HTML page containing all the site's components and styles that can act as a deliverable.



How the navigation looks open at every level
h2
h3
h4
h5
h1
.vcard
p.intro
p
.vevent
For flexibility, width is applied to columns only, not components.
ol
ul
blockquote
St Paul's School
This is the main page heading

Pattern portfolio by Natalie Downe for St Paul's School, kept up to date when new components are added. The entire page is about four times the length shown.

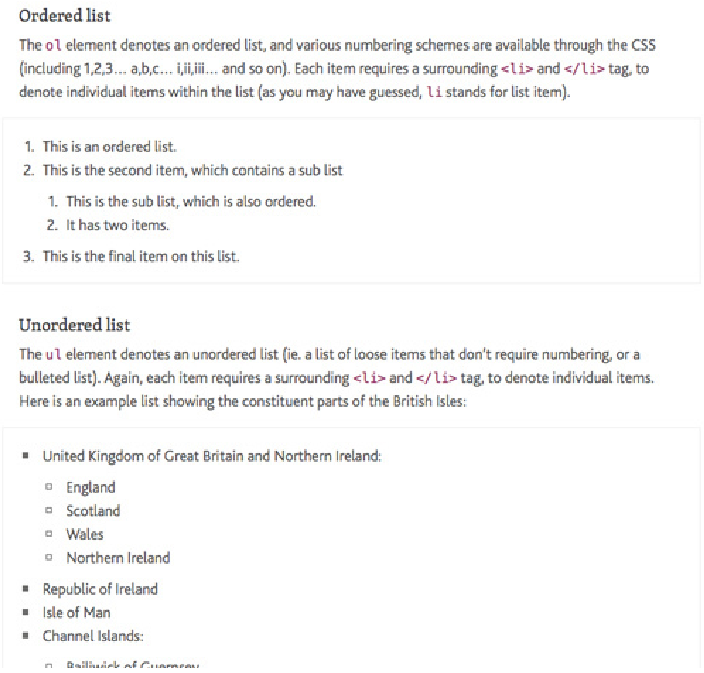
Each different item within a pattern portfolio is a building block or module. The components are decoupled from the layout, and linearized so they can slot into anywhere on a page.

> The pattern portfolio expresses every component and layout structure in the smallest number of documents. It sets out how the markup and CSS should be, and is used to illustrate the project's shared vocabulary.
>
> *Natalie Downe*

By developing a system, rather than individual pages, the result is flexible when the client wants to add more pages later on.

Paul Lloyd's style guide

Paul Lloyd has written an extremely comprehensive style guide for his site. Not only does it feature every plausible element, but it also explains in detail when it's appropriate to use each one.


Ordered list

The `ol` element denotes an ordered list, and various numbering schemes are available through the CSS (including 1,2,3... a,b,c... i,ii,iii... and so on). Each item requires a surrounding `<li>` and `</li>` tag, to denote individual items within the list (as you may have guessed, `li` stands for list item).

1. This is an ordered list.
2. This is the second item, which contains a sub list
   1. This is the sub list, which is also ordered.
   2. It has two items.
3. This is the final item on this list.

Unordered list

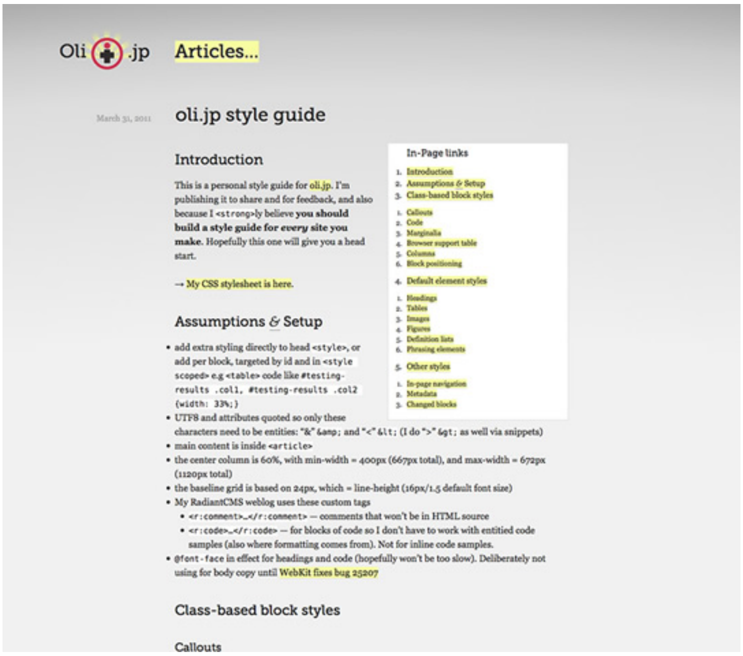
The `ul` element denotes an unordered list (ie. a list of loose items that don't require numbering, or a bulleted list). Again, each item requires a surrounding `<li>` and `</li>` tag, to denote individual items. Here is an example list showing the constituent parts of the British Isles:

- United Kingdom of Great Britain and Northern Ireland:
  - England
  - Scotland
  - Wales
  - Northern Ireland
- Republic of Ireland
- Isle of Man
- Channel Islands:
  - Bailiwick of Guernsey

Paul's style guide is also great educational material for people learning to write code.

Oli Studholme's style guide

Even though Oli's style guide is specific to his site, he's written it as though it's for someone else. It's exhaustive and gives justifications for all his decisions. In some places, he links to browser bug tickets and makes recommendations for cross-browser compatibility.



Oli.jp Articles...

March 31, 2011

oli.jp style guide

Introduction

This is a personal style guide for oli.jp. I'm publishing it to share and for feedback, and also because I `<strong>`ly believe **you should build a style guide for *every* site you make**. Hopefully this one will give you a head start.

→ My CSS stylesheet is here.

In-Page links

1. Introduction
2. Assumptions & Setup
3. Class-based block styles
   1. Callouts
   2. Code
   3. Marginalia
   4. Browser support table
   5. Columns
   6. Block positioning
4. Default element styles
   1. Headings
   2. Tables
   3. Images
   4. Figures
   5. Definition lists
   6. Phrasing elements
5. Other styles
   1. In-page navigation
   2. Metadata
   3. Changed blocks

Assumptions & Setup

- add extra styling directly to head `<style>`, or add per block, targeted by id and in `<style scoped>` e.g `<table>` code like `#testing-results .col1, #testing-results .col2 {width: 33%;}`
- UTF8 and attributes quoted so only these characters need to be entities: "&" `&amp;` and "<" `&lt;` (I do ">" `&gt;` as well via snippets)
- main content is inside `<article>`
- the center column is 60%, with min-width = 400px (667px total), and max-width = 672px (1120px total)
- the baseline grid is based on 24px, which = line-height (16px/1.5 default font size)
- My RadiantCMS weblog uses these custom tags
  - `<r:comment>…</r:comment>` — comments that won't be in HTML source
  - `<r:code>…</r:code>` — for blocks of code so I don't have to work with entitied code samples (also where formatting comes from). Not for inline code samples.
- `@font-face` in effect for headings and code (hopefully won't be too slow). Deliberately not using for body copy until WebKit fixes bug 25207

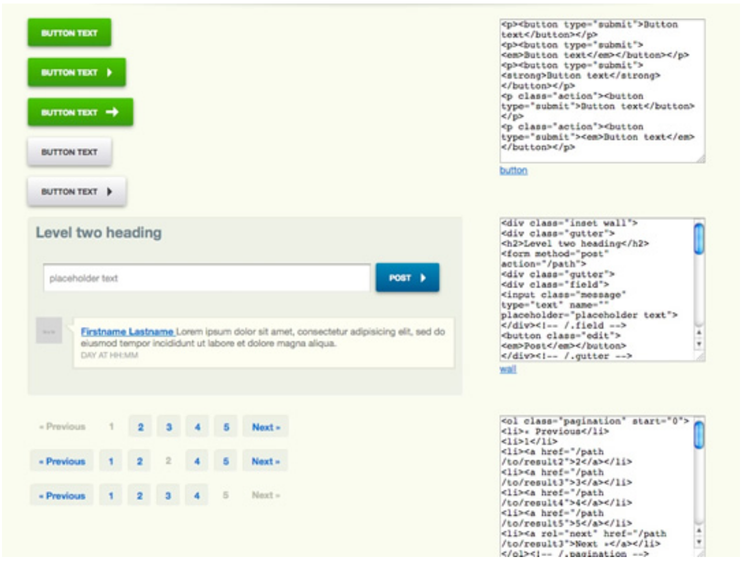
Class-based block styles

Callouts

Oli has released his style guide under a Creative Commons Attribution Share-alike license, and encourages others to create their own versions.

Jeremy Keith's pattern primer

Front-end style guides may have comments written in the code, annotations that appear on the page, or they may list components alongside their code, like Jeremy's pattern primer.



Firstname Lastname Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
DAY AT HH:MM

```
<p><button type="submit">Button
text</button></p>
<p><button type="submit">
<em>Button text</em></button></p>
<p><button type="submit">
<strong>Button text</strong>
</button></p>
<p class="action"><button
type="submit">Button text</button>
</p>
<p class="action"><button
type="submit"><em>Button text</em>
</button></p>
```

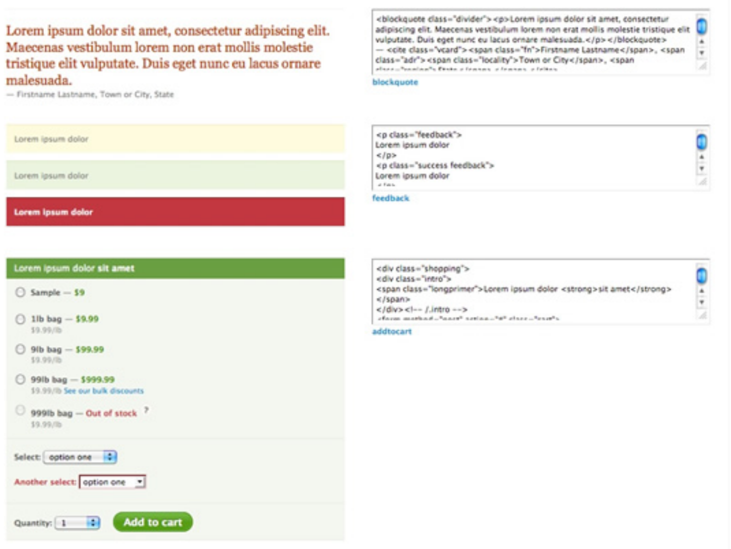
button

```
<div class="inset wall">
<div class="gutter">
<h2>Level two heading</h2>
<form method="post"
action="/path">
<div class="gutter">
<div class="field">
<input class="message"
type="text" name=""
placeholder="placeholder text">
</div><!-- /.field -->
<button class="edit">
<em>Post</em></button>
</div><!-- /.gutter -->
```

wall

```
<ol class="pagination" start="0">
<li>« Previous</li>
<li>1</li>
<li><a href="/path
/to/result2">2</a></li>
<li><a href="/path
/to/result3">3</a></li>
<li><a href="/path
/to/result4">4</a></li>
<li><a href="/path
/to/result5">5</a></li>
<li><a rel="next" href="/path
/to/result3">Next »</a></li>
</ol><!-- /.pagination -->
```

You can watch or fork Jeremy's pattern primer on Github.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas vestibulum lorem non erat mollis molestie tristique elit vulputate. Duis eget nunc eu lacus ornare malesuada.
— Firstname Lastname, Town or City, State

```
<blockquote class="divider"> <p>Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Maecenas vestibulum lorem non erat mollis molestie tristique elit
vulputate. Duis eget nunc eu lacus ornare malesuada.</p> </blockquote>
— <cite class="vcard"><span class="fn">Firstname Lastname</span>, <span
class="adr"> <span class="locality">Town or City</span>, <span
```

blockquote

Lorem ipsum dolor

Lorem ipsum dolor

Lorem ipsum dolor

```
<p class="feedback">
Lorem ipsum dolor
</p>
<p class="success feedback">
Lorem ipsum dolor
```

feedback

Lorem ipsum dolor sit amet

Sample — $9

1lb bag — $9.99
$9.99/lb

9lb bag — $99.99
$9.99/lb

99lb bag — $999.99
$9.99/lb See our bulk discounts

999lb bag — Out of stock ?
$9.99/lb

Select: option one

Another select: option one

Quantity: 1 Add to cart

```
<div class="shopping">
<div class="intro">
<span class="longprimer">Lorem ipsum dolor <strong>sit amet</strong>
</span>
</div><!-- /.intro -->
```

addtocart

Linearizing components like this resembles a kind of mobile first approach to development, which Jeremy talks about on the 5by5 podcast: The Web Ahead 3.

THE BENEFITS OF MAINTAINING A FRONT-END STYLE GUIDE

If you still need convincing that producing documentation like this for every project is worth the effort, here are a few nice side-effects to working this way:

Easier to test

A unified style guide makes it easier to spot where your design breaks. It's simple to check how components adapt to different screen widths, test for browser bugs and develop print style sheets when everything is on the same page. When I worked with Natalie, she'd resize the browser window and bump the text size up and down during development to see if anything would break.

Better workflow

The approach also forces you to think how something works in relation to the whole site, rather than just a specific page, making it easier to add more pages later on. Starting development by creating a style guide makes a lot more sense than developing on a page-by-page basis.

Shared vocabulary

Natalie's pattern portfolio in particular creates a shared vocabulary of names for components (teaser, global navigation, carousel…), so a team can refer to different regions of the site and have a shared understanding of its meaning.

Useful reference

A combined style guide also helps designers and writers to see the elements that will be incorporated in the site and, therefore, which need to be designed or populated. A boilerplate list of components for every project can act as a reminder of things that may get missed in the design, such as button states or error messages.

CREATING YOUR FRONT-END STYLE GUIDE

As you've seen, there are plenty of variations on the web style guide. Which method is best depends on your project and workflow. Let's say you want to show your content team how blockquotes and asides look, when it's appropriate to use them, and how to create them within the CMS. In this case, a combination of Jeremy's pattern primer and Paul's descriptive style guide – with the styled component alongside a code snippet and a description of when to use it – may be ideal.

Start work on your style guide as soon as you can, preferably during the design stage:

> Simply presenting flat image comps is by no means enough – it's only the start… As layouts become more adaptable, flexible and context-specific, so individual components will become the focus of our design. It is therefore essential to get the foundational aspects of our designs right, and style guides allow us to do that.
>
> *Paul Lloyd on* *Style guides for the Web*

1. Print out the designs and label the unique elements and components you'll need to add to your style guide. Make a note of the purpose of each component. While you're doing this, identify the main colours used for things like links, headings and buttons.

I draw over the print-outs on to tracing paper so I can make more annotations. Here, I've started annotating the widths from the designer's mockup so I can translate these into percentages.

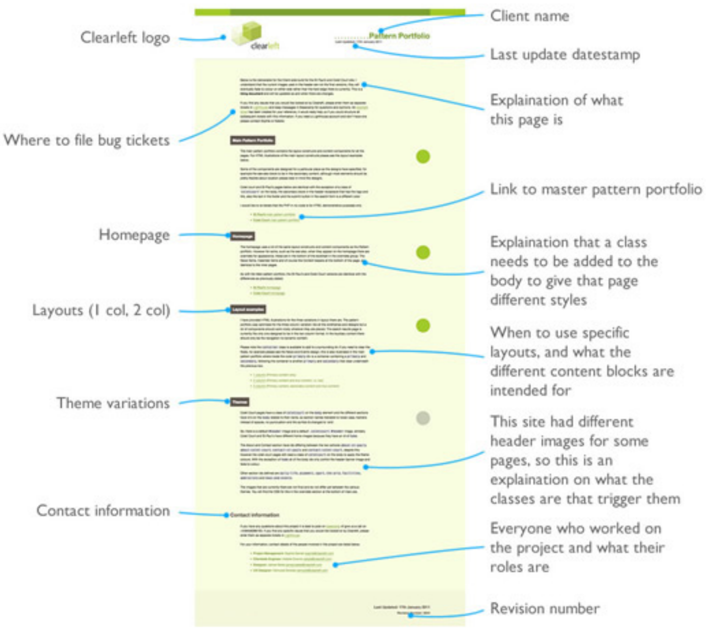
2. Start developing your style guide with base styles that target core elements: headings, links, tables, blockquotes, ordered lists, unordered lists and forms. For these elements, you could maintain a standard document to reuse for every project.
3. Next, add the components that override the base styles, like search boxes, breadcrumbs, feedback messages and blog comments. Include interaction styles, such as hover, focus and visited state on links, and hover, focus and active states on buttons.
4. Now start adding layout and begin slotting the components into place. You may want to present each layout as a separate document, or you could have them all on the same page stacked beneath one another.

Document code practices

Code can look messy when people use different conventions, so note down a standard approach alongside your style guide. For example, Paul Stanton has documented how he writes CSS.

THE GIFT WRAPPING

Presenting this documentation to your client may be a little overwhelming so, to be really helpful, create a simple page that links together all your files and explains what each of them do.


Clearleft logo
Client name
Last update datestamp
Explaination of what this page is
Where to file bug tickets
Link to master pattern portfolio
Homepage
Explaination that a class needs to be added to the body to give that page different styles
Layouts (1 col, 2 col)
When to use specific layouts, and what the different content blocks are intended for
Theme variations
This site had different header images for some pages, so this is an explaination on what the classes are that trigger them
Contact information
Everyone who worked on the project and what their roles are
Revision number

This is an example of a contents page that Clearleft produce for their clients. They've added date stamps, subversion revision numbers and written notes for each file.

Encourage participation

There's always a risk that the person you're writing the style guide for will ignore it completely, so make your documentation as user-friendly as possible. Justify why you do things a certain way to make it more approachable and encourage similar behaviour.

As always, good communication helps. Working with the designer to put together this document will improve the site. It's often not practical for designers to provide a style for everything, so drafting a web style guide and asking for feedback gives designers a chance to make sure any default styles fit in.

If you work in a team with other developers, documenting your code and development decisions will not only be useful as a deliverable, but will also force you to think about why you do things a certain way.

FUTURE-FRIENDLY

The roles of designer and developer are increasingly blurred, yet all too often we work in isolation. Working side-by-side with designers on web style guides can vastly improve the quality of our work, and the collaborative approach can spark discussions like "how would this work on a smaller screen?"

Sometimes we can be so focused on getting the site ready and live, that we lose sight of what happens after it's launched, and how it's going to be maintained. A simple web style guide can make all the difference.

If you make your own style guide, I'd love to add it to my stash of examples so please share a link to it in the comments.

ABOUT THE AUTHOR

Anna Debenham is a freelance front-end developer living in Brighton in the UK.

She's the author of Front-end Style Guides, and when she's not playing on them, she's testing as many game console browsers as she can get her hands on.

8. Adaptive Images for Responsive Designs… Again

Jake Archibald 24ways.org/201108

When I was asked to write an article for 24 ways I jumped at the chance, as I'd been wanting to write about some fun hacks for responsive images and related parsing behaviours. My heart sank a little when Matt Wilcox beat me to the subject, but it floated back up when I realized I disagreed with his method and still had something to write about.

So, Matt Wilcox, if that is your real name (and I'm pretty sure it is), **I disagree**. I see your dirty server-based hack and raise you an *even dirtier* client-side hack. Evil laugh, etc., etc.

You guys can stomach yet another article about responsive design, right? Right?

Half the room gets up to leave

Whoa, whoa… OK, I'll cut to the chase…

TL;DR

In a previous episode, we were introduced to Debbie and her responsive cat poetry page. Well, now she's added some reviews of cat videos and some images of cats. Check out her new page and have a play around with the browser window. At smaller widths, the images change and the design responds. The benefits of this method are:

- it's entirely client-side
- images are still shown to users without JavaScript
- your media queries stay in your CSS file
- no repetition of image URLs
- no extra downloads per image
- it's fast enough to work on resize
- it's pure filth

WHAT'S WRONG WITH THE SERVER-SIDE SOLUTION?

Responsive design is a client-side issue; involving the server creates a boatload of problems.

- It sets a cookie at the top of the page which is read in subsequent requests. However, the cookie is not guaranteed to be set in time for requests on the same page, so the server may see an old value or no value at all.
- Serving images via server scripts is much slower than plain old static hosting.
- The URL can only cache with `vary: cookie`, so the cache breaks when the cookie changes, even if the change is unrelated. Also, far-future caching is out for devices that can change width.
- It depends on detecting screen width, which is rather messy on mobile devices.
- Responding to things other than screen width (such as DPI) means packing more information into the cookie, and a more complicated script at the top of each page.

SO, WHY ISN'T THIS STRAIGHTFORWARD ON THE CLIENT?

Client-side solutions to the problem involve JavaScript testing user agent properties (such as screen width), looping through some images and setting their URLs accordingly. However, by the time JavaScript has sprung into action, the original image source has already started downloading. If you change the source of an image via JavaScript, you're setting off yet another request.

Images are downloaded as soon as their DOM node is created. They don't need to be visible, they don't need to be in the document.

```
new Image().src = url
```

The above will start an HTTP request for `url`. This is a handy trick for quick requests and preloading, but also shows the browser's eagerness to download images.

Here's an example of that in action. Check out the network tab in Web Inspector (other non-WebKit development aids are available) to see the image requests.

Because of this, some client-side solutions look like this:

```
<img src="t.gif" data-src="real-image.jpg"
data-bigger-src="real-bigger-image.jpg">
```

where t.gif is a 1×1px tiny transparent GIF.

This results in no images if JavaScript isn't available. Dealing with the absence of JavaScript is still important, even on mobile. I was recently asked to make a website work on an old Blackberry 9000. I was able to get most of the way there by preventing that OS parsing any JavaScript, and that was only possible because the site didn't depend on it.

We need to delay loading images for JavaScript users, but ensure they load for users without JavaScript. How can we conditionally parse markup depending on JavaScript support?

OH YEAH! `<NOSCRIPT>`!

```
<noscript>
    <img src="image.jpg">
</noscript>
```

Whoa! First spacer GIFs and now `<noscript>`? This really is the future! The image above will only load for users without JavaScript support. Now all we need to do is send JavaScript in there to get the `textContent` of the `<noscript>` element, then we can alter the image source before handing it to the DOM for parsing.

Here's an example of that working … unless you're using Internet Explorer.

Internet Explorer doesn't retain the content of `<noscript>` elements. As soon as it's parsed, it considers it an empty element. **FANKS INTERNET EXPLORER**. This is why some solutions do this:

```
<noscript data-src="image.jpg">
    <img src="image.jpg">
</noscript>
```

so JavaScript can still get at the URL via the `data-src` attribute. However, repeating stuff isn't great. Surely we can do better than that.

A DIRTY, DIRTY HACK

Thankfully, I managed to come up with a solution, and by me, I mean someone cleverer than me. Pornel's solution uses `<noscript>`, but surely we don't need that.

Now, before we look at this, I can't stress how dirty it is. It's so dirty that if you've seen it, schools will refuse to employ you.

```
<script>document.write('<' + '!--')</script>
<img src="image.jpg">
<!---->
```

Phwoar! Dirty, isn't it? I'll stop for a moment, so you can go have a wash.

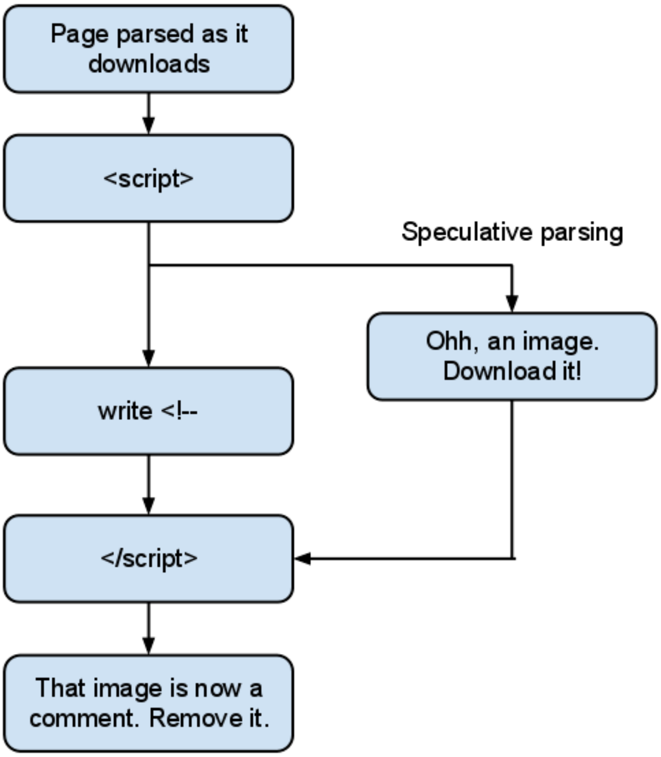
Done? Excellent.

With this, the image is wrapped in a comment *only* for users with JavaScript. Without JavaScript, we get the image. Unlike the `<noscript>` example above, we can get the text content of the comment pretty easily.

Hurrah! But wait… Some browsers are *sometimes* downloading the image, even with JavaScript enabled. Notably Firefox. Huh?

IMAGES ARE DOWNLOADED IN COMMENTS NOW? WHAT?

No. What we're seeing here is the effect of speculative parsing. Here's what's happening:


Page parsed as it downloads
<script>
Speculative parsing
Ohh, an image. Download it!
write <!--
</script>
That image is now a comment. Remove it.

While the browser is parsing the script, it parses the rest of the document. This is usually a good thing, as it can download subsequent images and scripts without waiting for the script to complete. The problem here is we create an unbalanced tree.

An unbalanced tree, yesterday.

In this case, the browser must throw away its speculative parsing and reparse from the end of the `<script>` element, taking our `document.write` into consideration. Unfortunately, by this stage it may have already discovered the image and sent an HTTP request for it.

A DIRTY, DIRTY HACK… THAT WORKS

Pornel was right: we still need the `<noscript>` element to cater for browsers with speculative parsing.

```
<script>document.write('<' + '!--')</script><noscript>
    <img src="image.jpg">
</noscript -->
```

And there we have it. We can now prevent images loading for users with JavaScript, but we can still get at the markup.

We're still creating an unbalanced tree and there's a performance impact in that. However, the parser won't have got far by the time our script executes, so the impact is small. Unbalanced trees are more of a concern for external scripts; a lot of parsing can happen by the time the script has downloaded *and* parsed.

USING DIRTINESS TO CREATE RESPONSIVE IMAGES

Now all we need to do is give each of our dirty scripts a class name, then JavaScript can pick them up, grab the markup from the comment and decide what to do with the images.

This technique isn't exclusively useful for responsive images. It could also be used to delay images loading until they've scrolled into view. But to do that you'll need a bulletproof way of detecting when elements are in view. This involves getting the height of the viewport, which is extremely unreliable on mobile devices.

Here’s a hastily thrown together example showing how it can be used for responsive images.

I adjust the end of the image URLs conditionally depending on the result of media queries. This is done on page load, and on resize.

I’m using regular expressions to alter the URLs. Using regex to deal with HTML is usually a sign of insanity, but parsing it with the browser’s DOM parser would trigger the download of images before we change the URLs. My implementation currently requires double-quoted image URLs, because I’m lazy. Wanna fight about it?

MEDIA QUERYING VIA JAVASCRIPT

Jeremy Keith used `document.documentElement.clientWidth` in his example, which is great as a proof of concept, but unfortunately is rather unreliable across mobile devices.

Thankfully, standards are coming to the rescue with window.matchMedia, which lets us provide a media query string and get a boolean result. There’s even a great polyfill for browsers that don’t support it (as long as they support media queries in CSS).

I didn’t go with that for three reasons:

1. I’d like to keep media queries in the CSS file, if possible.

2. I wanted something a little lighter to keep things speedy while resizing.
3. It's just not dirty enough yet.

To make things ultra-dirty, I add a test element to the page with a specific class, let's say `media-test`. Then, I control the width of it using media queries in my CSS file:

```
@media all and (min-width: 640px) {
    .media-test {
        width: 1px;
    }
}
@media all and (min-width: 926px) {
    .media-test {
        width: 2px;
    }
}
```

The JavaScript part changes the URL suffix depending on the width of `media-test`. I'm using a `min-width` media query, but you can use others such as `pixel-ratio` to detect high DPI displays. Basically, it's a hacky way for CSS to set a value that can be picked up by JavaScript. It means the bit that signals changes to the images sits with the rest of the responsive code, without duplication.

Also, phwoar, dirty!

THE API

I threw a script together to demonstrate the technique. I’m not particularly attached to it, I’m not even sure I like it, but here’s the API:

```
responsiveGallery({
    // Class name of dirty script element(s) to target
    scriptClass: 'dirty-gallery-script',
    // Class name for our test element
    testClass: 'dirty-gallery-test',
    // The initial suffix of URLs, the bit that changes.
    initialSuffix: '-mobile.jpg',
    // A map of suffixes, for each width of
'dirty-gallery-test'
    suffixes: {
        '1': '-desktop.jpg',
        '2': '-large-desktop.jpg',
        '3': '-mobile-retina.jpg'
    }
});
```

The API can cover individual images or multiple galleries at once. In the example I gave at the start of the article I make two calls to the API, one for both galleries, and one for the single image above the video reviews. They’re separate calls because they respond slightly differently.

THE FUTURE

Hopefully, we'll get a proper solution to this soon. My favourite suggestion is the `<picture>` element that Bruce Lawson covers.

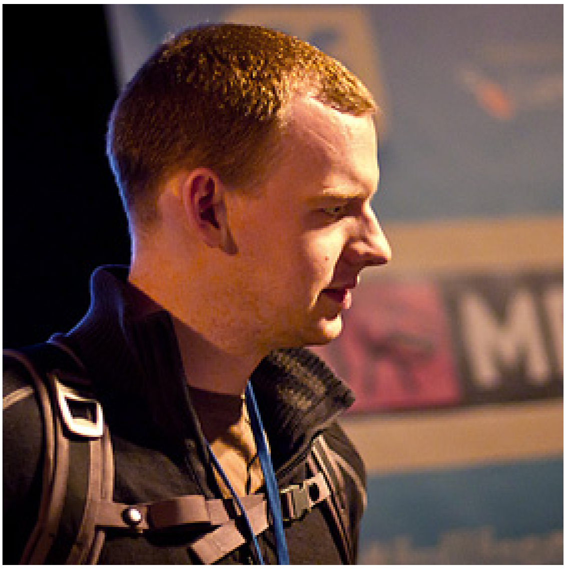
```
<picture alt="Angry pirate">
    <source src="hires.png" media="min-width:800px">
    <source src="midres.png" media="min-width:480px">
    <source src="lores.png">
    <!-- fallback for browsers without support -->
    <img src="midres.png" alt="Angry pirate">
</picture>
```

Unfortunately, we're nowhere near that yet, and I'd still rather have my media queries stay in CSS. Perhaps the source elements could be skipped if they're `display:none`; then they could have class names and be controlled via CSS. *Sigh*.

Well, I'm tired of writing now and I'm sure you're tired of reading. I realize what I've presented is a yet another dirty hack to the responsive image problem (perhaps the dirtiest?) and may be completely unfeasible in professional situations. But isn't that the true spirit of Christmas?

No.

ABOUT THE AUTHOR



Jake Archibald is a developer at Lanyrd specialising in client-side stuff, although dabbles in a bit of Django. He developed Sprite Cow to help ease the pain of sprite sheets, and started a blog way after blogs stopped being cool.

Outside of the web, he's a Formula One fan and likes taking photos of things. He tweets as @jaffathecake.

9. Composing the New Canon: Music, Harmony, Proportion

Owen Gregory 24ways.org/201109

> Ohne Musik wäre das Leben ein Irrtum
>
> —Friedrich NIETZSCHE, *Götzen-Dämmerung, Sprüche und Pfeile 33*, 1889

Somehow, music is hardcoded in human beings. It is something we understand and respond to without prior knowledge. Music exercises the emotions and our imaginative reflex, not just our hearing. It behaves so much like our emotions that music can seem to symbolize them, to bear them from one person to another. Not surprisingly, it conjures memories: the word music derives from Greek μουσική (mousike), art of the Muses, whose mythological mother was Mnemosyne, memory. But it can also summon up the blood, console the bereaved, inspire fanaticism, bolster governments and dissenters alike, help us learn, and make web designers dance. And what would Christmas be without music?

Music moves us, often in ways we can't explain. By some kind of alchemy, music frees us from the elaborate nuisance and inadequacy of words. Across the world and throughout recorded history – and no doubt well before that – people have listened and made (and made out to) music.

> [I]t appears probable that the progenitors of man, either the males or females or both sexes, before acquiring the power of expressing their mutual love in articulate language, endeavoured to charm each other with musical notes and rhythm.
>
> —Charles DARWIN, *The Descent of Man, and Selection in Relation to Sex*, 1871

It's so integral to humankind, we've sent it into space as a totem for who we are. (Who knows? It might be important.) Music is essential, a universal compulsion; as Nietzsche wrote, without music life would be a mistake.

MUSIC, DESIGN AND WEB DESIGN

There are some obvious and notable similarities between music and visual design. Both can convey mood and evoke emotion but, even under close scrutiny, how they do that remains to a great extent mysterious. Each has formal qualities or parts that can be abstracted, analysed and discussed, often using the same terminology: composition, harmony, rhythm, repetition, form, theme; even colour, texture and tone.

A possible reason for these shared aspects is that both visual design and music are means to connect with people in deep and lasting ways. Furthermore, I believe the connections to be made can complement direct emotional appeal. Certain aesthetic qualities in music work on an unconscious and, it could be argued, universal level. Using musical principles in our designs, then, can help provide the connectedness between content, device and user that we now seek as web designers.

Yet, when we talk about music and web design, the conversation is almost always about the music designers listen to while working, a theme finding its apotheosis in Designers.MX. Sometimes, articles in that dreary list format seek inspiration from music industry websites. There's even a service offering pre-templated web designs for bands, and at least one book surveyed the landscape back in 2006. Occasionally, discussions broaden somewhat into whether and how different kinds of music can inspire and influence the design work we produce.

Such enquiries, it seems to me, are beside the point. Do I really design differently when I listen to Bach rather than Bacharach? Will the barely restrained energy of Count

Basie's *The Kid from Red Bank* mean I choose a lively colour palette, and rural, autumnal shades when inspired by Fleet Foxes? Mahler means a thirteen-column layout? Gillian Welch leads to distressed black and white photography? While reflecting the importance we place in music and how it seems to help us in our work, surveys on musical taste and lists of favourite artists fail to recognize that some of the fundamental aesthetic characteristics of music can be adapted and incorporated into modern web design.

ANTIPHONAL GEOMETRY

Over recent years, web designers have embraced grid systems as powerful tools to help create good-looking and intuitive user experiences. With the advent of responsive design, these grids and their contents must adapt to the different screen sizes and properties of all kinds of user devices. Finding and using grid values that can scale well and retain or enhance their proportions and relationships while making the user experience meaningful in several different contexts is more important than ever.

In print, this challenge has always started with the dimensions and proportions of the page. Content can thereby be made to belong inside the page and be bound to it. And music has been used for centuries to further this aim. As Robert Bringhurst says in *The Elements of Typographical Style*:

> Indeed, one of the simplest of all systems of page proportions is based on the familiar intervals of the diatonic scale. Pages that embody these basic musical proportions have been in common use in Europe for more than a thousand years.

Very well. But while he goes on to list (from the full chromatic scale, rather than just diatonic) the proportions and the musical intervals they're based on, Bringhurst fails to mention what they're ratios of or their potential effects. Shame. In his favour, however, he later touches on how proportions in print might be considered to work:

> The page is a piece of paper. It is also a visible and tangible proportion, silently sounding the thoroughbass of the book. On it lies the textblock, which must answer to the page. The two together – page and textblock – produce an antiphonal geometry. That geometry alone can bond the reader to the book. Or conversely, it can put the reader to sleep, or put the reader's nerves on edge, or drive the reader away.

So what does Bringhurst mean by *antiphonal geometry*, a phrase that marries the musical to the spatial? By stating that the textblock "must answer to the page", the implication is that the relationship between the proportions of the page and the shape of the textblock printed on it embodies a spatial (geometrical) call-and-response (antiphony) that can be appealing or not.

BOULTON'S NEW CANON

But, as Mark Boulton has pointed out, on the web "*there are no edges*. There are no 'pages'. We've made them up." So, what is to be done? In January 2011 at the New Adventures in Web Design conference, Boulton presented his vision of a new canon of web design, a set of principles to guide us as we design the web. There are three overlapping tenets:

1. design from the content out
2. create connectedness between the different content elements
3. bind the content to the web device

Rather than design from the edges in, we need to design layout systems from the content out. To this end, Boulton asserts that grid system design should begin with a constraint, and he suggests we use the size of a fixed content element, such as an advertising unit or image, as a starting point for online grid calculations. Khoi Vinh advocates the same approach in his book, *Ordering Disorder: Grid Principles for Web Design*.

Boulton's second and third tenets, however, are more complex and overlap significantly with each other. Connecting the different parts of the content and binding the content to the device share many characteristics and solutions:

- adopting ems and percentages as units of size for layout elements
- altering text size, line length and line height for different viewport dimensions
- providing higher resolution images for devices with greater pixel densities
- fluid layout grids, flexible images and responsive design

All can help relate the presentation of the content to its delivery in a certain context.

But how do we determine the relationship between one element of a layout and another? How can we avoid making arbitrary decisions about the relative sizes of parts of our designs? What can we use to connect the parts of our design to one another, and how can we bind the presentation of the content to the user's device?

Tim Brown's application of modular typographic scales hints at an answer. In the very useful tool he created for calculating such scales, Brown includes two musical ratios: the perfect fifth (2:3); and the perfect fourth (3:4). Why? Where do they come from? And what do they mean?

HARMONIES MUSICAL AND VISUAL

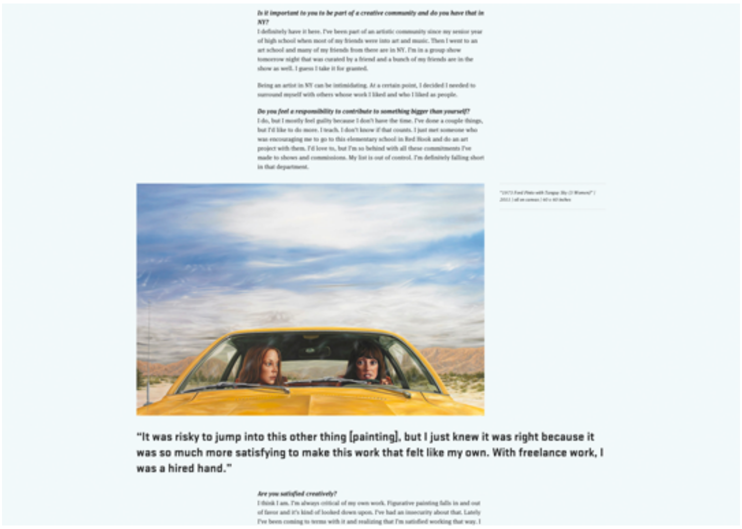
Fundamental to music are rhythm and harmony.

As any drummer will tell you, without rhythm there is no music. Even when there's no regular beat, any tune follows a rhythm, however irregular, simply because a change of note is a point of change in the music. Although rhythm, timing and pacing are all relevant to interaction design, right now it's harmony we're interested in.

Sometimes harmony is called the vertical aspect of music, and melody the horizontal. But this conceit overlooks the fact that harmony is both vertical and horizontal. A single melodic line, as it is played, implies various sets of harmonies on which it is grounded, whether or not those harmonies are played. So, harmony doesn't just sit vertically beneath the horizontal melody, but moves horizontally as well, through harmonic progression.

To stretch this arrangement pixel-thin, we could argue that in onscreen design melody is the content, and the layout and arrangement of the content is the harmony. We sometimes say a design is harmonious when the interplay of different elements of a design is pleasing or balanced or in proportion, and the content (the melody) is set off or conveyed well by or appropriate to the design.

We seem to know instinctively whether a layout is harmonious…



Is it important to you to be part of a creative community and do you have that in NY?
I definitely have it here. I've been part of an artistic community since my senior year of high school when most of my friends were into art and music. Then I went to an art school and many of my friends from there are in NY. I'm in a group show tomorrow night that was curated by a friend and a bunch of my friends are in the show as well. I guess I take it for granted.

Being an artist in NY can be intimidating. At a certain point, I decided I needed to surround myself with others whose work I liked and who I liked as people.

Do you feel a responsibility to contribute to something bigger than yourself?
I do, but I mostly feel guilty because I don't have the time. I've done a couple things, but I'd like to do more. I teach. I don't know if that counts. I just met someone who was encouraging me to go to this elementary school in Red Hook and do an art project with them. I'd love to, but I'm so behind with all these commitments I've made to shows and commissions. My list is out of control. I'm definitely falling short in that department.

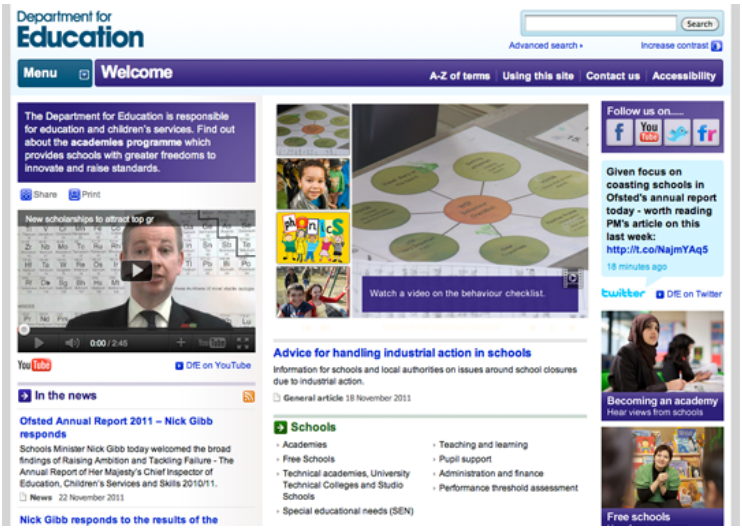
[illegible]

"It was risky to jump into this other thing [painting], but I just knew it was right because it was so much more satisfying to make this work that felt like my own. With freelance work, I was a hired hand."

Are you satisfied creatively?
I think I am. I'm always critical of my own work. Figurative painting falls in and out of favor and it's kind of looked down upon. I've had an insecurity about that. Lately I've been coming to terms with it and realizing that I'm satisfied working that way. I

In the design of The Great Discontent, the relationships between different elements combine to form a balanced whole.

…or not.


Department for Education

Search
Advanced search »
Increase contrast

Menu
Welcome
A-Z of terms | Using this site | Contact us | Accessibility

The Department for Education is responsible for education and children's services. Find out about the **academies programme** which provides schools with greater freedoms to innovate and raise standards.

Share Print

New scholarships to attract top gr

0:00 / 2:45

DfE on YouTube

In the news

Ofsted Annual Report 2011 – Nick Gibb responds
Schools Minister Nick Gibb today welcomed the broad findings of Raising Ambition and Tackling Failure - The Annual Report of Her Majesty's Chief Inspector of Education, Children's Services and Skills 2010/11.
News 22 November 2011

Nick Gibb responds to the results of the

Watch a video on the behaviour checklist.

Advice for handling industrial action in schools
Information for schools and local authorities on issues around school closures due to industrial action.
General article 18 November 2011

Schools
- Academies
- Free Schools
- Technical academies, University Technical Colleges and Studio Schools
- Special educational needs (SEN)
- Teaching and learning
- Pupil support
- Administration and finance
- Performance threshold assessment

Follow us on.....

Given focus on coasting schools in Ofsted's annual report today - worth reading PM's article on this last week: http://t.co/NajmYAq5
18 minutes ago

twitter DfE on Twitter

Becoming an academy
Hear views from schools

Free schools

There's no harmony in the Department for Education's website because the different parts of the content don't feel related to one another.

What is it that makes one design harmonious and another dissonant? It's not just whether things line up, though that's a start. I believe there are much deeper aesthetic forces at work, forces we can tap into in our onscreen designs. Now, I'm not going start a difficult discussion about aesthetics. For our purposes, we just need to know that it's the branch of philosophy dealing with the nature of beauty, and the creation and perception of beauty. And among the key components in the perception of beauty are harmony and proportion. These have been part of traditional western aesthetics since Plato (about 2,500 years).

One of the ways we appreciate the beauty of music is through the harmonic intervals we hear. A musical interval is a combination of two notes and it describes the distance between the two pitches. For example, the distance between C and the G above it (if we take C as the tonic or root) is called a perfect fifth.

Left: C to G, a perfect fifth. Right: C and G, not a perfect fifth.

And, to get superficially scientific for a moment, each musical interval can be expressed as a ratio of the wavelength frequencies of the notes; for our perfect fifth, with every two wavelengths of C, there are three of G. And what is a ratio, if not a proportion of one thing to another?

So, simple musical harmony (using what's known as *just intonation*[1]) affords us a set of proportions, expressed as ratios. Where better to apply these ideas of harmony and proportion from music in web design, than grid systems?

A digression: whither φ?

Quite often in our discussions of pure design and aesthetics, we mention the golden ratio and regurgitate the same justifications for its use: roots in antiquity; embodied in classical and Renaissance architecture and art; occurrence in nature; the New Twitter, and so forth (oh, really?).

Yet the ratios of musical intervals from just intonation are equally venerable and much more widespread: described by Pythagorus; employed in Palladian architecture, and printing, books and art from the Renaissance onwards; in modern times, film and television dimensions; standard international paper sizes (ISO 216, the A and B series); and, again and again, screen dimensions – chances are that screen you're probably looking at right now has the proportions 2:3 (iPhone and iPod Touch), 3:4 (iPad and Kindle), 3:5 (many smartphones), 5:8 or 16:9 (many widescreen monitors), all ratios of musical intervals.

Back to our theme…

MUSICAL INTERVAL RATIOS

Let’s take a look at most of the ratios within a couple of octaves and crunch some numbers to generate some percentages and other values that we can use in our designs. First, the intervals and their ratios in just intonation and expressed as ratios of one:

Name	Interval in C	Ratio	Ratio (1:x)
unison	C→C	1:1	1:1
minor second	C→D♭	15:16	1:1.067
major second	C→D	8:9	1:1.125
minor third	C→E♭	5:6	1:1.2
major third	C→E	4:5	1:1.25
perfect fourth	C→F	3:4	1:1.333
augmented fourth or diminished fifth	C→F♯/G♭	1:√2	1:1.414
perfect fifth	C→G	2:3	1:1.5
minor sixth	C→A♭	5:8	1:1.6
major sixth	C→A	3:5	1:1.667
minor seventh	C→B♭	9:16	1:1.778
major seventh	C→B	8:15	1:1.875
octave	C→C↑	1:2	1:2
major tenth	C→E↑	2:5	1:2.5
major eleventh	C→F↑	3:8	1:2.667
major twelfth	C→G↑	1:3	1:3
double octave	C→C↑	1:4	1:4
Name	**Interval in C**	**Ratio**	**Ratio (1:x)**

And now as percentages, of both the larger and smaller values in the ratios:

Name	Ratio	% of larger value	% of smaller value
unison	1:1	100%	100%
minor second	15:16	93.75%	106.667%
major second	8:9	88.889%	112.5%
minor third	5:6	83.333%	120%
major third	4:5	80%	125%
perfect fourth	3:4	75%	133.333%
augmented fourth or diminished fifth	1:√2	70.711%	141.421%
perfect fifth	2:3	66.667%	150%
minor sixth	5:8	62.5%	160%
major sixth	3:5	60%	166.667%
minor seventh	9:16	56.25%	177.778%
major seventh	8:15	53.333%	187.5%
octave	1:2	50%	200%
major tenth	2:5	40%	250%
major eleventh	3:8	37.5%	266.667%
major twelfth	1:3	33.333%	300%
double octave	1:4	25%	400%
Name	Ratio	% of larger value	% of smaller value

As you can see, the simple musical intervals are expressed as ratios of small whole numbers (integers). We can then normalize them as ratios of one, as well as derive percentage values, both in terms of the smaller value to the larger, and vice versa. These are the numbers we can incorporate into our designs. If you've ever written something like `body { font: 100%/1.5 "Museo Sans", Helvetica, sans-serif; }` in your CSS, you're already using a musical ratio: the perfect fifth.

Modular scales allow us to generate a set of numbers based on a musical interval that can be used for all kinds of typographic and layout decisions to create harmony in a visual design for the web. As Tim Brown said at the 2010 Build conference:

> I think that from that most atomic unit – type – whole experiences can resonate, whole experiences can be harmonious. And whole experiences can have a purpose suited to our design intentions.

ONCE MORE, WITH FEELING: CONNECTEDNESS

As well as modular scales, there are other methods of incorporating musical interval ratios into our work. Setting the ratio of font size to line height in CSS is one such example. We could also create a typographic hierarchy using the same principle and combining several ratios that we know harmonize well musically in a chord:

```
body { font-size: 75%; } /* =12px = base size or tonic */

h1 { font-size: 32px; font-size: 2.667rem; }
   /* =32px = 3:8 = major eleventh (C→F↑) */

h2 { font-size: 24px; font-size: 2rem; }
   /* =24px = 1:2 = octave (C→C↑) */

h3 { font-size: 20px; font-size: 1.667rem; }
   /* =20px = 3:5 = major sixth (C→A) */

figcaption, small { font-size: 9px; font-size : 0.75rem }
   /* =9px = 3:4 = perfect fourth (C→F) */
```

Whoa! Hold your reindeer, Santa! How can we know what interval combinations work well together to form chords? Well, I'm a classically trained musician, so perhaps I have an advantage. To avoid a long, technically complex digression into musical harmony, here are a few basic combinations of intervals that are harmonious in one way or another:

- unison; major third; perfect fifth; octave
- unison; perfect fourth; major sixth; octave
- unison; minor third; minor sixth; octave
- unison; minor third; diminished fifth; major sixth; octave

This isn't to say that other combinations can't be used to interesting effect and particular purpose – they surely can – but I have to make sure there's something left for you to experiment with in the wee small hours over the holiday. Bear in mind, though, were I to play you two notes from the same scale to form a minor second, for example, you'd probably say it was dissonant and maybe that quality of the 15:16 ratio would be translated to the design.

In the typographic hierarchy above, you'll notice I used an interval in the higher octave, which affords a broader range of ratios while retaining the harmony. Thus, a perfect fifth (2:3) becomes a major twelfth (1:3), or a major sixth (3:5) becomes a major thirteenth (3:10).

The harmonic ratios can obviously be used as proportions for layout as well, in several different ways:

- image width and height (for example, 450×800px = 9:16 = minor seventh)
- main content to page width (67%:100% = 2:3 = perfect fifth)
- page width to viewport width (80%:100% = 4:5 = major third)

One great benefit of using such ratios in web design work is that they can be applied in responsive web design. Proportional values, based on percentages or equivalent em units, will scale with changing viewports, so your layout and image proportions can be maintained or deliberately changed, as we're about to find out, across devices.

SMALL SPEAKERS, TALL SPEAKERS: BINDING TO THE DEVICE

The musical interval ratios also provide an opportunity, not only to create connectedness between the parts of a layout, but to bind the content to a device – that elusive *antiphonal geometry*. Just as a textblock and page resonate together, so too can web content and the screen. Earlier, I mentioned that several common screen aspect ratios match musical interval ratios. It would seem, then, that we have a set of proportions that we can use in different ways to establish and retain a sense of harmony that can be based on and change with those contexts. Using musical interval ratios, we can bind the display of our content to the device it's displayed on.

If you haven't met already, let me introduce you to the `device-aspect-ratio` property of CSS media queries.

```
@media only screen and (device-aspect-ratio: 3/4) { }
@media only screen and (device-aspect-ratio: 480/640) { }
@media only screen and (device-aspect-ratio: 600/800) { }
@media only screen and (device-aspect-ratio: 768/1024) {
}
```

Regardless of the precise pixel values, each of these media queries would apply to devices whose display area has an aspect ratio of 3:4. It works by comparing the `device-width` with the `device-height`. (It's not to be confused with `aspect-ratio`, which is defined by the `width` and `height` of the browser within the device.) The values in the media query are always presented as width/height, with portrait being the default orientation for smartphones and tablets; that is, to match an iPhone screen, you'd use `device-aspect-ratio: 2/3`, not `3/2`, which won't work.

Here's a table of the musical intervals with their corresponding screens.

Name	`device-aspect-ratio`	Screens	Common resolutions (pixels)
major third	5/4	TFT LCD computer screens	1,280×1,024
perfect fourth	3/4 or 4/3	iPad, Kindle and other tablets, PDAs	320×240, 768×1,024
perfect fifth	2/3	iPhone, iPod Touch	320×480, 640×960
minor sixth	8/5 (16/10)	Many widescreens	1,152×720, 1,440×900, 1,920×1,200
major sixth	3/5	Many smartphones	240×400, 480×800
minor seventh	16/9 or 9/16	Many widescreens and some smartphones	720×1,280, 1,366×768, 1,920×1,080, 2,560×1,440

[You might argue that I'm playing fast and loose with the ratios. I suppose, mathematically speaking, 9:16 is not the same as 16:9: I'm no expert. But let's not throw the baby out with the bath water, particularly at Christmas.]

With this in mind, we can begin to write media queries that will influence various typographic and layout values in line with the aspect ratios of specific screens and in combination with em-based `min-width` queries that work from smaller, mobile screens to larger, desktop screens.

Here's a very simple demo page with only some text, an image with a caption and a little basic layout: no seasonal overindulgence here.

Demo: Sample page using `device-aspect-ratio` media queries based on musical interval ratios

Our initial styles for all devices are based on the perfect fifth, with the major third and octave rounding things out into a harmonious whole, whether or not media queries are supported. For example:

```
html { font-size: 100%; line-height: 1.5; }
   /* font-size:line-height = 16:24 = 2:3 = perfect
fifth */

h1 { font-size: 32px; font-size: 2rem; line-height:
1.25; }
   /* font-size:line-height = 32:40 = 4:5 = major third
      body:h1 = 16:32 = 1:2 = octave */
```

While we should really consider methods of delivering images appropriate to the screen size, let's just stick to a single image for all devices. But why don't we change its aspect ratio from 4:3 to 3:2, to fit with our harmonic scheme? It’s easy enough to do with `overflow:hidden` on the `<figure>` element to hide a part of the image, and a negative `margin` fudge:

```
figure img { margin: -8.5% 0 0 0; width: 100%;
max-width: 100%; }
```

Our first break point targets devices 320 pixels wide with an aspect ratio of 2:3, namely the iPhone and iPod Touch:

```
/* 320px = 20×16 */
@media only screen and (min-width: 20em) and
(device-aspect-ratio: 2/3) { }
```

We’re actually already there, of course, as the intervals we’ve chosen resonate with this aspect ratio – the content is already bound to the device.

Our next media query, then, will make some changes to match a different ratio, the major sixth (3:5), which is same as that of many smartphones:

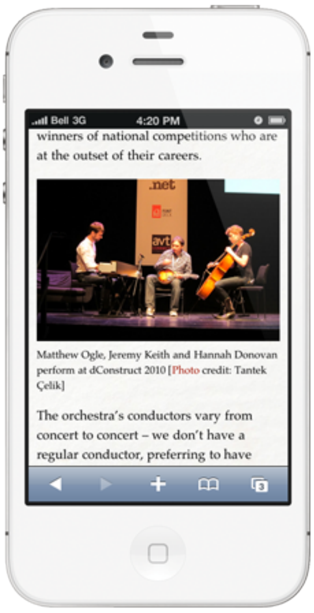
```
/* 480px = 30×16 */
@media only screen and (min-width: 30em) and
(device-aspect-ratio: 3/5) { }
```

A different aspect ratio might require a change in harmony. For devices with these proportions, we’ll now use the perfect fourth (3:4) and the major sixth (3:5) along with the octave we already have to create a new resonating harmony. For instance, a slightly wider screen means we can increase the `line-height` to aid the legibility of longer lines:

```
html { line-height: 1.667; }
   /* font-size:line-height = 16:26.672 = 3:5 = major
sixth */

h1 { font-size: 32px; font-size: 2rem; line-height:
1.667; }
   /* font-size:line-height = 32:53.333 = 3:5 = major
sixth
      body:h1 = 16:32 = 1:2 = octave */
```

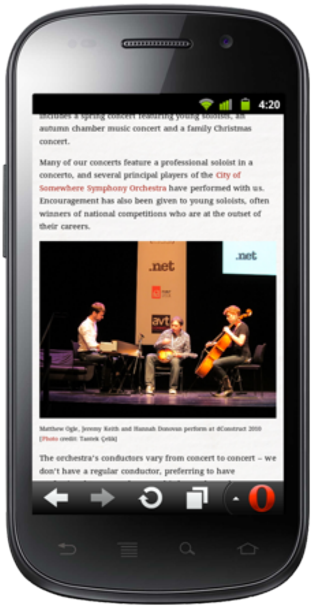
and we can remove the negative margin to display our 4:3 image in its entirety.



winners of national competitions who are at the outset of their careers.

Matthew Ogle, Jeremy Keith and Hannah Donovan perform at dConstruct 2010 [Photo credit: Tantek Çelik]

The orchestra's conductors vary from concert to concert – we don't have a regular conductor, preferring to have



autumn chamber music concert and a family Christmas concert.

Many of our concerts feature a professional soloist in a concerto, and several principal players of the City of Somewhere Symphony Orchestra have performed with us. Encouragement has also been given to young soloists, often winners of national competitions who are at the outset of their careers.

Matthew Ogle, Jeremy Keith and Hannah Donovan perform at dConstruct 2010 [Photo credit: Tantek Çelik]

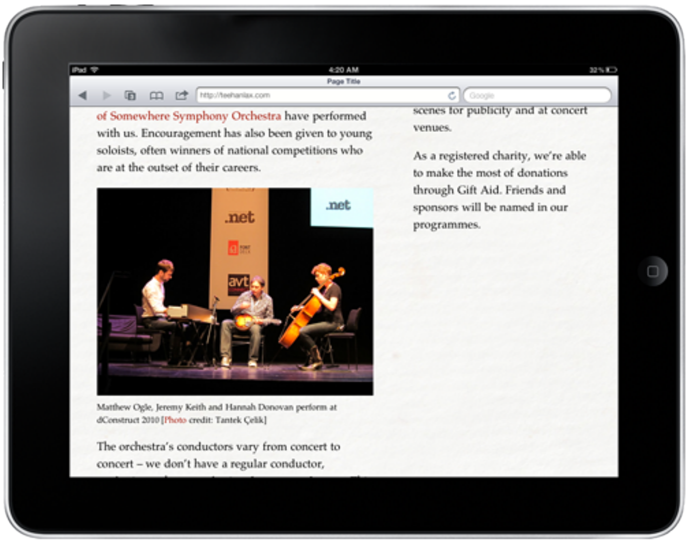
The orchestra's conductors vary from concert to concert – we don't have a regular conductor, preferring to have

Each screen displays content styled using relationships related to its own proportions. On the left, an iPhone 4 (2:3); on the right, a Samsung Nexus S (3:5). Your mileage may vary.

Another device, another media query. At 768 pixels, screens are wide enough to add columns. The ratios we've used for the 3:5 screens include the perfect fourth (3:4) so we don't need to change any of the font measurements, but we can base the proportions of the columns on the major sixth interval:

```
article { float: left; width: 56%; }
   /* width of main column 3:5 (60% of 100%, major sixth)
      incorporating gutter width */

aside { float : right; width : 36%; }
```



of Somewhere Symphony Orchestra have performed with us. Encouragement has also been given to young soloists, often winners of national competitions who are at the outset of their careers.

Matthew Ogle, Jeremy Keith and Hannah Donovan perform at dConstruct 2010 [Photo credit: Tantek Çelik]

The orchestra's conductors vary from concert to concert – we don't have a regular conductor,

scenes for publicity and at concert venues.

As a registered charity, we're able to make the most of donations through Gift Aid. Friends and sponsors will be named in our programmes.

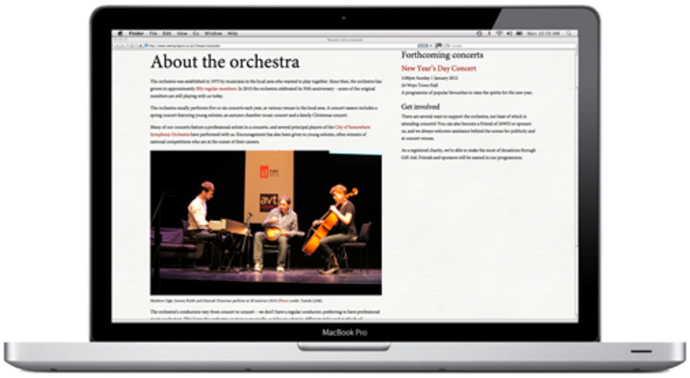
On devices with a 3:4 aspect ratio, this works even better in landscape orientation.

While not every screen over 768 pixels wide will have 3:4 proportions, the range of intervals informing the design ensure harmonious relationships between the different parts of the layout.

For wide screens proper (break point at 1,280 pixels) we can employ a new set of harmonious intervals. Many laptop and desktop screens have a 16:10 aspect ratio, which boils down to 8:5, equivalent to the minor sixth (5:8). Combined with a minor third (5:6) and the octave (1:2), this creates a new harmony appropriate to these devices. Let's increase the font size and change the image's aspect ratio to match:

```
html { font-size: 120%; line-height: 1.6; }
   /* font-size increased for wider screens from 16px to
19.2px
      (5:6 = minor third)
      font-size:line-height = 19.2:30.72 = 5:8 = minor
sixth */

figure img { margin: -12.5% 0 0 ; }
   /* using -ve margin combined with overflow:hidden
      on the figure element
      to crop the image from 4:3 to 8:5 = minor sixth */
```



A wide screen with a 8:5 (16:10) aspect ratio and an image to match.

With more pixels at our disposal, we can also now use the musical interval ratios to determine the width of the layout, and change the column proportions as well:

```
section { margin: 0 auto; width: 83.333%; }
   /* content width:screen width = 5:6 = minor third */

article { width: 60%; }
   /* width of main column 5:8 (62.5% of 100%, minor
sixth)
      incorporating gutter width */

aside { width: 35%; }
```

With some carefully targeted media queries, we can begin to reach towards fulfilling the second and third tenets of Boulton's new canon for web design: connecting the parts of content through relationships embodied in the layout design; and binding the content to the devices people use to access it.

CODA

Musical interval ratios and screen aspect ratios reveal more than convenient correspondence. These proportions work on a deep aesthetic level. Much is claimed for the golden ratio φ, but none of the screens pervading our lives use it. Perhaps that's an accident of technology, but can making screens to φ's proportions be more difficult or expensive than 2:3 or 3:4 or 16:10? Here, then, is not just one but a set of proportions with a uniquely human focus, originating in nature, recognized in antiquity, fundamental still.

We find music to be an art steeped with meaning, yet, unlike literary and representational arts, purely instrumental music has no obvious semantic content. It boasts an ability to express emotions while remaining an abstract art in some sense, which makes it very like design. These days, we're rightly encouraged to design for emotion, to make our users' experience meaningful, seductive, delightful. Using musical ideas and principles in our designs can help achieve those ends.

Let's not be naïve, of course; designing web pages is even less like composing music than it's like designing for print. In visual design, the eye will always be sovereign to the ear; following these principles will only get us so far. We cannot truly claim that a carefully composed web page layout will have the same qualities and effect as any musical patterns that inform it. In music, a set of intervals is always harmonious in relation to other sets of intervals: music rarely stands still. What aspect ratios will future screens take? Already today there is great variation in devices and support for media queries (and within that, support for `device-aspect-ratio`). And what of non-western musical traditions? Or rhythm, form, tempo and dynamics? What I've demonstrated above is only a suggestion, a tentative exploration of one possible way forward.

But as our discipline matures and we become more articulate about what we do, we must look longer and deeper into areas of human endeavour already rich with value. Music is a fertile ground to explore and has the potential to yield up new approaches for web design.

Footnotes

1. Just intonation is a system of tuning that uses small integers to describe the musical intervals, based initially on the perfect fifth, that most consonant of intervals.

Simple instruments such as vibrating strings and natural horns, as well as unaccompanied voices, tend to fall into just intonation naturally.

ABOUT THE AUTHOR

Owen Gregory is an editor, website designer and musician living in Birmingham, UK. He started designing for the web in 1998 and established his small business Full Cream Milk in 2006. Prior to that, Owen studied English and writing to master's level, and he now brings these two interests together for your friendly neighbourhood web book publishers, like Five Simple Steps. He tweets as @FullCreamMilk because FullCreamMilkMan is too long for Twitter.

Owen is what is sometimes called a classically trained musician, and he plays oboe and cor anglais (neither English nor a horn) in a number of non-professional orchestras.

Oh, and Andy Clarke once thanked Owen for “being Lewis to my Morse”. Which is better than being Robin to his Batman.

10. Context First: Web Strategy in Four Handy Ws

Alex Morris 24ways.org/201110

Many, many years ago, before web design became my proper job, I trained and worked as a journalist. I studied publishing in London and spent three fun years learning how to take a few little nuggets of information and turn them into a story. I learned a bunch of stuff that has all been a huge help to my design career. Flatplanning, layout, typographic theory. All of these disciplines have since translated really well to web design, but without doubt the most useful thing I learned was how to ask difficult questions.

Pretty much from day one of journalism school they hammer into you the importance of the Five Ws. Five disarmingly simple lines of enquiry that eloquently manage to provide the meat of any decent story. And with alliteration thrown in too. For a young journo, it's almost too good to be true.

Who? What? Where? When? Why? It seems so obvious to almost be trite but, fundamentally, any story that manages to answer those questions for the reader is doing a pretty good job. You'll probably have noticed feeling underwhelmed by certain news pieces in the past – disappointed, like something was missing. Some irritating oversight that really lets the story down. No doubt it was one of the Ws – those innocuous little suckers are generally only noticeable by their absence, but they sure get missed when they're not there.

QUESTION EVERYTHING

I've always been curious. An inveterate tinkerer with things and asker of dopey questions, often to the point of abject annoyance for anyone unfortunate enough to have ended up in my line of fire. So, naturally, the Five Ws started drifting into other areas of my life. I'd scrutinize everything, trying to justify or explain my rationale using these Ws, but I'd also find myself ripping apart the stuff that clearly couldn't justify itself against the same criteria.

So when I started working as a designer I applied the same logic and, sure enough, the Ws pretty much mapped to the exact same needs we had for gathering requirements at the start of a project. It seemed so obvious, such a simple way to establish the purpose of a product. What was it for? Why we were making it? And, of course, who were we making it for? It forced clients to stop and think, when really what they wanted was to get going and see something shiny. Sometimes that was a tricky conversation to have, but it's no coincidence that those who got it also understood the value of strategy and went on to have good solid products, while those that didn't often ended up with arrogantly insular and very shiny but ultimately unsatisfying and expendable products. Empty vessels make the most noise and all that…

CONTENT FIRST

I was both surprised and pleased when the whole content first idea started to rear its head a couple of years back. Pleased, because without doubt it's absolutely the right way to work. And surprised, because personally it's always been the way I've done it – I wasn't aware there was even an alternative way. Content in some form or another is the whole reason we were making the things we were making. I can't even imagine how you'd start figuring out what a site needs to do, how it should be structured, or how it should look without a really good idea of what that content might be. It baffles me still that this was somehow news to a lot of people. What on earth were they doing? Design without purpose is just folly, surely?

It's great to see the idea gaining momentum but, having watched it unfold, it occurred to me recently that although it's fantastic to see a tangible shift in thinking – away from those bleak times, where making things up was somehow deemed an appropriate way to do things – there's now a new bad guy in town.

With any buzzword solution of the moment, there's always a catch, and it seems like some have taken the content first approach a little too literally. By which I mean, it's literally the first thing they do. The project starts, there's a very cursory nod towards gathering requirements, and off they go, cranking content. Writing copy, making video, commissioning illustrations.

All that's happened is that the 'making stuff up' part has shifted along the line, away from layout and UI, back to the content.

STARTING IS TOO EASY

I can't remember where I first heard that phrase, but it's a great sentiment which applies to so much of what we do on the web. The medium is so accessible and to an extent disposable; throwing things together quickly carries far less burden than in any other industry. We're used to tweaking as we go, changing bits, iterating things into shape. The ubiquitous beta tag has become the ultimate caveat, and has made the unfinished and unpolished acceptable. Of course, that can work brilliantly in some circumstances. Occasionally, a product offers such a paradigm shift it's beyond the level of deep planning and prelaunch finessing we'd ideally like. But, in the main, for most client sites we work on, there really is no excuse not to do things properly. To ask the tricky questions, to challenge preconceptions and really understand the Ws behind the products we're making before we even start.

THE FOUR WS

For product definition, only four of the five Ws really apply, although there's a lot of discussion around the idea of **when** being an influencing factor. For example, the context of a user's engagement with your product is something you can make a call on depending on the specifics of the project.

So, here's my take on the four essential Ws. I'll point out here that, of course, these are not intended to be autocratic dictums. Your needs may differ, your clients' needs may differ, but these four starting points will get you pretty close to where you need to be.

Who

It's surprising just how many projects start without a real understanding of the intended audience. Many clients think they have an idea, but without really knowing – it's presumptive at best, and we all know what presumption is the mother of, right? Of course, we can't know our audiences in the same way a small shop owner might know their customers. But we can at least strive to find out what type of people are likely to be using the product. I'm not talking about deep user research. That should come later.

These are the absolute basics. What's the context for their visit? How informed are they? What's their level of comprehension? Are they able to self-identify and relate to categories you have created? I could go on, and it changes on a per-project basis. You'll only find this out by speaking to them, if not in person, then indirectly through surveys, questionnaires or polls. The mechanism is less important than actually reaching out and engaging with them, because without that understanding it's impossible to start to design with any empathy.

What

Once you become deeply involved directly with a product or service, it's notoriously difficult to see things as an outsider would. You learn the thing inside and out, you develop shortcuts and internal phraseology. Colloquialisms creep in. You become too close. So it’s no surprise when clients sometimes struggle to explain what it is their product actually does in a way that others can understand.

Often products are complex but, really, the core reasons behind someone wanting to use that product are very simple. There’s a value proposition for the customer and, if they choose to engage with it, there’s a value exchange. If that proposition or exchange isn’t transparent, then people become confused and will likely go elsewhere. Make sure both your client and you really understand what that proposition is and, in turn, what the expected exchange should be. In a nutshell: what is the intended outcome of that engagement? Often the best way to do this is strip everything back to nothing. Verbosity is rife on the web. Just because it’s easy to create content, that shouldn’t be a reason to do so. Figure out what the value proposition is and then reintroduce content elements that genuinely help explain or present that to a level that is appropriate for the audience.

Why

In advertising, they talk about the truths behind a product or service. Truths can be both tangible or abstract, but the most important part is the resonance those truths hit with a customer. In a digital product or service those truths are often exposed as benefits. Why is this what I need? Why will it work for me? Why should I trust you? The why is one of the more fluffy Ws, yet it’s such an important one to nail. Clients can get prickly when you ask them to justify the why behind their product, but it’s a fantastic way to make sure the value proposition is clear, realistic and meets with the expectations of both client and customer.

It’s our job as designers to question things: we’re not just a pair of hands for clients. Just recently I spoke to a potential client about a site for his business. I asked him why people would use his product and also why his product seemed so fractured in its direction. He couldnt answer that question so, instead of ploughing on regardless, he went back to his directors and is now re-evaluating that business. It was awkward but he thanked me and hopefully he’ll have a better product as a result.

Where

In this instance, where is not so much a geographical thing, although in some cases that level of context may indeed become a influencing factor… The where we’re talking about here is the position of the product in relation to others around it. By looking at competitors or similar services around the one you are designing, you can start to get a sense for many of the things that are otherwise hard to pin down or have yet to be defined. For example, in a collection of sites all selling cars, where does yours fit most closely? Where are the overlaps? How are they communicating to their customers? How is the product range presented or categorized?

It’s good to look around and see how others are doing it. Not in a quest for homogeneity but more to reference or to avoid certain patterns that may or may not make sense for your own particular product. Clients often strive to be different for the sake of it. They feel they need to provide distinction by going against the flow a bit. We know different. We know users love convention. They embrace familiar mental models. They’re comfortable with things that they’ve experienced elsewhere. By showing your client that position is a vital part of their strategy, you can help shape their product into something great.

TO CONCLUDE

So there we have it – the four Ws. Each part tells a different and vital part of the story you need to be able to make a really good product. It might sound like a lot of work, particularly when the client is breathing down your neck expecting to see things, but without those pieces in place, the story you’re building your product on, and the content that you’re creating to form that product can only ever fit into one genre. Fiction.

ABOUT THE AUTHOR

Alex Morris is an Interaction Designer with over 15 years experience designing and building websites, games and applications for clients ranging from small startups to multinational news networks (and pretty much everything in between). Now working as User Experience Director for Mark Boulton Design.

An obsessive tinkerer, Alex likes to get his hands dirty taking things apart and then re-building them. Alex has hand rolled his own CMS, written a bespoke ecommerce platform, released a number of iOS apps and is hell bent on creating a better way to design websites. The first part of that quest launched in November 2011 with Alex's first Macintosh application. CSS3 Toolkit is a simple utility tool that allows designers and developers to create complex CSS3 effects without writing a single line of code. The plan is to build on the custom webkit engine that runs Toolkit, to create a tool to design in the browser without needing to know any HTML or CSS.

Alex learned HTML whilst studying publishing back in 1996 and has since used it daily for everything from quick and dirty prototypes through to large scale applications.

With a firm focus on UX since the beginning of his career, Alex considers himself a multi-disciplined designer combining Interface Design with the ability to execute technical solutions when the need arises.

Alex contributes to Net magazine, blogs at mistermorris.tumblr.com and can be found on Twitter as @aexmo

11. Nine Things I've Learned

Mike Kus

24ways.org/201111

I’ve been a professional graphic designer for fourteen years and for just under four of those a professional web designer. Like most designers I’ve learned a lot in my time, both from a design point of view and in business as freelance designer. A few of the things I’ve learned stick out in my mind, so I thought I’d share them with you. They’re pretty random and in no particular order.

1. BECOMING THE DESIGNER YOU WANT TO BE

When I started out as a young graphic designer, I wanted to design posters and record sleeves, pretty much like every other young graphic designer. The problem is that the reality of the world means that when you get your first job you’re designing the back of a paracetamol packet or something equally weird. I recently saw a tweet that went something like this: “You’ll never become the designer you always dreamt of being by doing the work you never wanted to do". This is so true; to become the designer you want to be, you need to be designing the things you're passionate about designing. This probably this means working in the evenings and weekends for little or no money, but it's time well spent. Doing this will build up your portfolio with the work that really shows what you can do! Soon, someone will ask you to design something based on having seen this work. From this point, you're carving your own path in the direction of becoming the designer you always wanted to be.

2. COMPETE ON YOUR OWN TERMS

As well as all being friends, we are also competitors. In order to win new work we need a selling point, preferably a unique selling point. Web design is a combination of design disciplines – user experience design, user interface Design, visual design, development, and so on. Some companies will sell themselves as UX specialists, which is fine, but everyone who designs a website from scratch does some sort of UX, so it's not really a unique selling point. Of course, some people do it better than others.

One area of web design that clients have a strong opinion on, and will judge you by, is visual design. It's an area in which it's definitely possible to have a unique selling point. Designing the visual aesthetic for a website is a combination of logical decision making and a certain amount of personal style. If you can create a unique visual style to your work, it can become a selling point that's unique to you.

3. HOW MUCH TO CHARGE AND STAYING MOTIVATED

When you're a freelance designer one of the hardest things to do is put a price on your work and skills. Finding the right amount to charge is a fine balance between supplying value to your customer and also charging enough to stay motivated to do a great job. It's always tempting to offer a low price to win work, but it's often not the best approach: not just for yourself but for the client as well.

A client once asked me if I could reduce my fee by £1,000 and still be motivated enough to do a good job. In this case the answer was yes, but it was the question that resonated with me. I realized I could use this as a gauge to help me price projects. Before I send out a quote I now always ask myself the question "Is the amount I've quoted enough to make me feel motivated to do my best on this project?" I never send out a quote unless the answer is yes. In my mind there's no point in doing any project half-heartedly, as every project is an opportunity to build your reputation and expand your portfolio to show potential clients what you can do. Offering a client a good price but not being prepared to put everything you have into it, isn't value for money.

4. SUPPLYING THE RIGHT DESIGN

When I started out as a graphic designer it seemed to be the done thing to supply clients with a ton of options for their logo or brochure designs. In a talk given by Dan Rubin, he mentioned that this was a legacy of agencies competing with each other in a bid to create the illusion of offering more value for money. Over the years, I've realized that offering more than one solution makes no sense. The reason a client comes to you as a designer is because you're the person than can get it right. If I were to supply three options, I'd be knowingly offering my client at least two options that I didn't think worked.

To this day I still get asked how many homepage design options I'll supply for the quoted amount. The answer is one. Of course, I'm more than happy to iterate upon the design to fine-tune it and, on the odd occasion, I do revisit a design concept if I just didn't nail the design first time around. Your time is much better spent refining the right design option than rushing out three substandard designs in the same amount of time.

5. COLOUR IS KEY

There are many contributing factors that go into making a good visual design, but one of the simplest ways to do this is through the use of colour. The colour palette used in a design can have such a profound effect on a visual design that it almost feels like you're cheating. It's easy to add more and more subtle shades of colour to add a sense of sophistication and complexity to a design, but it dilutes the overall visual impact. When I design, I almost have a rule that only allows me to use a very limited colour palette. I don't always stick to it, but it's always in mind and something I'm constantly reviewing through my design process.

6. CREATIVE THINKING IS CENTRAL TO GOOD OR BOUNDARY-PUSHING WEB DESIGN

When we think of creativity in web design we often link this to the visual design, as there is an obvious opportunity to be creative in this area if the brief allows it. Something that I've learnt in my time as a web designer is that there's a massive need for creative thinking in the more technical aspects of web design. The tools we use for building websites are there to be manipulated and used in creative ways to design exciting and engaging user experiences. Great developers are constantly using their creativity to push the boundaries of what can be done with CSS, jQuery and JavaScript.

Being creative and creative thinking are things we should embrace as an industry and they are qualities that can be found in anyone, whether they be a visual designer or Rails developer.

7. CREATIVE BLOCK: DON'T BE AFRAID TO GET THINGS WRONG

Creative block can be a killer when designing. It's often applied to visual design, which is more subjective. I suffer from creative block on a regular basis. It's hugely frustrating and can screw up your schedule. Having thought about what creative block actually is, I've come to the conclusion that it's actually more of a lack of direction than a lack of ideas. You have ideas and solutions in mind but don't feel committed to any of them. You're scared that whatever direction you take, it'll turn out to be wrong. I've found that the best remedy for this is to work through this barrier. It's a bit like designing with a blindfold on – you don't really know where you're going. If you stick to your guns and keep pressing forward I find that, nine times out of ten, this process leads to a solution. As the page begins to fill, the direction you're looking for slowly begins to take shape.

8. YOU GET BETTER AT DESIGNING BY DESIGNING

I often get emails asking me what books someone can read to help them become a better designer. There are a lot of good books on subjects like HTML5, CSS, responsive web design and the like, that will really help improve anyone's web design skills. But, when it comes to visual design, the best way to get better is to design as much as possible. You can't follow instructions for these things because design isn't following instructions. A large part of web design is definitely applying a set of widely held conventions, but there's another part to it that is invention and the only way to get better at this is to do it as much as possible.

9. SELF-BELIEF IS OVERRATED

Throughout our lives we're told to have self-belief. Self-belief and confidence in what we do, whatever that may be. The problem is that some people find it easier than others to believe in themselves. I've spent years trying to convince myself to believe in what I do but have always found it difficult to have complete confidence in my design skills. Self-doubt always creeps in.

I've realized that it's ok to doubt myself and I think it might even be a good thing! I've realized that it's my self-doubt that propels me forward and makes me work harder to achieve the best results. The reason I'm sharing this is because I know I'm not the only designer that feels this way. You can spend a lot of time fighting self-doubt only to discover that it's your body's natural mechanism to help you do the best job possible.

ABOUT THE AUTHOR

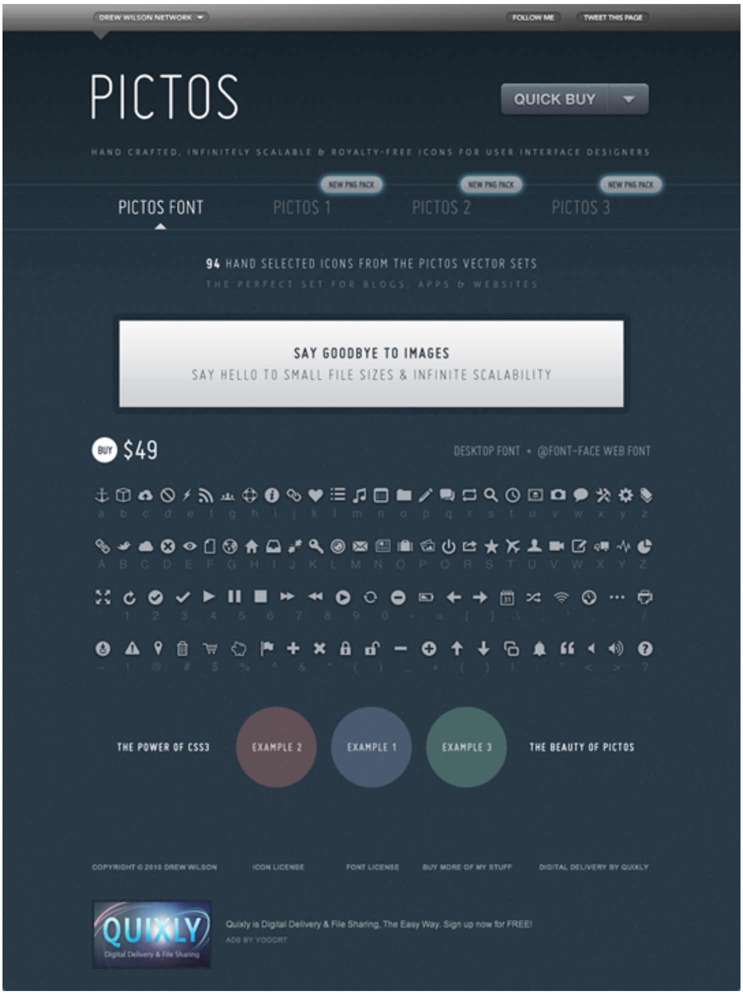
Mike Kus is a web/graphic designer & illustrator. He's based in UK and works for clients worldwide. You can see his work at mikekus.com.

12. Displaying Icons with Fonts and Data-Attributes

Jon Hicks 24ways.org/201112

Traditionally, bitmap formats such as PNG have been the standard way of delivering iconography on websites. They're quick and easy, and it also ensures they're as pixel crisp as possible. Bitmaps have two drawbacks, however: multiple HTTP requests, affecting the page's loading performance; and a lack of scalability, noticeable when the page is zoomed or viewed on a screen with a high pixel density, such as the iPhone 4 and 4S.

The requests problem is normally solved by using CSS sprites, combining the icon set into one (physically) large image file and showing the relevant portion via `background-position`. While this works well, it can get a bit fiddly to specify all the positions. In particular, scalability is still an issue. A vector-based format such as SVG sounds ideal to solve this, but browser support is still patchy.


DREW WILSON NETWORK
FOLLOW ME
TWEET THIS PAGE
PICTOS
QUICK BUY
HAND CRAFTED, INFINITELY SCALABLE & ROYALTY-FREE ICONS FOR USER INTERFACE DESIGNERS
NEW PNG PACK
PICTOS FONT
PICTOS 1
PICTOS 2
PICTOS 3
94 HAND SELECTED ICONS FROM THE PICTOS VECTOR SETS
THE PERFECT SET FOR BLOGS, APPS & WEBSITES
SAY GOODBYE TO IMAGES
SAY HELLO TO SMALL FILE SIZES & INFINITE SCALABILITY
BUY $49
DESKTOP FONT • @FONT-FACE WEB FONT
THE POWER OF CSS3
EXAMPLE 2
EXAMPLE 1
EXAMPLE 3
THE BEAUTY OF PICTOS
COPYRIGHT © 2010 DREW WILSON
ICON LICENSE
FONT LICENSE
BUY MORE OF MY STUFF
DIGITAL DELIVERY BY QUIXLY
QUIXLY Digital Delivery & File Sharing
Quixly is Digital Delivery & File Sharing, The Easy Way. Sign up now for FREE!
ADS BY YOOGRT

The rise and adoption of web fonts have given us another alternative. By their very nature, they're not only scalable, but resolution-independent too. No need to specify higher resolution graphics for high resolution screens!

That's not all though:

- **Browser support**: Unlike a lot of new shiny techniques, they have been supported by Internet Explorer since version 4, and, of course, by all modern browsers. We do need several different formats, however!
- **Design on the fly**: The font contains the basic graphic, which can then be coloured easily with CSS – changing colours for themes or `:hover` and `:focus` styles is done with one line of CSS, rather than requiring a new graphic. You can also use CSS3 properties such as `text-shadow` to add further effects. Using `-webkit-background-clip: text;`, it's possible to use gradient and inset shadow effects, although this creates a bitmap mask which spoils the scalability.
- **Small file size**: specially designed icon fonts, such as Drew Wilson's Pictos font, can be as little as 12Kb for the .woff font. This is because they contain fewer characters than a fully fledged font. You can see Pictos being used in the wild on sites like Garrett Murray's Maniacal Rage.

As with all formats though, it's not without its disadvantages:

- Icons can only be rendered in monochrome or with a gradient fill in browsers that are capable of rendering CSS3 gradients. Specific parts of the icon can't be a different colour.
- It's only appropriate when there is an accompanying text to provide meaning. This can be alleviated by wrapping the text label in a tag (I like to use `<b>` rather than `<span>`, due to the fact that it's smaller and isn' t being used elsewhere) and then hiding it from view with `text-indent:-999em`.
- Creating an icon font can be a complex and time-consuming process. While font editors can carry out hinting automatically, the best results are achieved manually.
- Unless you're adept at creating your own fonts, you're restricted to what is available in the font. However, fonts like Pictos will cover the most common needs, and icons are most effective when they're using familiar conventions.

The main complaint about using fonts for icons is that it can mean adding a meaningless character to our markup. The good news is that we can overcome this by using one of two methods – CSS generated content or the `data-icon` attribute – in combination with the `:before` and `:after` pseudo-selectors, to keep our markup minimal and meaningful.

Our simple markup looks like this:

```
<a href="/basket" class="icon basket">View Basket</a>
```

Note the multiple class attributes. Next, we'll import the Pictos font using the `@font-face` web fonts property in CSS:

```
@font-face {
    font-family: 'Pictos';
    src: url('pictos-web.eot');
    src: local('☺'),
    url('pictos-web.woff') format('woff'),
    url('pictos-web.ttf') format('truetype'),
    url('pictos-web.svg#webfontIyfZbseF') format('svg');
}
```

This rather complicated looking set of rules is (at the time of writing) the most bulletproof way of ensuring as many browsers as possible load the font we want. We'll now use the `content` property applied to the `:before` pseudo-class selector to generate our icon. Once again, we'll use those multiple class attribute values to set common icon styles, then specific styles for `.basket`. This helps us avoid repeating styles:

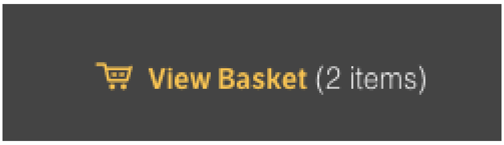
```
.icon {
    font-family: 'Pictos';
    font-size: 22px:
}

.basket:before {
    content: "$";
}
```

What does the `:before` pseudo-class do? It generates the dollar character in a browser, even when it's not present in the markup. Using the generated content approach means our markup stays simple, but we'll need a new line of CSS, defining what letter to apply to each class attribute for every icon we add.

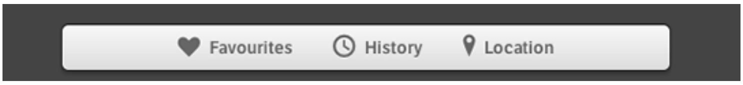
`data-icon` is a new alternative approach that uses the HTML5 `data-` attribute in combination with CSS attribute selectors. This new attribute lets us add our own metadata to elements, as long as its prefixed by `data-` and doesn't contain any uppercase letters. In this case, we want to use it to provide the letter value for the icon. Look closely at this markup and you'll see the `data-icon` attribute.

```
<a href="/basket" class="icon" data-icon="$">View
Basket</a>
```



We could add others, in fact as many as we like.

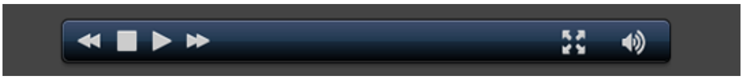
```
<a href="/" class="icon" data-icon="k">Favourites</a>
<a href="/" class="icon" data-icon="t">History</a>
<a href="/" class="icon" data-icon="@">Location</a>
```

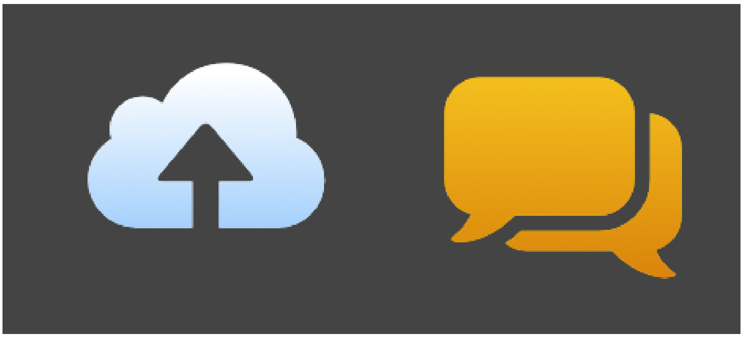


Then, we need just one CSS attribute selector to style all our icons in one go:

```
.icon:before {
    content: attr(data-icon);
    /* Insert your fancy colours here */
    }
```

By placing our custom attribute `data-icon` in the selector in this way, we can enable CSS to read the value of that attribute and display it before the element (in this case, the anchor tag). It saves writing a lot of CSS rules. I can imagine that some may not like the extra attribute, but it does keep it out of the actual content – generated or not.





This could be used for all manner of tasks, including a media player and large simple illustrations. See the demo for live examples. Go ahead and zoom the page, and the icons will be crisp, with the exception of the examples that use `-webkit-background-clip: text` as mentioned earlier.

Finally, it's worth pointing out that with both generated content and the `data-icon` method, the letter will be announced to people using screen readers. For example, with the shopping basket icon above, the reader will say "dollar sign view basket". As accessibility issues go, it's not exactly the worst, but could be confusing. You would need to decide whether this method is appropriate for the audience. Despite the disadvantages, icon fonts have huge potential.

ABOUT THE AUTHOR

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

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

13. Your jQuery: Now With 67% Less Suck

Scott Kosman 24ways.org/201113

Fun fact: more websites are now using jQuery than Flash.

jQuery is an amazing tool that's made JavaScript accessible to developers and designers of all levels of experience. However, as Spiderman taught us, "with great power comes great responsibility." The unfortunate downside to jQuery is that while it makes it easy to write JavaScript, it makes it easy to write really really f*&#ing bad JavaScript. Scripts that slow down page load, unresponsive user interfaces, and spaghetti code knotted so deep that it should come with a bottle of whiskey for the next ~~sucker~~ developer that has to work on it.

This becomes more important for those of us who have yet to move into the magical fairy wonderland where none of our clients or users view our pages in Internet Explorer. The IE JavaScript engine moves at the speed of an advancing glacier compared to more modern browsers, so optimizing our code for performance takes on an even higher level of urgency.

Thankfully, there are a few very simple things anyone can add into their jQuery workflow that can clear up a lot of basic problems. When undertaking code reviews, three of the areas where I consistently see the biggest problems are: inefficient selectors; poor event delegation; and clunky DOM manipulation. We'll tackle all three of these and hopefully you'll walk away with some new jQuery batarangs to toss around in your next project.

SELECTOR OPTIMIZATION

Selector speed: fast or slow?

Saying that the power behind jQuery comes from its ability to select DOM elements and act on them is like saying that Photoshop is a really good tool for selecting pixels on screen and making them change color – it's a bit of a gross oversimplification, but the fact remains that jQuery gives us a ton of ways to choose which element or elements in a page we want to work with. However, a surprising number of web developers are unaware that all selectors are not created equal; in fact, it's incredible just how drastic the performance difference can be between two selectors that, at first glance, appear nearly identical. For instance, consider these two ways of selecting all paragraph tags inside a `<div>` with an ID.

```
$("#id p");
```

```
$("#id").find("p");
```

Would it surprise you to learn that the second way can be more than twice as fast as the first? Knowing which selectors outperform others (and why) is a pretty key building block in making sure your code runs well and doesn't frustrate your users waiting for things to happen.

There are many different ways to select elements using jQuery, but the most common ways can be basically broken down into five different methods. In order, roughly, from fastest to slowest, these are:

- `$("#id");`
This is without a doubt the fastest selector jQuery provides because it maps directly to the native `document.getElementbyId()` JavaScript method. If possible, the selectors listed below should be prefaced with an ID selector in conjunction with jQuery's .find() method to limit the scope of the page that has to be searched (as in the `$("#id").find("p")` example shown above).
- `$("p");`, `$("input");`, `$("form");` and so on
Selecting elements by tag name is also fast, since it maps directly to the native `document.getElementsByTagname()` method.
- `$(".class");`
Selecting by class name is a little trickier. While still performing very well in modern browsers, it can cause

some pretty significant slowdowns in IE8 and below. Why? IE9 was the first IE version to support the native `document.getElementsByClassName()` JavaScript method. Older browsers have to resort to using much slower DOM-scraping methods that can really impact performance.

- `$("[attribute=value]");`

There is no native JavaScript method for this selector to use, so the only way that jQuery can perform the search is by crawling the entire DOM looking for matches. Modern browsers that support the `querySelectorAll()` method will perform better in certain cases (Opera, especially, runs these searches much faster than any other browser) but, generally speaking, this type of selector is Slowey McSlowersons.

- `$(":hidden");`

Like attribute selectors, there is no native JavaScript method for this one to use. Pseudo-selectors can be painfully slow since the selector has to be run against every element in your search space. Again, modern browsers with `querySelectorAll()` will perform slightly better here, but try to avoid these if at all possible. If you must use one, try to limit the search space to a specific portion of the page: `$("#list").find(":hidden");`

But, hey, proof is in the performance testing, right? It just so happens that said proof is sitting right here. Be sure to notice the class selector numbers beside IE7 and 8 compared to other browsers and then wonder how the people on the IE team at Microsoft manage to sleep at night. Yikes.

Chaining

Almost all jQuery methods return a jQuery object. This means that when a method is run, its results are returned and you can continue executing more methods on them. Rather than writing out the same selector multiple times over, just making a selection once allows multiple actions to be run on it.

WITHOUT CHAINING

```
$("#object").addClass("active");
$("#object").css("color","#f0f");
$("#object").height(300);
```

WITH CHAINING

```
$("#object").addClass("active").css("color",
"#f0f").height(300);
```

This has the dual effect of making your code shorter *and* faster. Chained methods will be slightly faster than multiple methods made on a cached selector, and both ways will be *much* faster than multiple methods made on non-cached selectors. Wait… "cached selector"? What is this new devilry?

Caching

Another easy way to speed up your code that seems to be a mystery to developers is the idea of caching your selectors. Think of how many times you end up writing the same selector over and over again in any project. Every `$(".element")` selector has to search the entire DOM each time, regardless of whether or not that selector had been previously run. Running the selection once and then storing the results in a variable means that the DOM only has to be searched once. Once the results of a selector have been cached, you can do anything with them.

First, run your search (here we're selecting all of the `<li>` elements inside `<ul id="blocks">`):

```
var blocks = $("#blocks").find("li");
```

Now, you can use the `blocks` variable wherever you want without having to search the DOM every time.

```
$("#hideBlocks").click(function() {
    blocks.fadeOut();
});
$("#showBlocks").click(function() {
    blocks.fadeIn();
});
```

My advice? Any selector that gets run more than once should be cached. This jsperf test shows just how much faster a cached selector runs compared to a non-cached one (and even throws some chaining love in to boot).

EVENT DELEGATION

Event listeners cost memory. In complex websites and apps it's not uncommon to have a lot of event listeners floating around, and thankfully jQuery provides some really easy methods for handling event listeners efficiently through delegation.

In a bit of an extreme example, imagine a situation where a 10×10 cell table needs to have an event listener on each cell; let's say that clicking on a cell adds or removes a class that defines the cell's background color. A typical way that this might be written (and something I've often seen during code reviews) is like so:

```
$('table').find('td').click(function() {
    $(this).toggleClass('active');
});
```

jQuery 1.7 has provided us with a new event listener method, .on(). It acts as a utility that wraps all of jQuery's previous event listeners into one convenient method, and the way you write it determines how it behaves. To rewrite the above `.click()` example using `.on()`, we'd simply do the following:

```
$('table').find('td').on('click',function() {
    $(this).toggleClass('active');
});
```

Simple enough, right? Sure, but the problem here is that we're still binding one hundred event listeners to our page, one to each individual table cell. A far better way to do things is to create one event listener on the table itself that listens for events inside it. Since the majority of events bubble up the DOM tree, we can bind a single event listener to one element (in this case, the `<table>`) and wait for events to bubble up from its children. The way to do this using the `.on()` method requires only one change from our code above:

```
$('table').on('click','td',function() {
    $(this).toggleClass('active');
});
```

All we've done is moved the `td` selector to an argument inside the `.on()` method. Providing a selector to `.on()` switches it into delegation mode, and the event is only fired for descendants of the bound element (`table`) that match the selector (`td`). With that one simple change, we've gone from having to bind one hundred event listeners to just one. You might think that the browser having to do one hundred times less work would be a good thing and you'd be completely right. The difference between the two examples above is staggering.

(Note that if your site is using a version of jQuery earlier than 1.7, you can accomplish the very same thing using the .delegate() method. The syntax of how you write the function differs slightly; if you've never used it before, it's worth checking the API docs for that page to see how it works.)

DOM MANIPULATION

jQuery makes it very easy to manipulate the DOM. It's trivial to create new nodes, insert them, remove other ones, move things around, and so on. While the code to do this is simple to write, every time the DOM is manipulated, the browser has to repaint and reflow content which can be extremely costly. This is no more evident than in a long loop, whether it be a standard `for()` loop, `while()` loop, or `jQuery $.each()` loop.

In this case, let's say we've just received an array full of image URLs from a database or Ajax call or wherever, and we want to put all of those images in an unordered list. Commonly, you'll see code like this to pull this off:

```
var arr = [reallyLongArrayOfImageURLs];
    $.each(arr, function(count, item) {
        var newImg = '<li><img src="'+item+'"></li>';
        $('#imgList').append(newImg);
    });
```

There are a couple of problems with this. For one (which you should have already noticed if you've read the earlier part of this article), we're making the `$("#imgList")` selection once for each iteration of our loop. The other problem here is that each time the loop iterates, it's adding a new `<li>` to the DOM. Each of those insertions is going to be costly, and if our array is quite large then this could lead to a massive slowdown or even the dreaded 'A script is causing this page to run slowly' warning.

```
var arr = [reallyLongArrayOfImageURLs],
    tmp = '';
$.each(arr, function(count, item) {
    tmp += '<li><img src="'+item+'"></li>';
});
$('#imgList').append(tmp);
```

All we've done here is create a `tmp` variable that each `<li>` is added to as it's created. Once our loop has finished iterating, that `tmp` variable will contain all of our list items in memory, and can be appended to our `<ul>` all in one go. Browsers work much faster when working with objects in memory rather than on screen, so this is a much faster, more CPU-cycle-friendly method of building a list.

WRAPPING UP

These are far from being the only ways to make your jQuery code run better, but they are among the simplest ones to implement. Though each individual change may only make a few milliseconds of difference, it doesn't take long for those milliseconds to add up. Studies have shown that the human eye can discern delays of as few as 100ms, so simply making a few changes sprinkled throughout your code can very easily have a noticeable effect on how well your website or app performs. Do you have other jQuery optimization tips to share? Leave them in the comments and help make us all better.

Now go forth and make awesome!

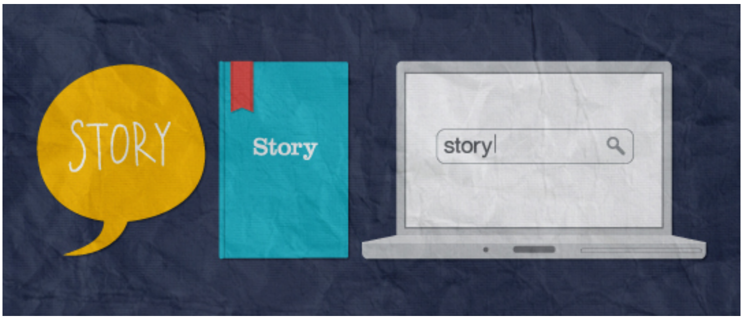
ABOUT THE AUTHOR

A 35 year old Canadian expat currently plying his trade as Associate Director of Standards Architecture at Crispin Porter + Bogusky in Göteborg, Sweden, **Scott Kosman** can also be found on the Twitters and some of his other work can even be found on the internet. He likes JavaScript, bicycles, cats, bacon, and thinks you're pretty awesome.

14. Design the Invisible to Tell Better Stories on the Web

Robert Mills 24ways.org/201114

For design to be meaningful we need to tell stories. We need to design the invisible, the cues, the messages and the extra detail hidden beneath the aesthetics. It's all about the story.



From verbal exchanges around the campfire to books, the web and everything in between, storytelling allows us to share, organize and process information more efficiently. It helps us understand our surroundings and make emotional connections to people, places and experiences.

Web design lends itself perfectly to the conventions of storytelling, a universal process. However, the stories vary because they're defined by culture, society, politics and religion. All of which need considering if you are to design stories that are relevant to your target audience.

The benefits of approaching design with storytelling in mind from the very start of the project is that we are creating considered design that allows users to quickly gather meaning from the website. They do this by reading between the lines and drawing on the wealth of knowledge they have acquired about the associations between colours, typyefaces and signs.

With so much recognition and analysis happening subconsciously you have to consider how design communicates on this level. This invisible layer has a significant impact on what you say, how you say it and who you say it to.

HOW CAN WE DESIGN SOMETHING THAT'S INVISIBLE?

By researching and making conscious decisions about exactly what you are communicating, you can make the invisible visible. As is often quoted, good design is like air, you only notice it when it's bad. So by designing the invisible the aim is to design stories that the audience receive subliminally, so that they go somewhat unnoticed, like good air.

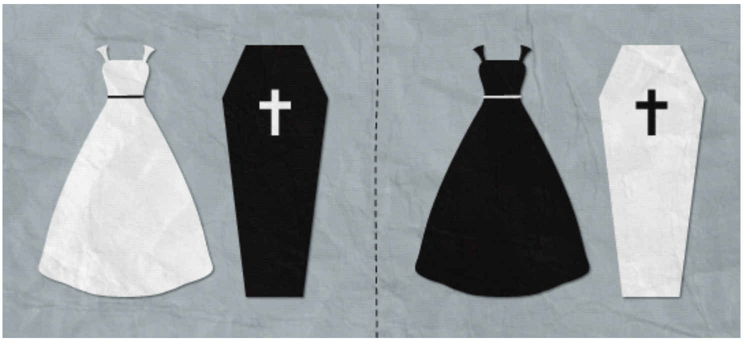
STORYTELLING STRANDS

To share these stories through design, you can break it down into several strands. Each strand tells a story on its own, but when combined they may start to tell a different story altogether. These strands are colour, typefaces, branding, tone of voice and symbols. All are literal and visible but the invisible element is the meaning behind them – meaning that you can extract and share. In this article I want to focus on colour, typefaces and tone of voice and on how combining story strands can change the meaning.

Colour

Let's start with colour. Red represents emotions such as love but can also signify war. Green is commonly used for all things environmental and purple is a colour that connotes wealth and royalty. These associations between colour and emotion or value have been learned over time and are continually reinforced through media and culture.

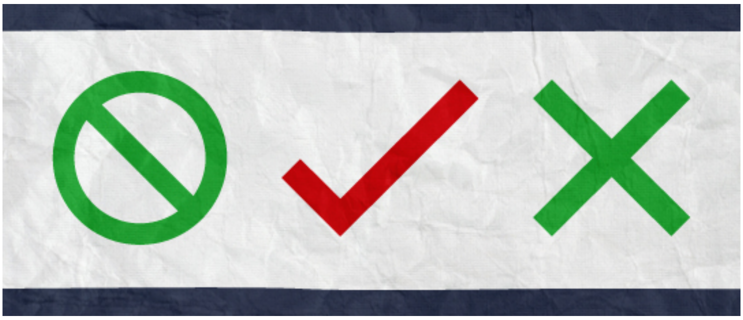
With this knowledge come expectations from your users. For example, they will expect Valentine's Day sites to be red and kids' sites to be bright and colourful. This is true in the same way audiences have expectations of certain genres of film or music. These conventions help savvy audiences decode texts and read between the lines or, rather, to draw meaning from the invisible. It's practically an innate skill. This is why you need to design the invisible: because users will quickly deduce meaning from your site and fill in the story's gaps, it's important to give them as much of that story to begin with. A story relevant to their culture.



Of all the ways you can tell stories through web design, colour is the most fascinating and important. Not only does it evoke emotions in users but its meaning varies significantly between cultures. In the west, for example, white is a colour associated with weddings, and black is the colour of mourning. This is signified by the traditions of brides wearing white and those in mourning wearing black. In other cultures the meanings are reversed, as black is a colour that represents good luck and white is a colour that signifies mourning. If you assume the same values are true in all cultures then you risk offending the very people you are targeting.

When colours combine, the story being told can change. If you design using red, white and blue then it's going to be difficult to shake off patriotic connotations because this colour combination is so ingrained as being American or British or French thanks largely to their flags. This extends to politics too. Each party has its own representative colour. In the UK, the Conservatives are blue and Labour is red so it would be inappropriate storytelling to design a Labour-related website in blue as there would be a conflict between the content and the design, a conflict that would result in a poor user experience.

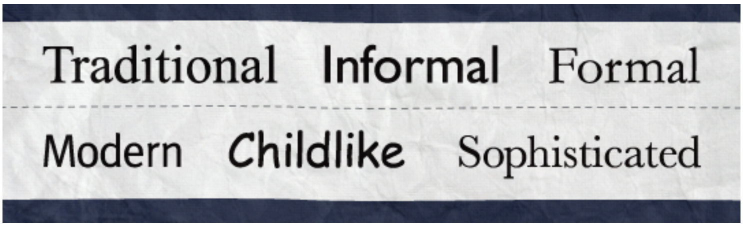
Conflicts become more of an issue when you start to combine story strands. I once saw a No VAT advert use the symbol on the left:



There's a complete conflict in storytelling here between the sign and its colour. The prohibition sign was used over the word VAT to mean no VAT; that makes sense. But this is a symbol that is used to communicate to people that something is being prohibited or prevented, it mustn't continue. So to use green contradicts the message of the sign itself; green is used as a colour to say yes, go, proceed, enter. The same would be true if we had a tick in red and a cross in green. Bad design here means the story is flawed and the user experience is compromised.

Typefaces

Typefaces also tell stories. They are so much more than the words that are written with them because they connote different values. Here are a few:



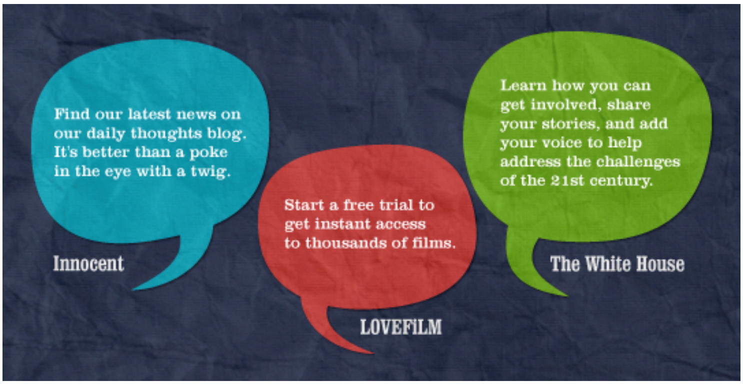
Serif fonts are more formal and are associated with tradition, sophistication and high-end values. Sans serif fonts, on the other hand, are synonymous with modernity, informality and friendliness. These perceptions are again reinforced through more traditional media such as newspaper mastheads, where the serious news-focused broadsheets have serif titles, and the showbiz and gossip-led tabloids have sans serif titles. This translates to the web as well. With these associations already familiar to users, they may see copy and focus on the words, but if the way that copy is displayed jars with the context then we are back to having conflicting stories like the No VAT sign earlier.

Let's take official institutions, for example. The White House, the monarchy, 10 Downing Street and other government departments are formal, traditional and important organisations. If the copy on their websites were written in a typeface like Cooper Black, it would erase any authority and respect that they were due. They need people to take them seriously and trust them, and part of the way to do this is to have a typeface that represents those values.

It works both ways though. If Innocent, Threadless or other fun companies used traditional typefaces, they wouldn't be accurate reflections of their core values, brand and personality. They are better positioned to use friendly, informal and modern typefaces. But still never Comic Sans.

Tone of voice

Closely tied to this is tone of voice, my absolute bugbear on the web. Tone of voice isn't what is said but, rather, how it is said. When we interact with others in person we don't just listen to the words they say, but we also draw meaning from their body language, and pitch and tone of voice. Just because the web removes that face-to-face interaction with your audience it doesn't mean you can't have a tone of voice.


Find our latest news on our daily thoughts blog. It's better than a poke in the eye with a twig.
Innocent
Start a free trial to get instant access to thousands of films.
LOVEFiLM
Learn how you can get involved, share your stories, and add your voice to help address the challenges of the 21st century.
The White House

Innocent pioneered the informal chatty tone of voice that so many others have since emulated, but unless it is representative of your company, then it's not appropriate. It works for Innocent because the tone of voice is consistent across all the company's materials, both online and offline. Ben and Jerry's takes the same approach, as does Threadless, but maybe you need a more formal or corporate tone of voice. It really depends on what your business or service is and who it is for, and that is why I think LoveFilm has it all wrong.

LoveFilm offers a film and game rental service, something fun for people in their downtime. While they aren't particularly stuffy, neither is their tone of voice very friendly or informal, which is what I would expect from a service like theirs. The reason they have it wrong is in the language they use and the way their sentences are constructed.

This is the first time we've discussed language because, on the whole, designing the invisible isn't concerned with language at all. But that doesn't mean that these strands can't still elicit an emotional response in users. Jon Tan quoted Dr Mazviita Chirimuuta in his New Adventures in Web Design talk in January 2011:

> Although there is no absolute separation between language and emotion, there will still be countless instances where you have emotional response without verbal input or linguistic cognition. In general language is not necessary for emotion.

This is even more pertinent when the emotions evoked are connected to people's culture, surroundings and way of life. It makes design personal, something that audiences can connect with at more than just face value but, rather, on a subliminal or, indeed, invisible level.

It also means that when you are asked the inevitable question of why – why is blue the dominant colour? why have you used that typeface? why don't we sound like Innocent? – you will have a rationale behind each design decision that can explain what story you are telling, how you discovered the story and how it is targeted at the core audience.

RESEARCH

This is where research plays a vital role in the project cycle. If you don’t know and understand your audience then you don’t know what story to design. Every project lends itself to some level of research, but how in-depth and what methods are most appropriate will be dictated by project requirements and budget restrictions – but do your research.

Even if you think you know your audience, it doesn’t hurt to research and reaffirm that because cultures and society do change, albeit slowly, but they can change. So ask questions at the start of the project during the research phase:

- What do different colours mean for your audience’s culture?
- Do the typeface and tone of voice appeal to the demographic?
- Does the brand identity represent the values and personality of your service?
- Are there any social, political or religious significances associated with your audience that you need to take into consideration so you don’t offend them?

Ask questions, understand your audience, design your story based on these insights, and create better user experiences in context that have good, solid storytelling at their heart.

Major hat tip to Gareth Strange for the beautiful graphics used within this article.

ABOUT THE AUTHOR

Robert Mills is Studio Manager at Bluegg and author of Designing the Invisible from Five Simple Steps. As a journalism graduate and former BBC Audience Researcher he likes words and data but mainly words.

In the day job Rob shares design decisions with clients and explains the rationale behind them. Aside from the general tasks needed to keep a studio ticking over he's also chief proof reader and content ambassador for internal work and client work.

You might also spot him in .Net Magazine as part of their big question panel, opinion piece writer or awards judge. You can also find him on Twitter. He will write for coffee or Monster Munch but never for raisins or yoghurts with bits in.

15. Extracting the Content

Relly Annett-Baker 24ways.org/201115

As we throw away our canvas in approaches and yearn for a content-out process, there remains a pain point: the Content. It is spoken of in the hushed tones usually reserved for Lord Voldemort. The-thing-that-someone-else-is-responsible-for-that-must-not-be-named.

Designers and developers have been burned before by not knowing what the Content is, how long it is, what style it is and when the hell it's actually going to be delivered, in internet eons past. Warily, they ask clients for it. But clients don't know what to make, or what is good, because no one taught them this in business school. Designers struggle to describe what they need and when, so the conversation gets put off until it's almost too late, and then everyone is relieved that they can take the cop-out of putting up a blog and maybe some product descriptions from the brochure.

THE CONTENT IN CONTENT OUT.

I'm guessing, as a smart, sophisticated, and, may I say, nicely-scented reader of the honourable and venerable tradition of 24 ways, that you sense something better is out there. Bunches of boxes to fill in just don't cut it any more in a responsive web design world. The first question is, how are you going to design something to ensure users have the easiest access to the best Content, if you haven't defined at the beginning what that Content is? Of course, it's more than possible that your clients have done lots of user research before approaching you to start this project, and have a plethora of finely tuned Content for you to design with.

Have you finished laughing yet? Alright then. Let's just assume that, for whatever reason of gross oversight, this hasn't happened. What next?

Bringing up Content for the first time with a client is like discussing contraception when you're in a new relationship. It might be awkward and either party would probably rather be doing something else, but it needs to be broached before any action happens (that, and it's disastrous to assume the other party has the matter in hand). If we can't talk about it, how can we expect people to be doing it right and not making stupid mistakes? That being the case, how do we talk about Content? Let's start by finding a way to talk about it without blushing and scuffing our shoes. And there's a reason I've been treating Content as a Proper Noun.

The first step, and I mean really-first-step-way-back-at-the-beginning-of-the-project-while-you-are-still-scoping-out-what-the-hell-you-might-do-for-each-other-and-it's-still-all-a-bit-awkward-like-a-first-date, is for you to explain to the client how important it is that you, together, work out what is important to your users as part of the user experience design, so that your users get the best user experience. The trouble is that, in most cases, this would lead to blank stares, possibly followed by a light cough and a query about using Comic Sans because it seems friendly.

Let's start by ensuring your clients understand the task ahead. You see, all the time we talk about the Content we do our clients a big disservice. Content is poorly defined. It looms over a project completion point like an unscalable (in the sense of a dozen stacked Kilimanjaros), seething, massive, singular entity. The Content.

DEFINING THE PROBLEM.

We should really be thinking of the Content as 'contents'; as many parts that come together to form a mighty experience, like hit 90s kids' TV show Mighty Morphin Power Rangers*.

*For those of you who might have missed the Power Rangers, they were five teenagers with attitude, each given crazy mad individual skillz and a coloured lycra suit from an alien overlord. In return, they had to fight a new monster of the week using their abilities and weaponry in sync (even if the audio was not) and then, finally, in thrilling combination as a Humongous Mechanoid Machine of Awesome. They literally joined their individual selves, accessories and vehicles into a big robot. It was a toy manufacturer's wet dream.

So, why do I say Content is like the Power Rangers? Because Content is not just a humongous mecha. It is a combination of well-crafted pieces of contents that come together to form a well-crafted humongous mecha. Of Content.

The Red Power Ranger was always the leader. You can imagine your text contents, found on about pages, product descriptions, blog articles, and so on, as being your Red Power Ranger.

Maybe your pictures are your Yellow Power Ranger; video is Blue (not used as much as the others, but really impressive when given a good storyline); maybe Pink is your infographics (it's wrong to find it sexier than the other equally important Rangers, but you kind of do anyway). And so on.

These bits of content – Red Text Ranger, Yellow Picture Ranger and others – often join together on a page, like they are teaming up to fight the bad guy in an action scene, and when they all come together (your standard workaday huge mecha) in a launched site, that's when Content becomes an entity.

While you might have a vision for the whole site, Content rarely works that way. Of course, you keep your eye on the bigger prize, the completion of your mega robot, but to get there you need to assemble your working parts, the cogs and springs of contents that will mesh together to finally create your Humongous Mecha of Content. You create parts and join them to form a whole. (It's rarely seamless; often we need to adjust as we go, but we can create our Mecha's blueprint by making sure we have all the requisite parts.)

The point here is the order these parts were created. No alien overlord plans a Humongous Mechanoid and then thinks, "Gee, how can I split this into smaller fighting units powered by teenagers in snazzy shiny suits?" No toy manufacturer goes into production of a mega robot, made up of model mecha vehicles with detachable arsenal, without thinking how they will easily fit back together to form the 'Buy all five now to create the mega robot' set. No good contents are created as a singular entity and chunked up to be slotted in to place any which way, into the body of a site.

Think contents, not the Content. Think of contents as smaller units, or as a plural. The Content is what you have at the end. The contents are what you are creating and they are easy to break down. You are no longer scaling the unscalable. You can draw the map and plot the path, page by page, section by section.

THE PAGE TABLE IS YOUR FRIEND

To do this, I use a page table. A page table is a simple table template you can create in the word processor of your choice, that you use to tell you everything about the contents of a page – everything except the contents itself.

Here's a page table I created for an employee's guide to redundancy in the alpha.gov.uk website:

Guide to redundancy for employees

- **Page objective**: Provide specific information for employees who are facing redundancy about the process, their options and next steps.
- **Source content**: directgov page on Redundancy.
- **Scope**: In scope

Page title	An employee's guide to redundancy
Priority content	**Message**: You have rights as an employee facing redundancy **Method**: A guide written in plain English, with links to appropriate additional content. A video guide (out of scope). Covers the stages of redundancy and rights for those in trade unions and not in trade unions. Glossary of unfamiliar terms. **Call to action**: Read full guide, act to explore redundancy actions, benefits or new employment. **Assets:** link to redundancy calculator.
Secondary	Related items, or popular additional links. Additional tools, such as search and suggestions. ▪ location set v not set states ▪ microcopy encouraging location set where location may make a difference to the content – ie, Scotland/Northern Ireland.
Tertiary	Footer and standard links.

- **Content creation**: Content exists but was created within the constraints of the previous CMS. Review, correct and edit where necessary.
- **Maintenance**: should be flagged for review upon advice from Department of Work and Pensions, and annually.

- **Technology/Publishing/Policy implications**: Should be reviewed once the glossary styles have been decided. No video guide in scope at this time, so languages should be simple and screen reader friendly.
- **Reliance on third parties**: None, all content and source exists in house.
- **Outstanding questions**: None.

Download a copy of this page table

This particular page table template owes a lot to Brain Traffic's version found in Kristina Halvorson's book *Content Strategy for the Web*. With smaller clients than, say, the government, I might use something a bit more casual. With clients who like timescales and deadlines, I might turn it into a covering sheet, with signatures and agreements from two departments who have to work together to get the piece done on time.

I use page tables, and the process of working through them, to reassure clients that I understand the task they face and that I can help them break it down section by section, page stack to page, down to product descriptions and interaction copy. About 80% of my clients break into relieved smiles. Most clients want to work with you to produce something good, they just don't understand how, and they want you to show them the mountain path on the map. With page tables, clients can understand that with baby steps they can break down their content requirements and commission content they need in time for the designers to work with it (as opposed to around it). If I was Santa, these clients would be on my nice list for sure.

My own special brand of Voldemort-content-evilness comes in how I wield my page tables for the other 20%. Page tables are not always thrilling, I'll admit. Sometimes they get ignored in favour of other things, yet they are crucial to the continual growth and maintenance of a truly content-led site. For these naughty list clients who, even when given the gift of the page table, continually say "Ooh, yes. Content. Right", I have a special gift. I have a stack of recycled paper under my desk and a cheap black and white laser printer. And I print a blank page table for every conceivable page I can find on the planned redesign. If I'm feeling extra nice, I hole punch them and put them in a fat binder.

There is nothing like saying, "This is all the contents you need to have in hand for launch", and the satisfying thud the binder makes as it hits the table top, to galvanize even the naughtiest clients to start working with you to create the content you need to really create in a content-out way.

ABOUT THE AUTHOR

Relly Annett-Baker lives in the Home Counties with her husband, Paul Annett, and their two small sons. As a result, she thrives on the country air and can be guaranteed to stand on Lego at least once a day. Her principle employment is as live-in domestic staff for two cats but when not being purred into submission she is a content strategist and writer, runs dedicated workshops in-house with companies big and small and continues to procrastinate over the draft of her Five Simple Steps book 'Content Creation for the Web' due out in 2012. She'll get right back to it just after she's had another cup of tea and checked her RSS feed.

16. CSS3 Patterns, Explained

Lea Verou 24ways.org/201116

Many of you have probably seen my CSS3 patterns gallery. It became very popular throughout the year and it showed many web developers how powerful CSS3 gradients really are. But how many really understand how these patterns are created? The biggest benefit of CSS-generated backgrounds is that they can be modified directly within the style sheet. This benefit is void if we are just copying and pasting CSS code we don't understand. We may as well use a data URI instead.

IMPORTANT NOTE

In all the examples that follow, I'll be using gradients without a vendor prefix, for readability and brevity. However, you should keep in mind that in reality you need to use all the vendor prefixes (`-moz-`, `-ms-`, `-o-`, `-webkit-`) as no browser currently implements them without a prefix. Alternatively, you could use -prefix-free and have the current vendor prefix prepended at runtime, only when needed.

The syntax described here is the one that browsers currently implement. The specification has since changed, but no browser implements the changes yet. If you are interested in what is coming, I suggest you take a look at the dev version of the spec.

If you are not yet familiar with CSS gradients, you can read these excellent tutorials by John Allsopp and return here later, as in the rest of the article I assume you already know the CSS gradient basics:

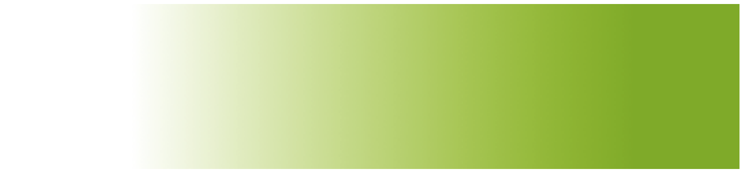
- CSS3 Linear Gradients
- CSS3 Radial Gradients

THE MAIN IDEA

I'm sure most of you can imagine the background this code generates:

```
background: linear-gradient(left, white 20%, #8b0 80%);
```

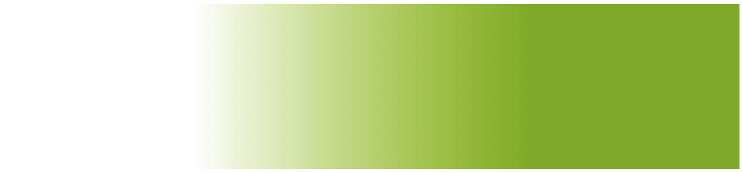
It's a simple gradient from one color to another that looks like this:



See this example live

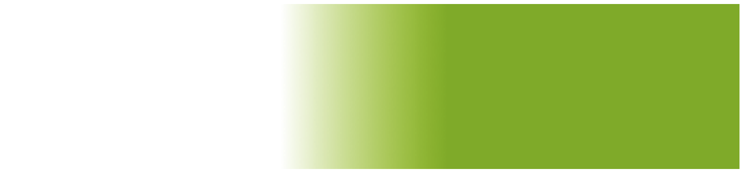
As you probably know, in this case the first 20% of the container's width is solid white and the last 20% is solid green. The other 60% is a smooth gradient between these colors. Let's try moving these color stops closer to each other:

```
background: linear-gradient(left, white 30%, #8b0 70%);
```



See this example live

```
background: linear-gradient(left, white 40%, #8b0 60%);
```



See this example live

```
background: linear-gradient(left, white 50%, #8b0 50%);
```

See this example live

Notice how the gradient keeps shrinking and the solid color areas expanding, until there is no gradient any more in the last example. We can even adjust the position of these two color stops to control where each color abruptly changes into another:

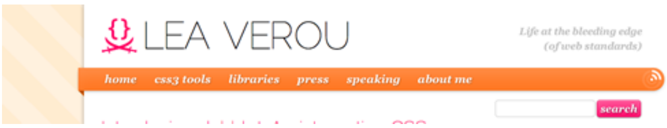
```
background: linear-gradient(left, white 30%, #8b0 30%);
```

See this example live

```
background: linear-gradient(left, white 90%, #8b0 90%);
```

See this example live

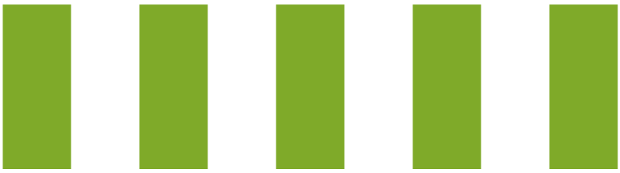
What you need to take away from these examples is that when two color stops are at the same position, there is no gradient, only solid colors. Even without going any further, this trick is useful for a number of different use cases like faux columns or the effect I wanted to achieve in my homepage or the -prefix-free page where the background is only shown on one side and hidden on the other:



COMBINING WITH BACKGROUND-SIZE

We can do wonders, however, if we combine this with the CSS3 `background-size` property:

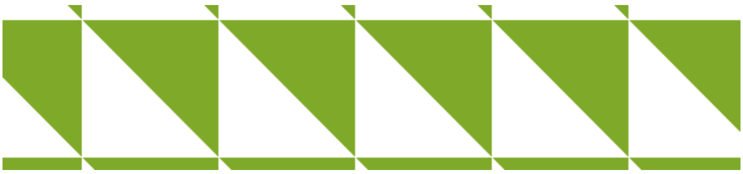
```
background: linear-gradient(left, white 50%, #8b0 50%);
background-size: 100px 100px;
```



See this example live

And there it is. We just created the simplest of patterns: (vertical) stripes. We can remove the first parameter (`left`) or replace it with `top` and we'll get horizontal stripes. However, let's face it: Horizontal and vertical stripes are kinda boring. Most stripey backgrounds we see on the web are diagonal. So, let's try doing that.

Our first attempt would be to change the angle of the gradient to something like `45deg`. However, this results in an ugly pattern like this:



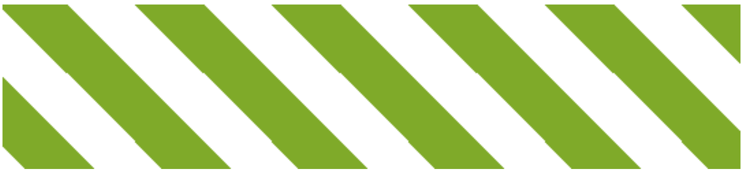
See this example live

Before reading on, think for a second: why didn't this produce the desired result? Can you figure it out?

The reason is that the gradient angle rotates the gradient **inside each tile**, not the tiled background as a whole. However, didn't we have the same problem the first time we tried to create diagonal stripes with an image? And then we learned that every stripe has to be included twice, like so:

So, let's try to create that effect with CSS gradients. It's essentially what we tried before, but with more color stops:

```
background: linear-gradient(45deg, white 25%,
    #8b0 25%, #8b0 50%,
    white 50%, white 75%,
    #8b0 75%);
background-size:100px 100px;
```

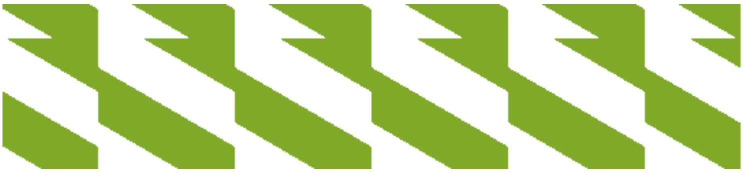


See this example live

And there we have our stripes! An easy way to remember the order of the percentages and colors it is that you always have two of the same in succession, except the first and last color.

Note: Firefox for Mac also needs an additional 100% color stop at the end of any pattern with more than two stops, like so: `..., white 75%, #8b0 75%, #8b0)`. The bug was reported in February 2011 and you can vote for it and track its progress at Bugzilla.

Unfortunately, this is essentially a hack and we will realize that if we try to change the gradient angle to `60deg`:



See this example live

Not that maintainable after all, eh? Luckily, CSS3 offers us another way of declaring such backgrounds, which not only helps this case but also results in much more concise code:

```
background: repeating-linear-gradient(60deg, white,
white 35px, #8b0 35px, #8b0 70px);
```

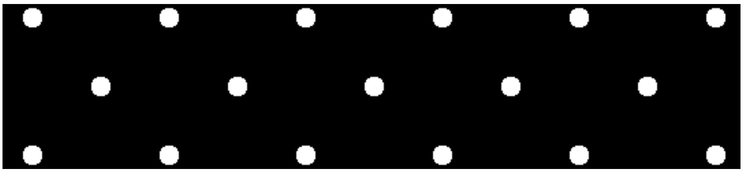
See this example live

In this case, however, the size has to be declared in the color stop positions and not through `background-size`, since the gradient is supposed to cover the entire container. You might notice that the declared size is different from the one specified the previous way. This is because the size of the stripes is measured differently: in the first example we specify the dimensions of the tile itself; in the second, the width of the stripes (35px), which is measured diagonally.

MULTIPLE BACKGROUNDS

Using only one gradient you can create stripes and that's about it. There are a few more patterns you can create with just one gradient (linear or radial) but they are more or less boring and ugly. Almost every pattern in my gallery contains a number of different backgrounds. For example, let's create a polka dot pattern:

```
background: radial-gradient(circle, white 10%,
transparent 10%),
radial-gradient(circle, white 10%, black 10%) 50px 50px;
background-size:100px 100px;
```



See this example live

Notice that the two gradients are almost the same image, but positioned differently to create the polka dot effect. The only difference between them is that the first (topmost) gradient has `transparent` instead of `black`. If it didn't have transparent regions, it would effectively be the same as having a single gradient, as the topmost gradient would obscure everything beneath it.

There is an issue with this background. Can you spot it?

This background will be fine for browsers that support CSS gradients but, for browsers that don't, it will be transparent as the whole declaration is ignored. We have two ways to provide a fallback, each for different use cases. We have to either declare another background before the gradient, like so:

```
background: black;
background: radial-gradient(circle, white 10%,
transparent 10%),
radial-gradient(circle, white 10%, black 10%) 50px 50px;
background-size:100px 100px;
```

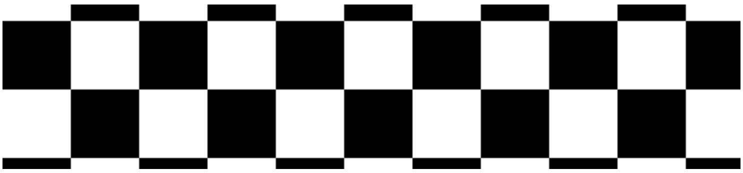
or declare each background property separately:

```
background-color: black;
background-image: radial-gradient(circle, white 10%,
transparent 10%),
radial-gradient(circle, white 10%, transparent 10%);
background-size:100px 100px;
background-position: 0 0, 50px 50px;
```

The vigilant among you will have noticed another change we made to our code in the last example: we altered the second gradient to have transparent regions as well. This way `background-color` serves a dual purpose: it sets both the fallback color and the background color of the polka dot pattern, so that we can change it with just one edit. Always strive to make code that can be modified with the least number of edits. You might think that it will never be changed in that way but, almost always, given enough time, you'll be proved wrong.

We can apply the exact same technique with linear gradients, in order to create checkerboard patterns out of right triangles:

```
background-color: white;
background-image: linear-gradient(45deg, black 25%,
transparent 25%, transparent 75%, black 75%),
linear-gradient(45deg, black 25%, transparent 25%,
transparent 75%, black 75%);
background-size:100px 100px;
background-position: 0 0, 50px 50px;
```


See this example live

USING THE RIGHT UNITS

Don't use pixels for the sizes without any thought. In some cases, ems make much more sense. For example, when you want to make a lined paper background, you want the lines to actually follow the text. If you use pixels, you have to change the size every time you change `font-size`. If you set the `background-size` in ems, it will naturally follow the text and you will only have to update it if you change `line-height`.

IS IT POSSIBLE?

The shapes that can be achieved with only one gradient are:

- stripes
- right triangles
- circles and ellipses
- semicircles and other shapes formed from slicing ellipses horizontally or vertically

You can combine several of them to create squares and rectangles (two right triangles put together), rhombi and other parallelograms (four right triangles), curves formed from parts of ellipses, and other shapes.

JUST BECAUSE YOU CAN DOESN'T MEAN YOU SHOULD

Technically, anything can be crafted with these techniques. However, not every pattern is suitable for it. The main advantages of this technique are:

- no extra HTTP requests
- short code
- human-readable code (unlike data URIs) that can be changed without even leaving the CSS file.

Complex patterns that require a large number of gradients are probably better left to SVG or bitmap images, since they negate almost every advantage of this technique:

- they are not shorter
- they are not really comprehensible – changing them requires much more effort than using an image editor

They still save an HTTP request, but so does a data URI.

I have included some very complex patterns in my gallery, because even though I think they shouldn't be used in production (except under very exceptional conditions), understanding how they work and coding them helps somebody understand the technology in much more depth.

Another rule of thumb is that if your pattern needs shapes to obscure parts of other shapes, like in the star pattern or the yin yang pattern, then you probably shouldn't use it. In these patterns, changing the background color requires you to also change the color of these shapes, making edits very tedious.

If a certain pattern is not practicable with a reasonable amount of CSS, that doesn't mean you should resort to bitmap images. SVG is a very good alternative and is supported by all modern browsers.

BROWSER SUPPORT

CSS gradients are supported by Firefox 3.6+, Chrome 10+, Safari 5.1+ and Opera 11.60+ (linear gradients since Opera 11.10). Support is also coming in Internet Explorer when IE10 is released. You can get gradients in older WebKit versions (including most mobile browsers) by using the proprietary `-webkit-gradient()`, if you really need them.

EPILOGUE

I hope you find these techniques useful for your own designs. If you come up with a pattern that's very different from the ones already included, especially if it demonstrates a cool new technique, feel free to send a pull request to the github repo of the patterns gallery.

Also, I'm always fascinated to see my techniques put in practice, so if you made something cool and used CSS patterns, I'd love to know about it!

Happy holidays!

ABOUT THE AUTHOR

Lea Verou is the lead web developer and designer of Fresset Ltd, which she co-founded in 2008. Fresset owns and manages some of the largest Greek community websites. Lea has a long-standing passion for open web standards, especially CSS and JavaScript. She loves researching new ways to use them and shares her findings through her blog, lea.verou.me. She speaks at a number of the largest web development conferences and writes for leading industry publications. Lea also co-organized and occasionally lectures the web development course at the Athens University of Economics and Business.

17. Designing for Perfection

Greg Wood 24ways.org/201117

Hello, 24 ways readers. I hope you're having a nice run up to Christmas. This holiday season I thought I'd share a few things with you that have been particularly meaningful in my work over the last year or so. They may not make you wet your santa pants with new-idea-excitement, but in the context of 24 ways I think they may serve as a nice lesson and a useful seasonal reminder going into the New Year. Enjoy!

STORY

Despite being a largely scruffy individual for most of my life, I had some interesting experiences regarding kitchen tidiness during my third year at university.

As a kid, my room had always been pretty tidy, and as a teenager I used to enjoy reordering my CDs regularly (by artist, label, colour of spine – you get the picture); but by the time I was twenty I'd left most of these traits behind me, mainly due to a fear that I was turning into my mother. The one remaining anally retentive part of me that remained however, lived in the kitchen. For some reason, I couldn't let all the pots and crockery be strewn across the surfaces after cooking. I didn't care if they were washed up or not, I just needed them tidied. The surfaces needed to be continually free of grated cheese, breadcrumbs and ketchup spills. Also, the sink always needed to be clear. Always. Even a lone teabag, discarded casually into the sink hours previously, would give me what I used to refer to as "kitchen rage".

Whilst this behaviour didn't cause any direct conflicts, it did often create weirdness. We would be happily enjoying a few pre-night out beverages (Jack Daniels and Red Bull – nice) when I'd notice the state of the kitchen following our round of customized 49p Tesco pizzas. Kitchen rage would ensue, and I'd have to blitz the kitchen, which usually resulted in me having to catch everyone up at the bar afterwards.

One evening as we were just about to go out, I was stood there, in front of the shithole that was our kitchen with the intention of cleaning it all up, when a realization popped into my head. In hindsight, it was a pretty obvious one, but it went along the lines of "What the fuck are you doing? Sort your life out". I sodded the washing up, rolled out with my friends, and had a badass evening of partying.

After this point, whenever I got the urge to clean the kitchen, I repeated that same realization in my head. My tidy kitchen obsession strived for a level of perfection that my housemates just didn't share, so it was ultimately pointless. It didn't make me feel that good, either; it was like having a cigarette after months of restraint – initially joyous but soon slightly shameful.

LESSON

Now, around seven years later, I'm a designer on the web and my life is chaotic. It features no planning for significant events, no day-to-day routine or structure, no thought about anything remotely long-term, and I like to think I do precisely what I want. It seems my days at striving for something ordered and tidy, in most parts of my life, are long gone.

For much of my time as a designer, though, it's been a different story. I relished industry-standard terms such as 'pixel perfection' and 'polished PSDs', taking them into my stride as I strove to design everything that was put on my plate perfectly. Even down to grids and guidelines, all design elements would be painstakingly aligned to a five-pixel grid. There were no seven-pixel margins or gutters to be found in my design work, that's for sure. I put too much pride and, inadvertently, too much ego into my work. Things took too long to create, and because of the amount of effort put into the work, significant changes, based on client feedback for example, were more difficult to stomach.

Over the last eighteen months I've made a conscious effort to change the way I approach designing for the web. Working on applications has probably helped with this; they seem to have a more organic development than rigid content-based websites. Mostly though, a realization similar to my kitchen rage one came about when I had to make significant changes to a painstakingly crafted Photoshop document I had created. The changes shouldn't have been difficult or time-consuming to implement, but they were turning out to be. One day, frustrated with how long it was taking, the refrain "What the fuck are you doing? Sort your life out" again entered my head. I blazed the rest of the work, not rushing or doing scruffy work, but just not adhering to the insane levels of perfection I had previously set for myself. When the changes were presented, everything went down swimmingly. The client in this case (and I'd argue most cases) cared more about the ideas than the perfect way in which they had been implemented. I had taken myself and my ego out of the creative side of the work, and it had been easier to succeed.

ARGUMENT

I know many other designers who work on the web share such aspirations to perfection. I think it's a common part of the designer DNA, but I'm not sure it really has a place when designing for the web.

First, there's the environment. The landscape in which we work is continually shifting and evolving. The inherent imperfection of the medium itself makes attempts to create perfect work for it redundant. Whether you consider it a positive or negative point, the products we make are never complete. They're always scaling and changing.

Like many aspects of web design, this striving for perfection in our design work is a way of thinking borrowed from other design industries where it's more suited. A physical product cannot be as easily altered or developed after it has been manufactured, so the need to achieve perfection when designing is more apt.

Designers who can relate to anything I've talked about can easily let go of that anal retentiveness if given the right reasons to do so. Striving for perfection isn't a bad thing, but I simply don't think it can be achieved in such a fast-moving, unique industry. I think design for the web works better when it begins with quick and simple, followed by iteration and polish over time.

To let go of ego and to publish something that you’re not completely happy with is perhaps the most difficult part of the job for designers like us, but it’s followed by a satisfaction of knowing your product is alive and breathing, whereas others (possibly even competitors) may still be sitting in Photoshop, agonizing over whether a margin should be twenty or forty pixels.

I keep telling myself to stop sitting on those two hundred ideas that are all half-finished. Publish them, clean them up and iterate over time. I’ve been telling myself this for months and, hopefully, writing this article will give me the kick in the arse I need. Hopefully, it will also give someone else the same kick.

ABOUT THE AUTHOR

Greg Wood is a designer based in Nottingham, England. He specialises in designing websites and online applications. He has a journal thing, which hasn't been updated in almost a year. He'll get back round to it one day. He is currently working on some nifty stuff, including an ambitious music application and the New Adventures in Web Design conference.

18. Getting the Most Out of Google Analytics

Matt Curry 24ways.org/201118

Something a bit different for today's 24 ways article. For starters, I'm not a designer or a developer. I'm an evil man who sells things to people on the internet. Second, this article will likely be a little more nebulous than you're used to, since it covers quite a number of points in a relatively short space.

This isn't going to be the complete Google Analytics Conversion University IQ course compressed into a single article, obviously. What it will be, however, is a primer on setting up and using Google Analytics in real life, and a great deal of what I've learned using Google Analytics nearly every working day for the past six (crikey!) years.

Also, to be clear, I'll be referencing new Google Analytics here; old Google Analytics is for loooosers (and those who want reliable e-commerce conversion data per site search term, natch).

You may have been running your Analytics account for several years now, dipping in and out, checking traffic levels, seeing what's popular… and that's about it. Google Analytics provides so much more than that, but the number of reports available can often intimidate users, and documentation and case studies on their use are minimal at best.

LET'S START! SETTING UP YOUR ANALYTICS PROFILE

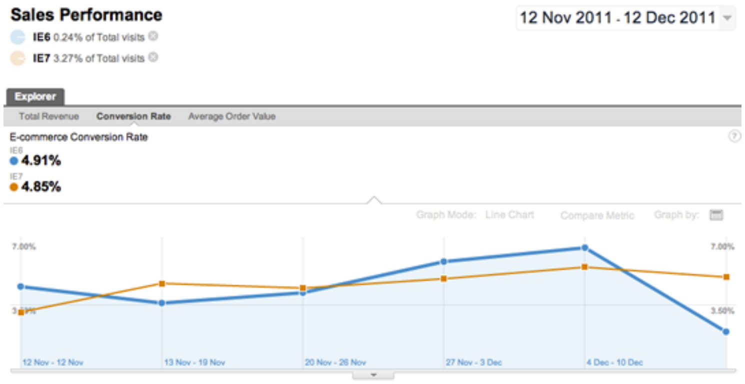
Before we plough on, I just want to run through a quick checklist that some basic settings have been enabled for your profile. If you haven't clicked it, click the big cog on the top-right of Google Analytics and we'll have a poke about.

1. If you have an e-commerce site, e-commerce tracking has been enabled
2. If your site has a search function, site search tracking has been enabled.

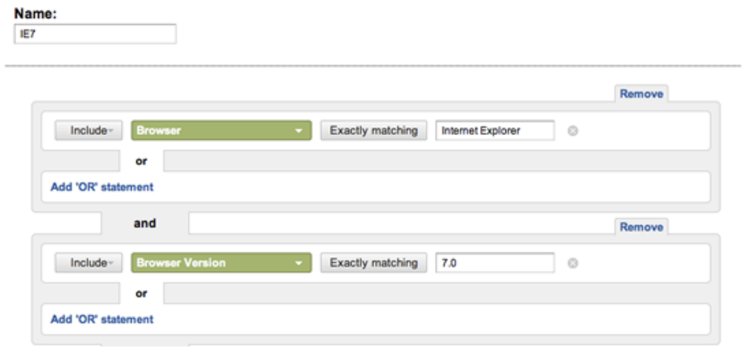
3. Query string parameters that you do not want tracked as separate pages have been excluded (for example, any parameters needed for your platform to function, otherwise you'll get multiple entries for the same page appearing in your reports)
4. Filters have been enabled on your main profile to exclude your office IP address and any IPs of people who frequently access the site for work purposes. In decent numbers they tend to throw data off a tad.
5. You may also find the need to set up multiple profiles prefiltered for specific audience segments. For example, at Lovehoney we have seventeen separate profiles that allow me quick access to certain countries, devices and traffic sources without having to segment first. You'll also find load time for any complex reports much improved. Use the same filter screen as above to set up a series of profiles that only include, say, mobile visits, or UK visitors, so you can quickly analyse important segments.

MATT, WHAT'S A SEGMENT?

A segment is a subsection of your visitor base, which you define and then call on in reports to see specific data for that subsection. For example, in this report I've defined two segments, the first for IE6 users and the second for IE7.



Segments are easily created by clicking the Advanced Segments tabs at the top of any report and clicking +New Custom Segment.


Name: IE7

Include · Browser · Exactly matching · Internet Explorer

and

Include · Browser Version · Exactly matching · 7.0

WHAT DOES YOUR SITE DO?

Understanding the goals of your site is an oft-covered topic, but it's necessary not just to form a better understand of your business and prioritize your time.

Understanding what you wish visitors to do on your site translates well into a goal-driven analytics package like Google Analytics.

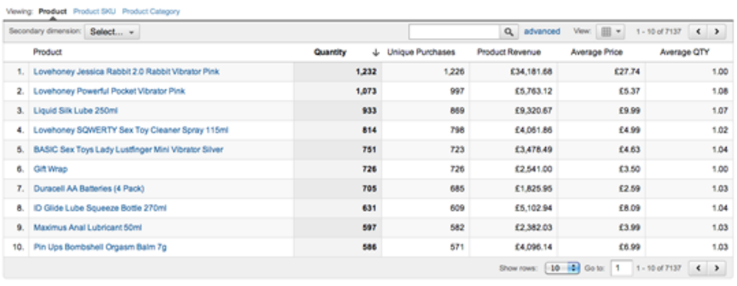
Every site exists essentially to sell something, either financially through e-commerce, or to sell an idea or impart information, get people to download a CV or enquire about service, or to sell space on that website to advertisers. If the site did not provide a positive benefit to its owners, it would not have a reason for being.

Once you have understood the reason why you have a site, you can map that reason on to one of the three goal types Google Analytics provides.

E-commerce

This conversion type registers transactions as part of a sales process which requires a monetary value, what products have been bought, an SKU (stock keeping unit), affiliation (if you're then attributing the sale to a third party or franchise) and so on.

The benefit of e-commerce tracking is not only assigning non-arbitrary monetary value to behaviour of visitors on your site, as well as being able to see ancillary costs such as shipping, but seeing product-level information, like which products are preferred from various channels, popular categories, and so on.



Viewing: **Product** Product SKU Product Category

Secondary dimension: Select... | advanced | View: | 1 - 10 of 7137

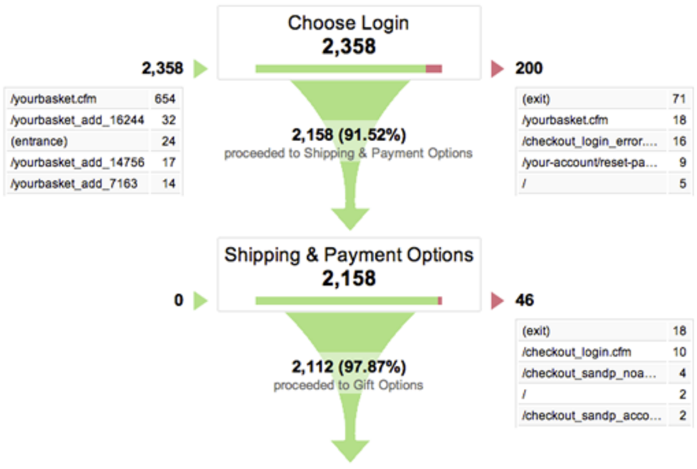
	Product	Quantity ↓	Unique Purchases	Product Revenue	Average Price	Average QTY
1.	Lovehoney Jessica Rabbit 2.0 Rabbit Vibrator Pink	**1,232**	1,226	£34,181.68	£27.74	1.00
2.	Lovehoney Powerful Pocket Vibrator Pink	**1,073**	997	£5,763.12	£5.37	1.08
3.	Liquid Silk Lube 250ml	**933**	869	£9,320.67	£9.99	1.07
4.	Lovehoney SQWERTY Sex Toy Cleaner Spray 115ml	**814**	798	£4,061.86	£4.99	1.02
5.	BASIC Sex Toys Lady Lustfinger Mini Vibrator Silver	**751**	723	£3,478.49	£4.63	1.04
6.	Gift Wrap	**726**	726	£2,541.00	£3.50	1.00
7.	Duracell AA Batteries (4 Pack)	**705**	685	£1,825.95	£2.59	1.03
8.	ID Glide Lube Squeeze Bottle 270ml	**631**	609	£5,102.94	£8.09	1.04
9.	Maximus Anal Lubricant 50ml	**597**	582	£2,382.03	£3.99	1.03
10.	Pin Ups Bombshell Orgasm Balm 7g	**586**	571	£4,096.14	£6.99	1.03

Show rows: 10 Go to: 1 1 - 10 of 7137

However, I find the e-commerce tracking options also useful for non-e-commerce sites. For example, if you're offering downloads or subscriptions and having an email address or user's details is worth something to you, you can set up e-commerce tracking to understand how much value your site is bringing. For example, an email address might be worth 20p to you, but if it also includes a name it's worth 50p. A contact telephone number is worth £2, and so on.

Page goals

Page goals, unsurprisingly, track a visit to a page (often with a sequence of pages leading up to that page). This is what's referred to as a goal funnel, and is generally used to track how visitors behave in a multistep checkout.


Choose Login
2,358
2,358
200
2,158 (91.52%)
proceeded to Shipping & Payment Options
Shipping & Payment Options
2,158
0
46
2,112 (97.87%)
proceeded to Gift Options

/yourbasket.cfm	654
/yourbasket_add_16244	32
(entrance)	24
/yourbasket_add_14756	17
/yourbasket_add_7163	14

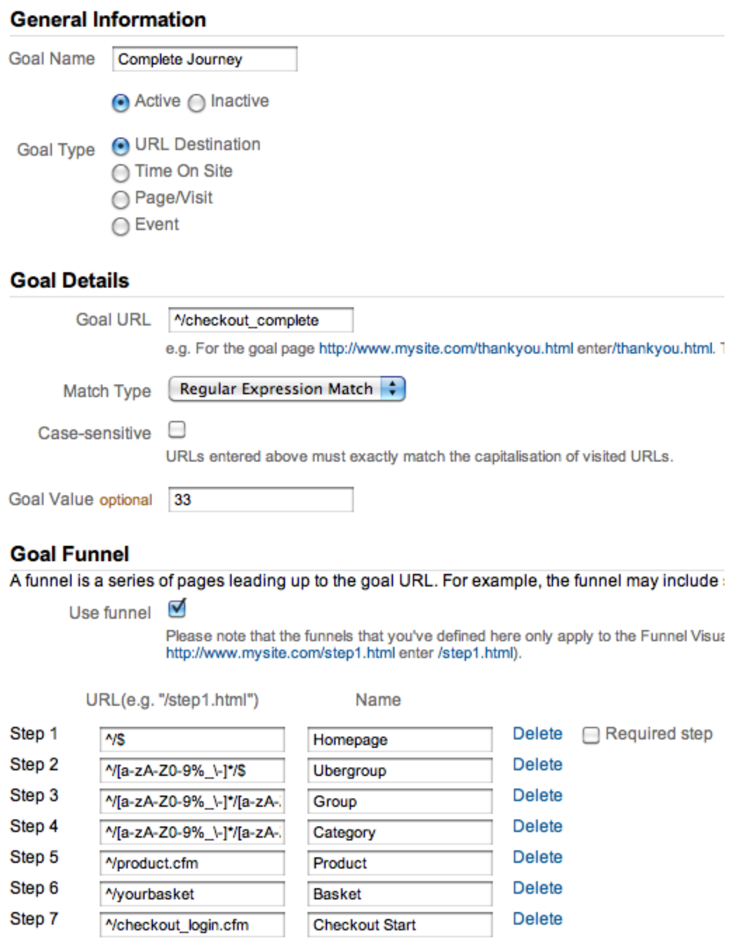
(exit)	71
/yourbasket.cfm	18
/checkout_login_error....	16
/your-account/reset-pa...	9
/	5

(exit)	18
/checkout_login.cfm	10
/checkout_sandp_noa...	4
/	2
/checkout_sandp_acco...	2

Interestingly, the page doesn't have to actually exist. For example, if you have a single page checkout, you can register virtual page views using `trackPageview()` when a visitor clicks into a particular section of the checkout or other form. If your site is geared towards getting someone to a particular page, but where there isn't a transaction (for example, a subscription page) this is for you.

There are also behavioural goals, such as time on site and number of pages viewed, which are geared towards sites that make money from advertising.

But, going back to the page goals, these can be abstracted using regular expressions, meaning that you can define a funnel based on page type rather than having to set individual folders.



General Information

Goal Name: Complete Journey

[✓ Active] Inactive

Goal Type: [✓ URL Destination] Time On Site Page/Visit Event

Goal Details

Goal URL: ^/checkout_complete

e.g. For the goal page http://www.mysite.com/thankyou.html enter/thankyou.html. T

Match Type: Regular Expression Match

Case-sensitive: ☐

URLs entered above must exactly match the capitalisation of visited URLs.

Goal Value optional: 33

Goal Funnel

A funnel is a series of pages leading up to the goal URL. For example, the funnel may include :

Use funnel: ✓

Please note that the funnels that you've defined here only apply to the Funnel Visua
http://www.mysite.com/step1.html enter /step1.html).

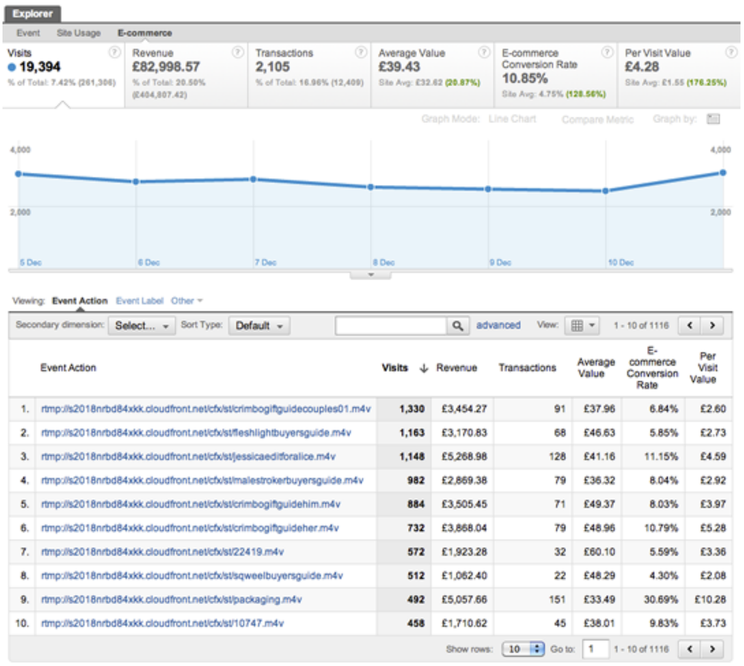
	URL(e.g. "/step1.html")	Name		
Step 1	^/$	Homepage	Delete	☐ Required step
Step 2	^/[a-zA-Z0-9%_\-]*/$	Ubergroup	Delete	
Step 3	^/[a-zA-Z0-9%_\-]*/[a-zA-	Group	Delete	
Step 4	^/[a-zA-Z0-9%_\-]*/[a-zA-	Category	Delete	
Step 5	^/product.cfm	Product	Delete	
Step 6	^/yourbasket	Basket	Delete	
Step 7	^/checkout_login.cfm	Checkout Start	Delete	

In this example, I've created regexes for the main page types on my site, so I can create a wide funnel that captures visitors from where they enter through to checkout.

Events

Event tracking registers a predefined event, such as playing a video, or some interaction that can trigger JavaScript, such as a Tweet This button. Events can then be triggered using the `trackEvent()` call. If you want someone to complete watching a video, you would code your player to fire `trackEvent()` upon completion.

While I don't use events as goals, I use events elsewhere to see how well a video play aids to conversion. This not only helps me justify the additional spend on creating video content, but also quickly highlights which videos are underperforming as sales tools.



Visits	Revenue	Transactions	Average Value	E-commerce Conversion Rate	Per Visit Value
19,394	£82,998.57	2,105	£39.43	10.85%	£4.28
% of Total: 7.42% (261,306)	% of Total: 20.50% (£404,807.42)	% of Total: 16.96% (12,409)	Site Avg: £32.62 (20.87%)	Site Avg: 4.75% (128.56%)	Site Avg: £1.55 (176.25%)

	Event Action	Visits ↓	Revenue	Transactions	Average Value	E-commerce Conversion Rate	Per Visit Value
1.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/crimbogiftguidecouples01.m4v	1,330	£3,454.27	91	£37.96	6.84%	£2.60
2.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/fleshlightbuyersguide.m4v	1,163	£3,170.83	68	£46.63	5.85%	£2.73
3.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/jessicaeditforalice.m4v	1,148	£5,268.98	128	£41.16	11.15%	£4.59
4.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/malestrokerbuyersguide.m4v	982	£2,869.38	79	£36.32	8.04%	£2.92
5.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/crimbogiftguidehim.m4v	884	£3,505.45	71	£49.37	8.03%	£3.97
6.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/crimbogiftguideher.m4v	732	£3,868.04	79	£48.96	10.79%	£5.28
7.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/22419.m4v	572	£1,923.28	32	£60.10	5.59%	£3.36
8.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/sqweelbuyersguide.m4v	512	£1,062.40	22	£48.29	4.30%	£2.08
9.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/packaging.m4v	492	£5,057.66	151	£33.49	30.69%	£10.28
10.	rtmp://s2018nrbd84xkk.cloudfront.net/cfx/st/10747.m4v	458	£1,710.62	45	£38.01	9.83%	£3.73

WHAT A VISITOR CAN TELL YOU

Now you have some proper goals set up, we can start to see how changes in content (on-site and external) affect those goals.

Ultimately, when a visitor comes to your site, they bring information with them:

- where they came from (a search engine – including: keyword searched for; a referral; direct; affiliate; or ad campaign)

- demographics (country; whether they're new or returning, within thirty days)
- technical information (browser; screen size; device; bandwidth)
- site-specific information (landing page; next click; previous values assigned to them as custom variables*)

* A note about custom variables. There's no hope in hell that I can cover custom variables in this article. Go research them. Custom variables are the single best way to hack Google Analytics and bend it to your will. Custom variables allow you to record anything you want about a visitor, which that visitor will then carry around with them between visits. It's also great for plugging other services into Google Analytics (as shown by the marvelous way Visual Website Optimizer allows you to track and segment tests within the GA interface). Just make sure not to breach the terms of service, eh?

CSI YOUR WEBSITE

Police procedural TV shows are all the same: the investigators are called to a crime and come across a clue; there's then an autopsy; new evidence leads them to a new location; they find a new clue; they put two and two together; they solve the mystery.

This is your life now. **Exciting!**

So, now you're gathering a wealth of information about what sort of people visit your site, what they do when they're there, and what eventually gets them to drive value to you. It’s now your job to investigate all these little clues to see which types of people drive the most value, and what you can change to improve it.

Maybe not that exciting.

However, Google Analytics comes pre-armed with extensive reports for you to delve into. As an e-commerce guy (as opposed to a page goal guy) my day pretty much follows the pattern below.

1. Look at e-commerce conversion rate by traffic source compared to the same day in the previous week and previous month. As ours is an e-commerce site, we have weekly and monthly trends. A big spike on Sundays and Mondays, and payday towards the end of the month is always good; on the third week of a month there tends to be a lull. Spend time letting your Google Analytics data brew, understand your own trends and patterns, and you’ll start to get a feel for when something isn’t quite right.
   - **Traffic Sources → Sources → All Traffic**

2. Look at the conversion rate by landing page for any traffic source that feels significantly different to what’s expected. Check bounce rates, drill down to likely landing pages and check search keyword or referral site to see if it’s a particular subset of visitor. You can do this by

clicking Secondary Dimension and choosing Keyword or Source. If it's direct, choose Visitor Type to break down by new or returning visitor.

- **Content → Site Content → Landing Pages**

3. I then tend to flip into Content Drilldown to see what the next clicks were from those landing pages, and whether they changed significantly to the date I'm comparing with. If they have, that's usually an indicator of changed content (or its relevancy). Remember, if a bunch of people have found their way to your page via a method you're not expecting (such as a mention on a Spanish radio station – this actually happened to me once), while the content hasn't changed, the relevancy of it to the audience may have.

- **Content → Site Content → Content Drilldown**

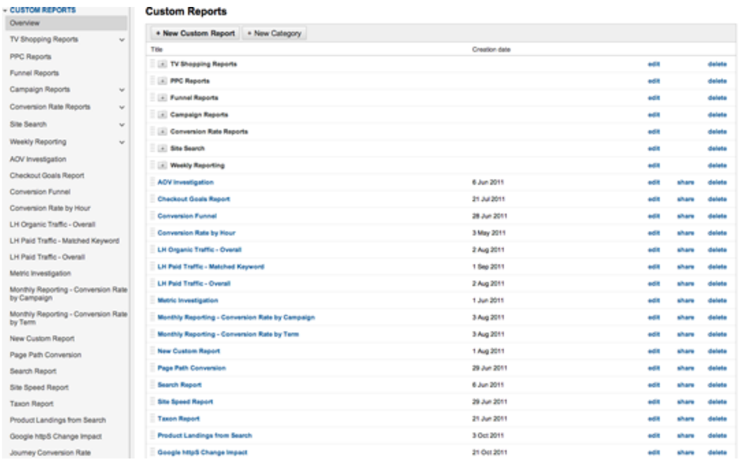
4. Once I have an idea of what content was consumed, and whether it was relevant to the user, I then look at the visitor specifics, such as browser or demographic data, to see again whether the change was limited to a specific subset. Site speed, for example, is normally a good factor towards bounce rate, so compare that with previous data as well.

Now, to be investigating at this level you still need a serious amount of data, in order to tell what's a significant change or not. If you're struggling with a small number of visitors, you might find reporting on a weekly or fortnightly basis more appropriate.

However, once you've looked into the basics of why changes happen to the value of your site, you'll soon find yourself limited by the reports offered in Standard Reporting. So, it's time to build your own. Hooray!

CUSTOM REPORTING

Google Analytics provides the tools to build reports specific to the types of investigations you frequently perform.



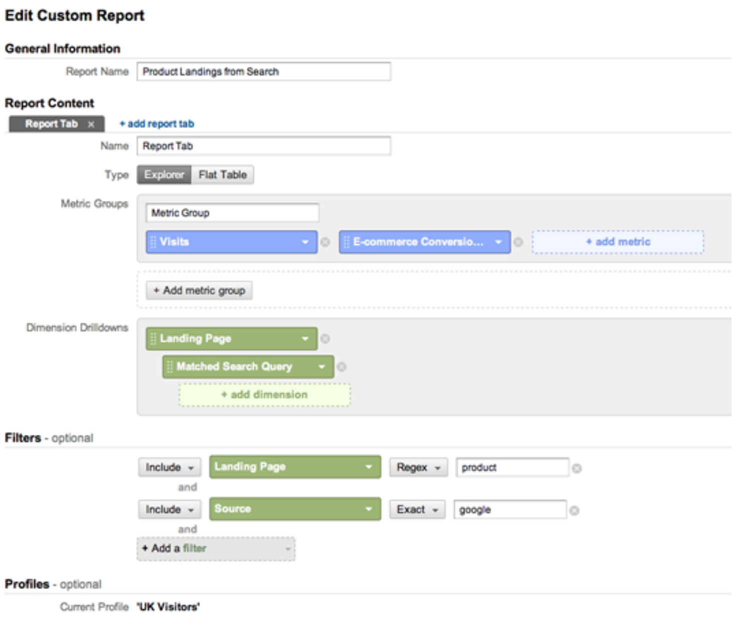
Title	Creation date			
TV Shopping Reports		edit		delete
PPC Reports		edit		delete
Funnel Reports		edit		delete
Campaign Reports		edit		delete
Conversion Rate Reports		edit		delete
Site Search		edit		delete
Weekly Reporting		edit		delete
AOV Investigation	6 Jun 2011	edit	share	delete
Checkout Goals Report	21 Jul 2011	edit	share	delete
Conversion Funnel	28 Jun 2011	edit	share	delete
Conversion Rate by Hour	3 May 2011	edit	share	delete
LH Organic Traffic - Overall	2 Aug 2011	edit	share	delete
LH Paid Traffic - Matched Keyword	1 Sep 2011	edit	share	delete
LH Paid Traffic - Overall	2 Aug 2011	edit	share	delete
Metric Investigation	1 Jun 2011	edit	share	delete
Monthly Reporting - Conversion Rate by Campaign	3 Aug 2011	edit	share	delete
Monthly Reporting - Conversion Rate by Term	3 Aug 2011	edit	share	delete
New Custom Report	1 Aug 2011	edit	share	delete
Page Path Conversion	29 Jun 2011	edit	share	delete
Search Report	6 Jun 2011	edit	share	delete
Site Speed Report	29 Jun 2011	edit	share	delete
Taxon Report	21 Jun 2011	edit	share	delete
Product Landings from Search	3 Oct 2011	edit	share	delete
Google httpS Change Impact	21 Oct 2011	edit	share	delete

Welcome to my world.

Custom reports are quite simple to build: first, you determine the metric you want the report to cover (number of visitors, bounce rate, conversion rate, and so on), then choose a set of dimensions that you'd like to segment the report by (say, the source of the traffic, and whether they were new or returning users). You can filter the report, including or excluding particular dimension values, and you can assign the report to any of the profiles you created earlier.

In the example below, I've created a report that shows me visits and conversion rate for any Google traffic that landed directly only on a product page. I can then drill down on each product page to see the complete phrases use to search. I can use this information in two ways:

1. I can see which products aren't converting, which shows me where I need to work harder on merchandising.
2. I can give this information to my content team, showing them the actual phrases visitors used to reach our product content, helping them write better targeted product descriptions.



Edit Custom Report

General Information

Report Name: Product Landings from Search

Report Content

Report Tab × + add report tab

Name: Report Tab

Type: Explorer | Flat Table

Metric Groups: Metric Group — Visits, E-commerce Conversio... + add metric

+ Add metric group

Dimension Drilldowns: Landing Page, Matched Search Query, + add dimension

Filters - optional

Include Landing Page Regex product

and

Include Source Exact google

and

+ Add a filter

Profiles - optional

Current Profile 'UK Visitors'

The possibilities here are nearly endless, but here are a few examples of reports I find useful:

1. **Non-brand inbound search**
By creating a report that shows inbound search traffic which doesn't include your brand, you can see more clearly the behaviour of visitors most likely to be unfamiliar with your site and brand values, without having to rely on the clumsy new or returning demographic date.
2. **Traffic/conversion/sales by hour**
This is pure stats porn, but actually more useful than real-time data. By seeing this data broken down at an hourly

level, you can not only compare the current day to previous days, but also see the best performing times for email broadcasts and tweets.

3. **Visits, load time, conversion and sales by page and browser**
Page speed can often kill conversion rates, but it's difficult to prove the value of focusing on speed in monetary terms. Having this report to hand helps me drive Operation Greenbelt, our effort to get into the sub-1.5 second band in Google Webmaster Tools.

Useful things you can't do in custom reporting

If you have a search function on your website, then Conversion Rate and Products Bought by Site Search Term is an incredibly useful report that allows you to measure the effectiveness of your site's search engine at returning products and content related to the search term used. By including the products actually bought by visitors who searched for each term, you can use this information to better searchandise these results, escalating high propensity and high value products to the top of the results.

However, it's not possible to get this information out of new Google Analytics.

Try it, select the following in the report builder:

- **Metrics:** total unique searches; e-commerce or goal conversion rate
- **Dimensions:** search term; product

You'll see that the data returned is a little nonsensical, though a 2,000% conversion rate would be nice. However, you can get more accurate information using advanced segments. By creating individual segments to define users who have searched for a particular term, you can run the sales performance and product performance reports as normal. It's laborious, but it teaches a good lesson: data that seems inaccessible can normally be found another way!

REPORTING INFRASTRUCTURE

Now that you have a series of reports that you can refer to on a daily or weekly basis, it's time to put together a regular reporting infrastructure.

Even if you're not reporting to someone, having a set of key performance indicators that you can use to see how your performance is improving over time allows you to set yourself business goals on a monthly and annual basis.

For my own reporting, I take some high-level metrics (such as visitors, conversion rate and average order value), and segment them by traffic source and, separately, landing page. These statistics I record weekly and report:

- current week compared with previous week
- same week previous year (if available)
- 4 week average
- 13 week average
- 52 week average (if available)

This takes into account weekly, monthly, seasonal and annual trends, and gives you a much clearer view of your performance.

GETTING DATA IN OTHER WAYS

If you’re using Google Analytics frequently, with any large site you’ll come to a couple of conclusions:

1. Doing any kind of practical comparative analysis is unwieldy.
2. Boy, Google Analytics is slow!

As you work with bigger datasets and put together more complex queries, you’ll see the loading graphic more than you’ll see actual data. So when you reach that level, there are ways to completely bypass the Google Analytics interface altogether, and get data into your own spreadsheet application for manipulation.

Data Feed Query Explorer

If you just want to pull down some quick statistics but still use complex filters and exotic metric and dimension combinations, the Data Feed Query Explorer is the quickest way of doing so. Authenticate with your Google Analytics account, select a profile, and you can start selecting metrics and dimensions to be generated in a handy, selectable tabulated format.

Google Analytics API

If you're feeling clever, you can bypass having to copy and paste data by pulling in directly into Excel, Google Docs or your own application using the Google Analytics API. There are several scripts and plugins available to do this. I use Automate Analytics Google Docs code (there's also a paid version that simplifies setup and creates some handy reports for you).

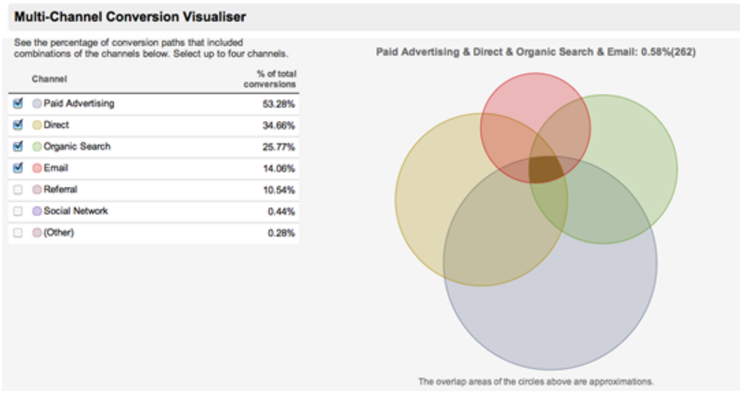
NEW SHINY THINGS

Well, now that that's over, I can show you some cool stuff. Well, at least it's cool to me. Google Analytics is being constantly improved and new functionality is introduced nearly every month. Here are a couple of my favourites.

Multichannel attribution

Not every visitor converts on your site on the first visit. They may not even do so on the second visit, or third. If they convert on the fourth visit, but each time they visit they do so via a different channel (for example, Search PPC, Search Organic, Direct, Email), which channel do you attribute the conversion to? The last channel, or the first? Dilemma!

Google now has a Multichannel Attribution report, available in the Conversion category, which shows how each channel assists in converting, the overlap between channels, and where in the process that channel was important.



Multi-Channel Conversion Visualiser

See the percentage of conversion paths that included combinations of the channels below. Select up to four channels.

	Channel	% of total conversions
✓	Paid Advertising	53.28%
✓	Direct	34.66%
✓	Organic Search	25.77%
✓	Email	14.06%
☐	Referral	10.54%
☐	Social Network	0.44%
☐	(Other)	0.28%

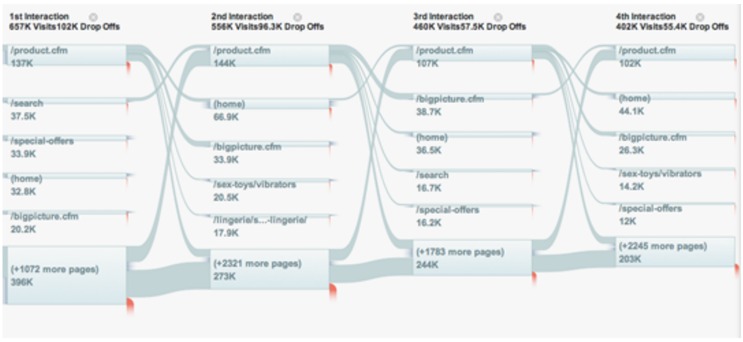
Paid Advertising & Direct & Organic Search & Email: 0.58%(262)

The overlap areas of the circles above are approximations.

For example, you may have analysed your blog traffic from Twitter and become disheartened that not many people were subscribing after visiting from Twitter links, but instead your high-value subscribers were coming from natural search. On the face of it, you'd spend less time tweeting, but a multichannel report may tell you that visitors first arrived via a Twitter link and didn't subscribe, but then came back later after searching for your blog name on Google, after which they did. Don't pack Twitter in yet!

Visitor and goal flow

Visitor and goal flow are amazing reports that help you visualize the flow of traffic through your site and, ultimately, into your checkout funnel or similar goal path. Flow reports are perfect for understanding drop-off points in your process, as well as what the big draws are on each page.



Previously, if you wanted to visualize this data you had to set up several abstracted microgoals and chain them together in custom reports. Frankly, it was a pain in the arse and burned through your precious and limited goal allocation.

Visitor flow bypasses all that and produces the report in an interactive flow diagram. While it doesn't show you the holy grail of conversion likelihood by each path, you can segment visitor flow so that you can see very specifically how different segments of your visitor base behave.

Go play with it now!

ABOUT THE AUTHOR

Matt Curry is Head of e-Commerce for Lovehoney. He's seen things you wouldn't believe actually exist let alone are bought online, is not easily embarrassed and has worked in e-commerce for nearly 10 years. Previous to perverting the nation, he sold frozen food to the elderly. He's a statistician by trade, a perfumer by fancy and a constant delight at parties. You can find him on Twitter @mattycurry

19. Going Both Ways

Jonathan Snook 24ways.org/201119

It’s that time of the year again: Santa is getting ready to travel the world. Up until now, girls and boys from all over have sent in letters asking for what they want. I hope that Santa and his elves have—unlike me—learned more than just English.

On the Internet, those girls and boys want to participate in sharing their stories and videos of opening presents and of being with friends and family. Ah, yes, the wonders of user generated content. But more than that, people also want to be able to use sites in the language they know.

While you and I might expect the text to read from left to right, not all languages do. Some go from right to left, such as Arabic and Hebrew. (Some also go from top to bottom, but for now, let’s just worry about those first two directions!)

If we were building a site for girls and boys to send their letters to Santa, we need to consider having the interface in the language and direction that they prefer. On the elves' side, they may be viewing the site in one direction but reading the user generated content in the other direction. We need to build a site that supports bidirectional (or bidi) text.

Let's take a look at some things to be aware of when it comes to building bidi interfaces.

SETTING THE DIRECTION OF THE INTERFACE

Right off the bat, we need to tell the browser what direction the text should be going in. To do this, we add the `dir` attribute to an HTML element and set it to either `LTR` (for left to right) or `RTL` (for right to left).

```
<body dir="rtl">
```

You can add the `dir` attribute to any element and it will set or change the direction for the content within that element.

```
<body dir="ltr">
   Here is English Content.
   <div dir="rtl">الموضوع</div>
</body>
```

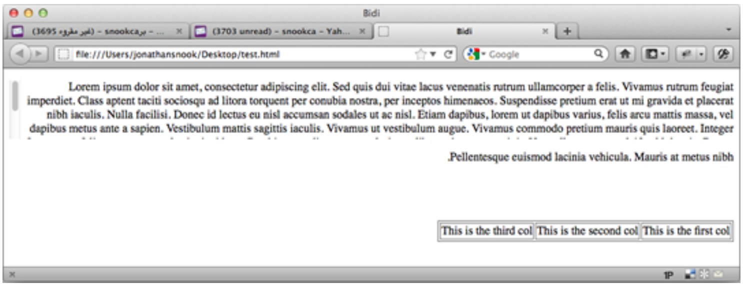
You can also set the direction via CSS.

```
.rtl {
    direction: rtl;
    }
```

It's generally recommended that you don't use CSS to set the direction of the text. Text direction is an important part of the content that should be retained even in environments where the CSS may not be available or fails to load.

HOW THINGS CHANGE WITH THE DIRECTION ATTRIBUTE

Just adding the `dir` attribute tells the browser to render the content within it differently.


Bidi
(3695 غير مقروء) - snookca بريد - ...
(3703 unread) - snookca - Yah...
Bidi
file:///Users/jonathansnook/Desktop/test.html
Google

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed quis dui vitae lacus venenatis rutrum ullamcorper a felis. Vivamus rutrum feugiat imperdiet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Suspendisse pretium erat ut mi gravida et placerat nibh iaculis. Nulla facilisi. Donec id lectus eu nisl accumsan sodales ut ac nisl. Etiam dapibus, lorem ut dapibus varius, felis arcu mattis massa, vel dapibus metus ante a sapien. Vestibulum mattis sagittis iaculis. Vivamus ut vestibulum augue. Vivamus commodo pretium mauris quis laoreet. Integer

.Pellentesque euismod lacinia vehicula. Mauris at metus nibh

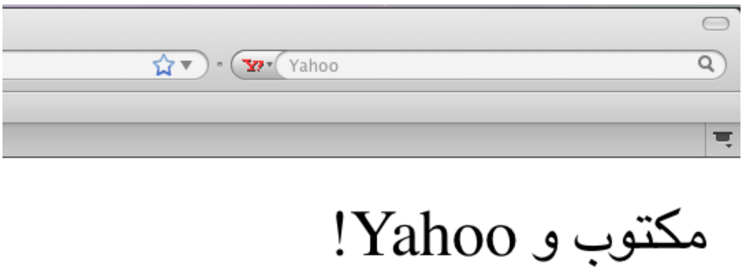
This is the third col	This is the second col	This is the first col

The text aligns to the right of the page and, interestingly, punctuation appears at the left of the sentence. (We'll get to that in a little bit.)

Scrollbars in most browsers will appear on the left instead of the right. Webkit is the notable exception here which always shows the scrollbar on the right, no matter what the text direction is. Avoid having a design that has an expectation that the scrollbar will be in a specific place (and a specific size).

CHANGING THE ORDER OF TEXT MID-WAY

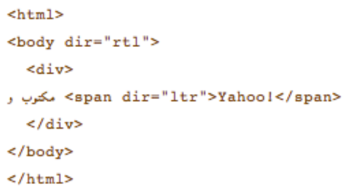
As we saw in that previous example, the punctuation appeared at the beginning of the sentence instead of the end, even though the text was English. At Yahoo!, we have an interesting dilemma where the company name has punctuation in it. Therefore, when the name appears in the middle of (for example) Arabic text, the exclamation mark appears at the beginning of the word instead of the end.



There are two ways in which this problem can be solved:

1. Use HTML around the left-to-right content, or

To solve the problem of the Yahoo! name in the midst of Arabic text, we can wrap a span around it and change the direction on that element.



```
<html>
<body dir="rtl">
  <div>
, مكتوب <span dir="ltr">Yahoo!</span>
  </div>
</body>
</html>
```

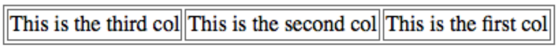
2. Use a text direction mark in the content.

Unicode has two marks, `U+200E` and `U+200F`, that tell the browser that the text is in a particular direction. Placing this right after the punctuation will correct the placement.

Using the HTML entity:
`Yahoo!`

TABLES

Thankfully, the cells of a data table also get reordered from right to left. Equally as nice, if you're using `display:table`, the content will still get reordered.



This is the third col	This is the second col	This is the first col

CSS

So far, we've seen that the `dir` attribute does a pretty decent job of getting content flowing in the direction that we need it. Unfortunately, there are huge swaths of design that is handled by CSS that the handy `dir` attribute has zero effect over.

Many properties, like float or absolute positioning with left and right values, are unaffected and must be handled manually. Elements that were floated left must now by floated right. Left margins and paddings must now move to the right and the right margins and paddings must now move to the left.

Since the browser won't handle this for us, we have a couple approaches that we can use:

CSS Only

We can take advantage of the attribute selector to target CSS to apply in one direction or another.

```
[dir=ltr] .module {
  float: left;
  margin: 0 0 0 20px;
}

[dir=rtl] .module {
  float: right;
  margin: 0 20px 0 0;
}
```

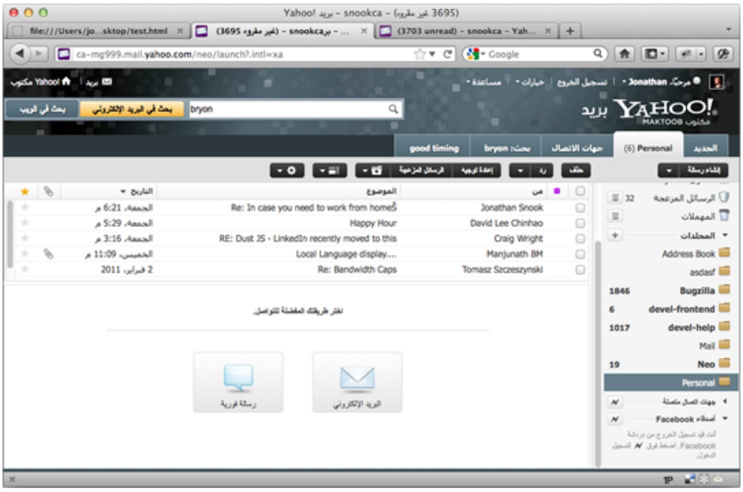
As you can see from this example, both of the properties have been modified for the flipped interface. If your interface is rather complicated, you will have to create a lot of duplicate rules to have the site looking good in both directions while serving up a single stylesheet.

CSSJanus

Google has a tool called CSSJanus. It’s a Python script that runs over the LTR versions of your CSS files and generates RTL versions. For the RTL version of the site, just serve up those CSS files instead of the LTR versions.

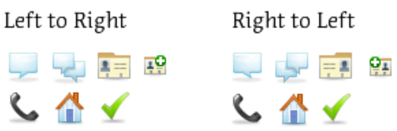
The script looks for keywords and value combinations and automatically swaps them so you don’t have to.

At Yahoo!, CSSJanus was a huge help in speeding up our development of a bidi interface. We’ve also made a number of improvements to the script to better handle border radius, background positioning, and gradients. We will be pushing those changes back into the CSSJanus project.



BACKGROUND IMAGES

Background images, especially for things like CSS sprites, also raise an interesting dilemma. Background images are positioned relative to the left of the element. In a flipped interface, however, we need to position it relative to the right. An icon that would be to the left of some text will now need to appear on the right.



If the x position of the background is percentage-based, then it's fairly easy to swap the values. 0 becomes 100%, 10% becomes 90% and so on. If the x position is pixel-based, then we're in a bit of a pickle. There's no way to say that the image should be a certain number of pixels from the right.

Therefore, you'll need to ensure that any background image that needs to be swapped should be percentage-based. (99.9% of the the time, the background position will need to be 0 so that it can be changed to 100% for RTL.)

If you're taking an existing implementation, background positioning will likely be the biggest hurdle you'll have to overcome in swapping your interface around. If you make sure your x position is always percentage-based from the beginning, you'll have a much smoother process ahead of you!

FLIPPING IMAGES

This is a more subtle point and one where you'll really want an expert with the region to weigh in on. In RTL interfaces, users may expect certain icons to also be flipped. Pencil icons that skew to the right in LTR interfaces might need to be swapped to skew to the left, instead. Chat bubbles that come from the left will need to come from the right.

The easiest way to handle this is to create new images. Name the LTR versions with `-ltr` in the name and name the RTL versions with `-rtl` in the name. CSSJanus will automatically rename all file references from `-ltr` to `-rtl`.

THE FUTURE

Thankfully, those within the W3C recognize that CSS should be more agnostic. As a result, they've begun introducing new properties that allow the browser to manage the swapping from left to right for us.

The CSS3 specification for backgrounds allows for the `background-position` to be relative to other corners other than the top left by specifying keywords before each position.

This will position the background 5px from the bottom right of the element.

```
background-position: right 5px bottom 5px;
```

Opera 11.60 is currently the only browser that supports this syntax.

For margin and padding, we have `margin-start` and `margin-end`. In LTR interfaces, `margin-start` would be the same as `margin-left` and in RTL interfaces, `margin-start` would be the same as `margin-right`.

Firefox and Webkit support these but with vendor prefixes right now:

```
-webkit-margin-start: 20px;
-moz-margin-start: 20px;
```

In the CSS3 Images working draft specification, there's an `image()` property that allows us to specify image fallbacks and whether those fallbacks are for LTR or RTL interfaces.

```
background: image('sprite.png' ltr, 'sprite-rtl.png'
rtl);
```

Unfortunately, no browser supports this yet but it's nice to be able to dream of how much easier this will be in the future!

HO HO HO

Hopefully, after all of this, you're full of cheer knowing that you're well on your way to creating interfaces that can go both ways!

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. Raising the Bar on Mobile

Scott Jehl 24ways.org/201120

One of the primary challenges of designing for mobile devices is that screen real estate is often in limited supply. Through the advocacy of Luke W and others, we've drawn comfort from the idea that this constraint ends up benefiting users and designers alike, from obvious advantages like portability and reach, to influencing our content strategy decisions through focus and restraint. But that doesn't mean we shouldn't take advantage of every last pixel of that screen we can snag!

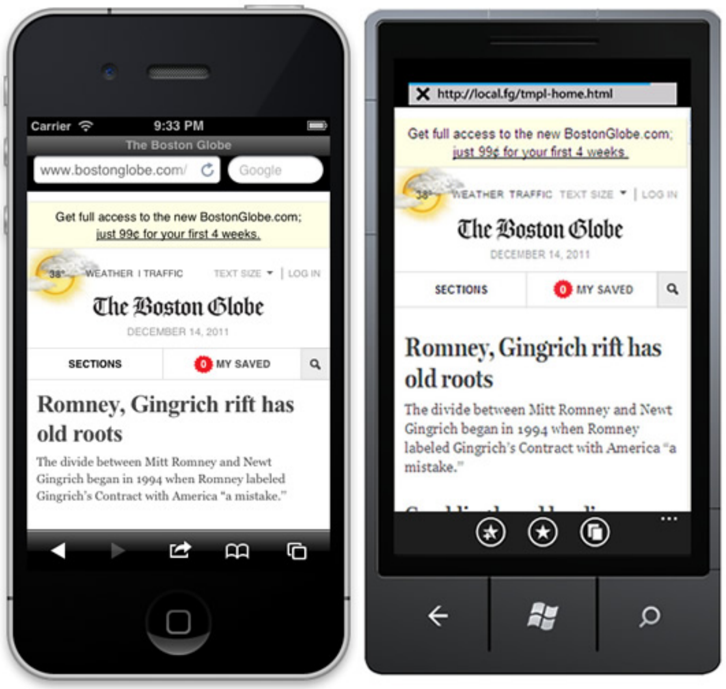
As anyone who has designed a website for use on a smartphone can attest, there's an awful lot of space on mobile screens dedicated to browser functions that would be better off toggled out of view. Unfortunately, the visibility of some of these elements is beyond our control, such as the buttons fixed to the bottom of the viewport in iOS's Safari and the WebOS browser. However, in many devices, the address bar at the top can be manually hidden, and its absence frees up enough pixel room for a large, impactful heading, a critical piece of navigation, or even just a little more white space to air things out.

So, as my humble contribution to this most festive of web publications, today I'll dig into the approach I used to hide the address bar in a browser-agnostic fashion for sites like BostonGlobe.com, and the jQuery Mobile framework.

SURVEYING THE LAND

First, let's assess the chromes of some popular, current mobile browsers. For example purposes, the following screen-captures feature the homepage of the Boston Globe site, without any address-bar-hiding logic in place.

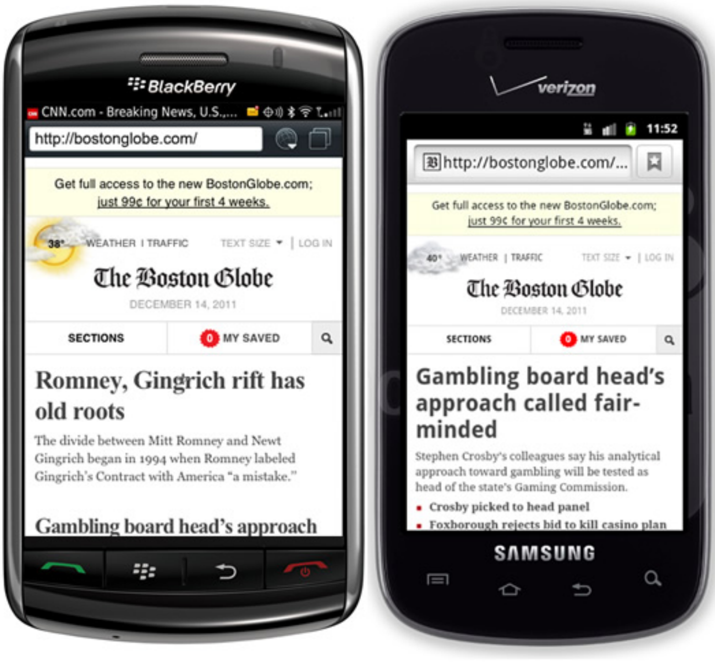
Note: these captures are just mockups – actual experience on these platforms may vary.



Romney, Gingrich rift has old roots

The divide between Mitt Romney and Newt Gingrich began in 1994 when Romney labeled Gingrich's Contract with America "a mistake."

On the left is iOS5's Safari (running on iPhone), and on the right is Windows Phone 7 (pre-Mango).


BlackBerry
http://bostonglobe.com/
Get full access to the new BostonGlobe.com; just 99¢ for your first 4 weeks.
The Boston Globe
DECEMBER 14, 2011
SECTIONS
MY SAVED
Romney, Gingrich rift has old roots
Gambling board head's approach called fair-minded
verizon
SAMSUNG

BlackBerry 7 (left), and Android 2.3 (right).

The Boston Globe
bostonglobe.com/
Get full access to the new BostonGlobe.com; just 99¢ for your first 4 weeks.
DECEMBER 14, 2011
SECTIONS
MY SAVED
Romney, Gingrich rift has old roots
Gambling board head's approach called fair-minded
SAMSUNG

WebOS (left), Opera Mini (middle), and Opera Mobile (right).

Some browsers, such the default browsers on WebOS and BlackBerry 5, hide the bar automatically without any developer intervention, but many of them don't. Of these, we can only manually hide the address bar on iOS Safari and Android (according to Opera Web Opener, Mike Taylor, some discussion is underway for support in Opera Mini and Mobile as well, which would be great!). This is unfortunate, but iOS and Android are incredibly popular, so let's direct our focus there.

GREAT API, OR GREATEST API?

As it turns out, iOS and Android not only allow you to hide the address bar, they use the same JavaScript method to do so, too (this shouldn't be surprising, given that they *are* both WebKit browsers, but **nothing** expected happens in mobile). However, the method they use is not exactly intuitive. You might set out looking for a JavaScript API dedicated to this purpose, like, say, `window.toolbar.hide()`, but alas, to hide the address bar you need to use the `window.scrollTo` method!

WINDOW.SCROLLTO(0, 0);

The `scrollTo` method is not new, it's just this particular use of it that is. For the uninitiated, `scrollTo` is designed to scroll a document to a particular set of coordinates, assuming the document is large enough to scroll to that spot. The method accepts two arguments: a left coordinate; and a top coordinate. It's both simple and supported well pretty much everywhere. In iOS and Android, these coordinates are calculated from the top of the browser's viewport, just below the address bar (interestingly, it seems that some platforms like BlackBerry 6 treat the top of the browser chrome as `0` instead, meaning the page content is closer to `20px` from the top).

Anyway, by passing the coordinates `0, 0` to the `scrollTo` method, the browser will jump to the top of the page and pull the address bar out of view! Of course, if a quick call to `scrollTo` was all we need to do to hide the address bar in iOS and Android, this article would be pretty short, and nothing new. Unfortunately, the first issue we need to deal with is that this method alone will not usually do the trick: it must be called after the page has finished loading.

The browser gives us a `load` event for just that purpose, so we'll wrap our `scrollTo` method in it and continue on our merry way! We'll use the standard, `addEventListener` method to bind the the `load` event, passing arguments for event name `load`, and a callback function to execute when the event is triggered.

```
window.addEventListener("load",function() {
    window.scrollTo(0, 0);
});
```

For the sake of preventing errors in those using browsers that don't support `addEventListener`, such as Internet Explorer 8 and under, let's make sure that method exists before we use it:

```
if( window.addEventListener ){
    window.addEventListener("load",function() {
        window.scrollTo(0, 0);
    });
}
```

Now we're getting somewhere, but we must also call the method *after* the `load` event's default behavior has been applied. For this, we can use the `setTimeout` method, delaying its execution to after the `load` event has run its course.

```
if( window.addEventListener ){
    window.addEventListener("load",function() {
        setTimeout(function(){
            window.scrollTo(0, 0);
        }, 0);
    });
}
```

Sweet sugar of Christmas! Hit this demo in iOS and watch that address bar drift up and away!

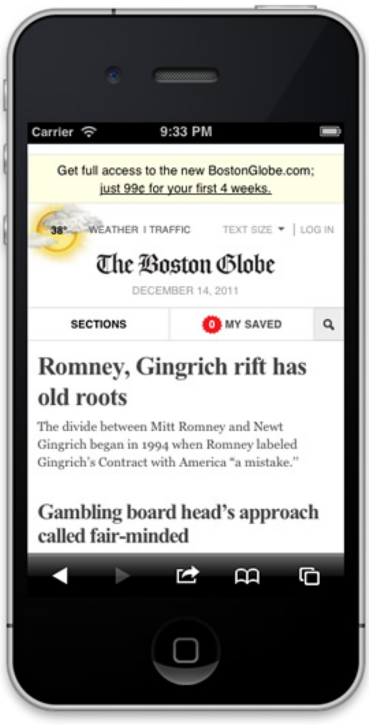
NOT SO FAST…

We've got a little problem: the approach above does work in iOS but, in some cases, it works a little *too well*. In the process of applying this behavior, we've broken one of the primary tenets of responsible web development: **don't break the browser's default behaviour**. This usability rule of thumb is often violated by developers with even the best of intentions, from breaking the browser's back button through unrecorded Ajax page refreshes, to fancy momentum touch scrolling scripts that can wreak havoc in all but the most sophisticated of devices. In this case, we've prevented the browser's native support of deep-linking to sections of a page (a hash identifier in the URL matching a page element's `id` attribute, for example, `http://example.com#contact`) from working properly, because our script always scrolls to the top.

To avoid this collision, we'll need to detect whether a deep link, or hash, is present in the URL before applying our logic. We can do this by ensuring that the `location.hash` property is falsey:

```
if( !window.location.hash && window.addEventListener ){
    window.addEventListener( "load",function() {
        setTimeout(function(){
            window.scrollTo(0, 0);
        }, 0);
    });
}
```

Still works great! And a quick test using a hash-based URL confirms that our script will not execute when a deep anchor is in play. Now iOS is looking sharp, and we've added our feature defensively to avoid conflicts.



Get full access to the new BostonGlobe.com; just 99¢ for your first 4 weeks.

Romney, Gingrich rift has old roots

The divide between Mitt Romney and Newt Gingrich began in 1994 when Romney labeled Gingrich's Contract with America "a mistake."

Gambling board head's approach called fair-minded

NOW, ON TO ANDROID…

Wait. You didn't expect that we could write code for one browser and be finished, right? Of course you didn't. I mentioned earlier that Android uses the same method for getting rid of the scrollbar, but I left out the fact that the arguments it prefers vary slightly, but significantly, from iOS. Bah!

Differering from the earlier logic from iOS, to remove the address bar on Android's default browser, you need to pass a Y coordinate of `1` instead of `0`. Aside from being just plain odd, this is particularly unfortunate because to any other browser on the planet, 1px is a very real, however small, distance from the top of the page!

```
window.scrollTo( 0, 1 );
```

Looks like we're going to need a fork…

R UA ANDROID?

At this point, some developers might decide to simply not support this feature in Android, and more determined devs might decide that a quick check of the User Agent string would be a reliable way to determine the browser and tweak the scroll value accordingly. Neither of those decisions would be tragic, but in the spirit of cross-browser and future-friendly development, I'll propose an alternative.

By this point, it should be clear that neither of the implementations above offer a particularly intuitive way to hide an address bar. As such, one might be skeptical that these approaches will stick around very long in their present state in either browser. Perhaps at some point, Android will decide to use `0` like iOS, making our lives a little easier, or maybe some new browser will decide to model their address bar hiding method after one of these implementations. In any case, detecting the User Agent only allows us to apply logic based on the known present, and in the world of mobile, let's face it, the present is already the past.

WRITING A CHECK

In this next step of today's technique, we'll apply some logic to quickly determine the behavior model of the browser we're using, then capitalize on that model – without caring which browser it happens to come from – by applying the appropriate scroll distance.

To do this, we'll rely on a fortunate side effect of Android's implementation, which is when you programatically scroll the page to `1` using `scrollTo`, Android will report that it's still at `0` because oddly enough, it is! Of course, any other browser in this situation will report a scroll distance of `1`. Thus, by scrolling the page to 1, then asking the browser its scroll distance, we can use this artifact of their wacky implementation to our advantage and scroll to the location that makes sense for the browser in play.

GETTING THE SCROLL DISTANCE

To pull off our test, we'll need to ask the browser for its current scroll distance. The methods for getting scroll distance are not entirely standardized across popular browsers, so we'll need to use some cross-browser logic. The following scroll distance function is similar to what you'd find in a library like jQuery. It checks the few common ways of getting scroll distance before eventually falling back to `0` for safety's sake (that said, I'm unaware of any browsers that won't return a numeric value from one of the first three properties).

```
// scrollTop getter
function getScrollTop(){
    return scrollTop  = window.pageYOffset ||
document.compatMode === "CSS1Compat" &&
document.documentElement.scrollTop ||
document.body.scrollTop || 0;
}
```

In order to execute that code above, the body object (referenced here as `document.body`) will need to be defined already, or we'll risk an error. To determine that it's defined, we can run a quick timer to execute code as soon as that object is defined and ready for use.

```
var bodycheck = setInterval(function(){
    if( document.body ){
        clearInterval( bodycheck );
        //more logic can go here!!
    }
}, 15 );
```

Above, we've defined a 15 millisecond interval called `bodycheck` that checks if `document.body` is defined and, if so, clears itself of running again. Within that `if` statement, we can extend our logic further to run other code, such as our check for the scroll distance, defined via the variable `scrollTop` below:

```
var scrollTop,
    bodycheck = setInterval(function(){
    if( document.body ){
        clearInterval( bodycheck );
        scrollTop = getScrollTop();
    }
}, 15 );
```

With this working, we can immediately scroll to `1`, then check the scroll distance when the body is defined. If the distance reports `1`, we're likely in a non-Android browser, so we'll scroll back to `0` and clean up our mess.

```
window.scrollTo( 0, 1 );

var scrollTop,
    bodycheck = setInterval(function(){
    if( document.body ){
        clearInterval( bodycheck );
        scrollTop = getScrollTop();
        window.scrollTo( 0, scrollTop === 1 ? 0 : 1 );
    }
}, 15 );
```

CASHING IN

All of the pieces are written now, so all we need to do is combine them with our previous logic for scrolling when the window is loaded, and we'll have a cross-browser solution of which John Resig would be proud. Here's our combined code snippet, with some formatting updates rolled in as well:

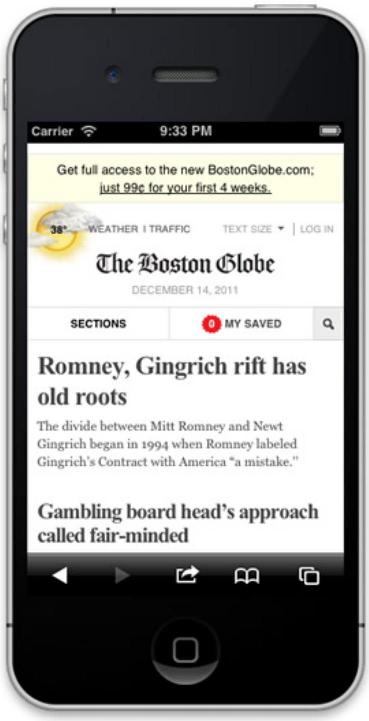
```
(function( win ){
  var doc = win.document;
```

// If there's a hash, or addEventListener is undefined, stop here if(!location.hash && win.addEventListener){ //scroll to 1 window.scrollTo(0, 1); var scrollTop = 1, getScrollTop = function(){ return win.pageYOffset || doc.compatMode = "CSS1Compat" && doc.documentElement.scrollTop || doc.body.scrollTop || 0; }, //reset to 0 on bodyready, if needed bodycheck = setInterval(function(){ if(doc.body){ clearInterval(bodycheck); scrollTop = getScrollTop(); win.scrollTo(0, scrollTop = 1 ? 0 : 1); } }, 15); win.addEventListener("load", function(){ setTimeout(function(){ //reset to hide addr bar at onload

```
win.scrollTo( 0, scrollTop === 1 ? 0 : 1 ); }, 0); } ); }
})( this );
```

View code example

And with that, we've got a bunch more room to play with on both iOS and Android.



Get full access to the new BostonGlobe.com; just 99¢ for your first 4 weeks.

The Boston Globe

DECEMBER 14, 2011

Romney, Gingrich rift has old roots

The divide between Mitt Romney and Newt Gingrich began in 1994 when Romney labeled Gingrich's Contract with America "a mistake."

Gambling board head's approach called fair-minded

BREAK OUT THE EGGNOG

…because we're not done yet! In the spirit of making our script act more defensively, there's still another use case to consider. It was essential that we used the window's `load` event to trigger our scripting, but on pages with a lot of content, its use can come at a cost. Often, a user will begin interacting with a page, scrolling down as they read, before the load event has fired. In those situations, our script will jump the user back to the top of the page, resulting in a jarring experience.

To prevent this problem from occurring, we'll need to ensure that the page has not been scrolled beyond a certain amount. We can add a simple check using our `getScrollTop` function again, this time ensuring that its value is not greater than 20 pixels or so, accounting for a small tolerance.

```
if( getScrollTop() < 20 ){
    //reset to hide addr bar at onload
    window.scrollTo( 0, scrollTop === 1 ? 0 : 1 );
}
```

And with that, we're pretty well protected! Here's a final demo.

The completed script can be found on Github (full source: https://gist.github.com/1183357). It's MIT licensed. Feel free to use it anywhere or any way you'd like!

YOUR THOUGHTS?

I hope this article provides you with a browser-agnostic approach to hiding the address bar that you can use in your own projects today. Perhaps alternatively, the complications involved in this approach convinced you that doing this well is more trouble than it's worth and, depending on the use case, that could be a fair decision. But at the very least, I hope this demonstrates that there's a lot of work involved in pulling off this small task in only two major platforms, and that there's a real need for standardization in this area.

Feel free to leave a comment or criticism and I'll do my best to answer in a timely fashion.

Thanks, everyone!

SOME PARTING NOTES

I scream, you scream…

At the time of writing, I was not able to test this method on the latest Android 4.0 (Ice Cream Sandwich) build. According to Sencha Touch's browser scorecard, the browser in 4.0 may have a different way of managing the address bar, so I'll post in the comments once I get a chance to dig into it further.

Short pages get no love

Today's technique only works when the page is as tall, or taller than, the device's available screen height, so that the address bar may be scrolled out of view. On a short page, you might work around this issue by applying a minimum height to the body element (`body { min-height: 460px; }`), but given the variety of screen sizes out there, not to mention changes in orientation, it's tough to find a value that makes much sense (unless you manipulate it with JavaScript).

ABOUT THE AUTHOR

Scott Jehl is a web designer / developer who works with the bright folks at Filament Group. At Filament, Scott helps craft websites and applications for a range of clients, including the recent Responsive design of the Boston Globe, and regularly contributes ideas and code to the open source community, such as the recent Respond.js project.

Scott enjoys writing and speaking about web design, and in 2010 co-authored the book Designing with Progressive Enhancement. He has written for A List Apart and is a jQuery core team member, most recently leading the development of the jQuery Mobile project.

Currently, Scott is tromping around Southeast Asia with his wife, pushing his commits from afar.

21. Taming Complexity

Simon Collison 24ways.org/201121

I'm going to step into my UX trousers for this one. I wouldn't usually wear them in public, but it's Christmas, so there's nothing wrong with looking silly.

Anyway, to business. Wherever I roam, I hear the familiar call for simplicity and the denouncement of complexity. I read often that the simpler something is, the more usable it will be. We understand that simple is hard to achieve, but we push for it nonetheless, convinced it will make what we build easier to use. Simple is better, right?

Well, I'll try to explore that. Much of what follows will not be revelatory to some but, like all good lessons, I think this serves as a welcome reminder that as we live in a complex world it's OK to sometimes reflect that complexity in the products we build.

MYTHS AND LEGENDS

Less is more, we've been told, ever since master of poetic verse Robert Browning used the phrase in 1855. Well, I've conducted some research, and it appears he knew nothing of web design. Neither did modernist architect Ludwig Mies van der Rohe, a later pedlar of this worthy yet contradictory notion. Broad is narrow. Tall is short. Eggs are chips. See: anyone can come up with this stuff.

To paraphrase Einstein, simple doesn't have to be simpler. In other words, simple doesn't dictate that we remove the complexity. Complex doesn't have to be confusing; it can be beautiful and elegant. On the web, complex can be necessary and powerful. A website that simplifies the lives of its users by offering them everything they need in one site or screen is powerful. For some, the greater the density of information, the more useful the site.

In our decision-making process, principles such as Occam's razor's_razor (in a nutshell: simple is better than complex) are useful, but simple is for the user to determine through their initial impression and subsequent engagement. What appears simple to me or you might appear very complex to someone else, based on their own mental model or needs. We can aim to deliver simple, but they'll be the judge.

As a designer, developer, content alchemist, user experience discombobulator, or whatever you call yourself, you're often wrestling with a wealth of material, a huge number of features, and numerous objectives. In many cases, much of that stuff is extraneous, and goes in the dustbin. However, it can be just as likely that there's a truckload of suggested features and content because it all needs to be there. Don't be afraid of that weight.

In the right hands, less can indeed mean more, but it's just as likely that less can very often lead to, well… less.

COMPLEXITY IS POWERFUL

Simple is the ability to offer a powerful experience without overwhelming the audience or inducing information anxiety. Giving them everything they need, without having them ferret off all over a site to get things done, is important.

It's useful to ask throughout a site's lifespan, "does the user have everything they need?" It's so easy to let our designer egos get in the way and chop stuff out, reduce down to only the things we want to see. That benefits us in the short term, but compromises the audience long-term.

The trick is not to be afraid of complexity in itself, but to avoid creating the *perception* of complexity. Give a user a flight simulator and they'll crash the plane or jump out. Give them everything they need and more, but make it *feel* simple, and you're building a relationship, empowering people.

This can be achieved carefully with what some call gradual engagement, and often the sensible thing might be to unleash complexity in carefully orchestrated phases, initially setting manageable levels of engagement and interaction, gradually increasing the inherent power of the product and fostering an empowered community.

THE DESIGN AESTHETIC

Here's a familiar scenario: the client or project lead gets overexcited and skips most of the important decision-making, instead barrelling straight into a bout of creative direction Tourette's. Visually, the design needs to be minimal, white, crisp, full of white space, have big buttons, and quite likely be "clean". Of course, we all like our websites to be clean as that's more hygienic.

But what do these words even mean, really? Early in a project they're abstract distractions, unnecessary constraints. This premature narrowing forces us to think much more about throwing stuff out rather than acknowledging that what we're building is complex, and many of the components perhaps necessary.

Simple is not a formula. It cannot be achieved just by using a white background, by throwing things away, or by breathing a bellowsful of air in between every element and having it all float around in space. Simple is not a design treatment. Simple is hard. Simple requires deep investigation, a thorough understanding of every aspect of a project, in line with the needs and expectations of the audience.

Recognizing this helps us empathize a little more with those most vocal of UX practitioners. They usually appreciate that our successes depend on a thorough understanding of the user's mental models and expected outcomes. I personally still consider UX people to be web designers like the rest of us (mainly to wind them up), but they're web designers that design every decision, and by putting the user experience at the heart of their process, they have a greater chance of finding simplicity in complexity. The visual design aesthetic — the façade — is only a part of that.

DIVIDE AND CONQUER

I'm currently working on an app that's complex in architecture, and complex in ambition. We'll be releasing in carefully orchestrated private phases, gradually introducing more complexity in line with the unavoidably complex nature of the objective, but my job is to design the whole, the complete system as it will be when it's out of beta and beyond.

I’ve noticed that I’m not throwing much out; most of it needs to be there. Therefore, my responsibility is to consider interesting and appropriate methods of navigation and bring everything together logically.

I’m using things like smart defaults, graphical timelines and colour keys to make sense of the complexity, techniques that are sympathetic to the content. They act as familiar points of navigation and reference, yet are malleable enough to change subtly to remain relevant to the information they connect. It’s really OK to have a lot of stuff, so long as we make each component work smartly.

It’s a divide and conquer approach. By finding simplicity and logic in each content bucket, I’ve made more sense of the whole, allowing me to create key layouts where most of the simplified buckets are collated and sometimes combined, providing everything the user needs and expects in the appropriate places.

I’m also making sure I don’t reduce the app’s power. I need to reflect the scale of opportunity, and provide access to or knowledge of the more advanced tools and features for everyone: a window into what they can do and how they can help. I know it’s the minority who will be actively building the content, but the power is in providing those opportunities for all.

Much of this will be familiar to the responsible practitioners who build websites for government, local authorities, utility companies, newspapers, magazines, banking, and we-sell-everything-ever-made online shops. Across the web, there are sites and tools that thrive on complexity.

Alas, the majority of such sites have done little to make navigation intuitive, or empower audiences. Where we can make a difference is by striving to make our UIs *feel* simple, look wonderful, not intimidating — even if they're mind-meltingly complex behind that façade.

EMBRACE, EMPATHIZE AND TAME

So, there are loads of ways to exploit complexity, and make it seem simple. I've hinted at some methods above, and we've already looked at gradual engagement as a way to make sense of complexity, so that's a big thumbs-up for a release cycle that increases audience power.

Prior to each and every release, it's also useful to rest on the finished thing for a while and use it yourself, even if you're itching to release. 'Ready' often isn't, and 'finished' never is, and the more time you spend browsing around the sites you build, the more you learn what to question, where to add, or subtract. It's definitely worth building in some contingency time for sitting on your work, so to speak.

One thing I always do is squint at my layouts. By squinting, I get a sort of abstract idea of the overall composition, and general feel for the thing. It makes my face look stupid, but helps me see how various buckets fit together, and how simple or complex the site feels overall.

I mentioned the need to put our design egos to one side and not throw out anything useful, and I think that's vital. I'm a big believer in economy, reduction, and removing the extraneous, but I'm usually referring to decoration, bells and whistles, and fluff. I wouldn't ever advocate the complete removal of powerful content from a project roadmap.

Above all, don't fear complexity. Embrace and tame it. Work hard to empathize with audience needs, and you can create elegant, playful, risky, surprising, emotive, delightful, and ultimately simple things.

ABOUT THE AUTHOR

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

Photo: Lachlan Hardy

22. From Side Project to Not So Side Project

Elliot Jay Stocks 24ways.org/201122

In the last article I wrote for 24 ways, back in 2009, I enthused about the benefits of having a pet project, suggesting that we should all have at least one so that we could collaborate with our friends, escape our day jobs, fulfil our own needs, help others out, raise our profiles, make money, and — most importantly — have fun. I don't think I need to offer any further persuasions: it seems that designers and developers are launching their own pet projects left, right and centre. This makes me very happy.

However, there still seems to be something of a disconnect between having a side project and turning it into something that is moderately successful; in particular, the challenge of making enough money to sustain the project and perhaps even elevating it from the sidelines so that it becomes something not so on the side at all.

Before we even begin this, let's spend a moment talking about money, also known as…

EVIL, NASTY, FILTHY MONEY

Over the last couple of years, I've started referring to myself as an accidental businessman. I say accidental because my view of the typical businessman is someone who is driven by money, and I usually can't stand such people. Those who are motivated by profit, obsessed with growth, and take an active interest in the world's financial systems don't tend to be folks with whom I share a beer, unless it's to pour it over them. Especially if they're wearing pinstriped suits.

That said, we all want to make money, don't we? And most of us want to make a relatively decent amount, too. I don't think there's any harm in admitting that, is there? Hello, I'm Elliot and I'm a capitalist.

The key is making money from doing what we love. For most people I know in our community, we've already achieved that — I'm hard-pressed to think of anyone who isn't extremely passionate about working in our industry and I think it's one of the most positive, unifying benefits we enjoy as a group of like-minded people — but side projects usually arise from another kind of passion: a passion for something other than what we do as our day jobs. Perhaps it's because your clients are driving you mental and you need a break; perhaps it's because you want to create something that is truly your own; perhaps it's because you're sick of seeing your online work disappear so fast and you want to try your hand at print in order to make a more permanent mark.

The three factors I listed there led me to create 8 Faces, a printed magazine about typography that started as a side project and is now a very significant part of my yearly output and income.

Like many things that prove fruitful, 8 Faces' success was something of an accident, too. For a start, the magazine was never meant to be profitable; its only purpose at all was to scratch my own itch. Then, after the first issue took off and I realized how much time I needed to spend in order to make the next one decent, it became clear that I would have to cover more than just the production costs: I'd have to take time out from client work as well. Doing this meant I'd have to earn some money. Probably not enough to equate to the exact amount of time lost when I could be doing client work (not that you could ever describe time as being lost when you work on something you love), but enough to survive; for me to feel that I was getting paid while doing all of the work that 8 Faces entailed. The answer was to raise money through partnerships with some cool companies who were happy to be associated with my little project.

A SUSTAINABLE BUSINESS MODEL

Business model! I can’t believe I just wrote those words! But a business model is really just a loose plan for how not to screw up. And all that stuff I wrote in the paragraph above about partnering with companies so I could get some money in while I put the magazine together? Well, that’s my business model.

If you’re making any product that has some sort of production cost, whether that’s physical print run expenses or up-front dev work to get an app built, covering those costs before you even release your product means that you’ll be in profit from the first copy you sell. This is no small point: production expenses are pretty much the only cost you’ll ever need to recoup, so having them covered before you launch anything is pretty much the best possible position in which you could place yourself. Happy days, as Jamie Oliver would say.

Obtaining these initial funds through partnerships has another benefit. Sure, it’s a form of advertising but, done right, your partners can potentially provide you with great content, too. In the case of 8 Faces, the ads look as nice as the rest of the magazine, and a couple of our partners also provide proper articles: genuinely meaningful, relevant, reader-pleasing articles at that. You'd be amazed at how many companies are willing to become partners and, as the old adage goes, **if you don't ask, you don't get.**

WITH PROFIT COMES RESPONSIBILITY

Don't forget about the responsibility you have to your audience if you engage in a relationship with a partner or any type of advertiser: although I may have freely admitted my capitalist leanings, I'm still essentially a hairy hippy, and I feel that any partnership should be good for me as a publisher, good for the partner and — most importantly — good for the reader. Really, the key word here is **relevance**, and that's where 99.9% of advertising fails abysmally.

(99.9% is not a scientific figure, but you know what I'm on about.)

The main grey area when a side project becomes profitable is how you share that profit, partly because — in my opinion, at least — the transition from non-profitable side project to relatively successful source of income can be a little blurred. Asking for help for nothing when there's no money to be had is pretty normal, but sometimes it's easy to get used to that free help even once you start making money. I believe the best approach is to ask for help with the promise that it will always be rewarded as soon as there's money available. (Oh, god: this sounds like one of those nightmarish client proposals. It's not, honest.) If you're making something cool, people won't mind helping out while you find your feet.

Events often think that they're exempt from sharing profit. Perhaps that's because many event organizers think they're doing the speakers a favour rather than the other way around (that's a whole separate article), but it's shocking to see how many people seem to think they can profit from content-makers — speakers, for example — and yet not pay for that content. It was for this reason that Keir and I paid all of our speakers for our Insites: The Tour side project, which we ran back in July. We probably could've got away without paying them, especially as the gig was so informal, but it was the right thing to do.

IN CONCLUSION: MONEY AS A BY-PRODUCT

Let's conclude by returning to the slightly problematic nature of money, because it's the pivot on which your side project's success can swing, regardless of whether you measure success by monetary gain. I would argue that success has nothing to do with profit — it's about you being able to spend **the time you want** on the project. Unfortunately, that is almost always linked to money: money to pay yourself while you work on your dream idea; money to pay for more servers when your web app hits the big time; money to pay for efforts to get the word out there. The key, then, is to judge success on your own terms, and seek to generate as much money as **you** see fit, whether it's purely to cover your running costs, or enough to buy a small country. There's nothing wrong with profit, as long as you're ethical about it. (Pro tip: if you've earned enough to buy a small country, you've probably been unethical along the way.)

The point at which individuals and companies fail — in the moral sense, for sure, but often in the competitive sense, too — is when money is the primary motivation. It should **never** be the primary motivation. If you're not passionate enough about something to do it as an unprofitable side project, you shouldn't be doing it all.

Earning money should be a by-product of doing what you love. And who **doesn't** want to spend their life doing what they love?

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

23. There's No Formula for Great Designs

Andrew Clarke 24ways.org/201123

Before he combined them with fluid images and CSS3 media queries to coin responsive design, Ethan Marcotte described fluid grids — one of the most enjoyable parts of responsive design. Enjoyable that is, if you like working with math(s). But fluid grids aren't perfect and, unless we're careful when applying them, they can sometimes result in a design that feels disconnected.

RECAPPING FLUID GRIDS

If you haven't read Ethan's Fluid Grids, now would be a good time to do that. It centres around a simple formula for converting pixel widths to percentages:

```
(target ÷ context) × 100 = result
```

How does that work in practice? Well, take that Fireworks or Photoshop comp you're working on (I call them static design visuals, or just visuals.) Of course, everything on that visual — column divisions, inline images, navigation elements, everything — is measured in pixels. Now:

1. Pick something in the visual and measure its width. That's our target.
2. Take that target measurement and divide it by the width of its parent (context).
3. Multiply what you've got by 100 (shift two decimal places).
4. What you're left with is a percentage width to drop into your style sheets.

For example, divide this 300px wide sidebar division by its 948px parent and then multiply by 100: your original 300px is neatly converted to 31.646%.

```
.content-sub {
width : 31.646%; /* 300px ÷ 948px = .31646 */ }
```

That formula makes it surprisingly simple for even die-hard fixed width aficionados to convert their visuals to percentage-based, fluid layouts.

It's a handy formula for those who still design using static visuals, and downright essential for those situations where one person in an organization designs in Fireworks or Photoshop and another develops with CSS. Why?

Well, although I think that designing in a browser makes the best sense — particularly when designing for multiple devices — I'll wager most designers still make visuals in Fireworks or Photoshop and use them for demonstrations and get feedback and sign-off. That's OK. If you haven't made the transition to content-out designing in a browser yet, the fluid grids formula helps you carry on pushing pixels a while longer.

You can carry on moving pixel width measurements from your visuals to your style sheets, too, in the same way you always have. You can be precise to the pixel and even apply a grid image as a CSS background to help you keep everything lined up perfectly.

Once you're done, and the fixed width layout in the browser matches your visual, loop back through your style sheets and convert those pixels to percentages using the fluid grids formula. With very little extra work, you'll have a fluid implementation of your fixed width layout.

The fluid grids formula is simple and incredibly effective, but not long after I started working responsively I realized that the formula shouldn't (always) be a one-fix, set-and-forget calculation. I noticed that unless we compensate for problems it sometimes creates, the result can be a disconnected design.

STAYING CONNECTED

Good design relies on connectedness, a feeling of natural balance between elements and the grid they're placed on. Give an element greater prominence or position in a visual hierarchy and you can fundamentally alter the balance and sometimes the meaning of a design.

Different from a browser's page zooming feature — where images, text and overall layout change size by the same ratio — fluid grids flex a layout in response to a window or device width. Columns expand and contract, and within them fluid media (images and videos) can also change size. This can be one of the most impressive demonstrations of responsive design.

But not every element within a fluid grid can change size along with the window or device width. For example, type size and leading won't change along with a column's width.

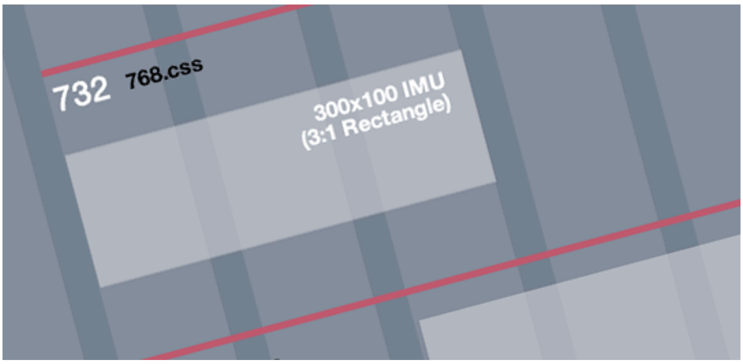
When columns and elements within them change width, all too easily a visual hierarchy can be broken and along with it the relationship between element sizes and the outer window or viewport. This can happen quickly if you make just one set of fluid grid calculations and use those percentages across every screen width, from smartphones through tablets and up to large desktops.

The answer? Make several sets of fluid grids calculations, each one at a significant window or device width breakpoint. Then apply those new percentages, when needed, to help keep elements in proportion and maintain balance and connectedness. Here's how I work.

AVOIDING DISCONNECTION

I've never been entirely happy with grid frameworks such as the 960 Grid System, so I start almost every project by creating a custom grid to inform my layout decisions. Here's a plain version of a grid from a recent project that I'll use as an illustration.

This project's grid comprises 84px columns and 24px gutters. This creates an odd number of columns at common tablet and desktop widths, and allows for 300px fixed width assets — useful when I need to fit advertising into a desktop layout's sidebar.

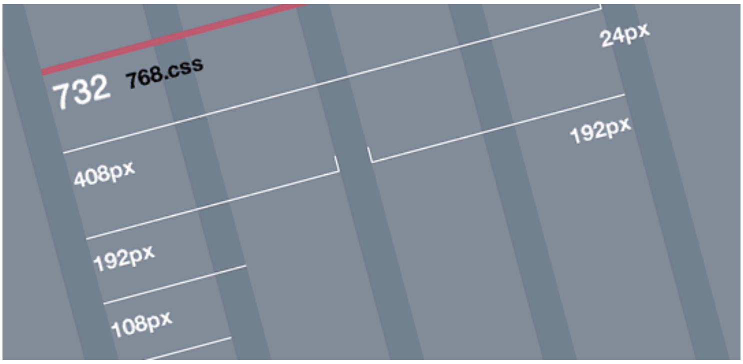


Showing common advertising sizes (Larger image)

For this project I chose three 320 and Up breakpoints above 320px and, after placing as many columns as would fit those breakpoint widths, I derived three content widths:

Breakpoint	Columns	Content width
768px	7	732px
992px	9	948px
1,382px	13	1,380px

Here's my grid again, this time with pixel measurements and breakpoints overlaid.



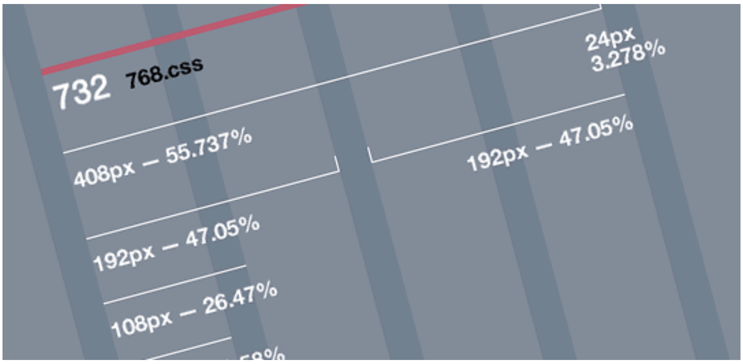
Showing pixel measurements and breakpoints (Larger image)

Now cast your mind back to the fluid grids calculation I made earlier. I divided a 300px element by 948px and arrived at 31.646%. For some elements it's possible to use that percentage across all screen widths, but others will feel too small in relation to a narrower 768px and too large inside 1,380px.

To help maintain connectedness, I make a set of fluid grids calculations based on each of the content widths I established earlier. Now I can shift an element's percentage width up or down when I switch to a new breakpoint and content width. For example:

- 300px is 40.984% of 732px
- 300px is 31.646% of 948px
- 300px is 21.739% of 1,380px

I'll add all those fluid grid percentages to my grid image and save it for quick reference.



Showing percentages at all breakpoints (Larger image)

Then I can apply those different percentage widths to elements at each breakpoint using CSS3 media queries. For example, that sidebar division again:

```
/*  732px, 7-column width */

@media only screen and (min-width: 768px) {

    .content-sub {
        width : 40.983%; /* 300px ÷ 732px = .40983 */ }

}

/*  948px, 9-column width */
@media only screen and (min-width: 992px) {

    .content-sub {
        width : 31.645%; /* 300px ÷ 948px = .31645 */ }

}

/*  1380px, 13-column width */
@media only screen and (min-width: 1382px) {

    .content-sub {
        width : 21.739%; /* 300px ÷ 1380px = .21739 */ }

}
```

The number of changes you make to a layout at different breakpoints will, of course, depend on the specifics of the design you're working on. Yes, this is additional work, but the result will be a layout that feels better balanced and within which elements remain in harmony with each other while they respond to new screen or device widths.

PUTTING THE DESIGN IN RESPONSIVE WEB DESIGN

Until now, many of the conversations around responsive web design have been about aspects of technical implementation, rather than design. I believe we're only beginning to understand what's involved in designing responsively. In future, we'll likely be making design decisions not just about proportions but also about responsive typography. We'll also need to learn how to adapt our designs to device characteristics such as touch targets and more.

Sometimes we'll make decisions to improve function, other times because they make a design 'feel' right. You'll know when you've made a right decision. You'll feel it.

After all, there really is no formula for making great designs.

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.

24. Crafting the Front-end

Ben Bodien 24ways.org/201124

Much has been spoken and written recently about the virtues of craftsmanship in the context of web design and development. It seems that we as fabricators of the web are finally tiring of seeking out parallels between ourselves and architects, and are turning instead to the fabled specialist artisans.

Identifying oneself as a craftsman or craftswoman (let's just say craftsperson from here onward) will likely be a trend of early 2012. In this pre-emptive strike, I'd like to expound on this movement as I feel it pertains to front-end development, and encourage care and understanding of the true qualities of craftsmanship (craftspersonship).

THE CORE VALUES

I'll begin by defining craftspersonship. What distinguishes a craftsperson from a technician? Dictionaries tend to define a craftsperson as one who possesses great skill in a chosen field. The badge of a craftsperson for me, though, is a very special label that should be revered and used sparingly, only where it is truly deserved. A genuine craftsperson encompasses a few other key traits, far beyond raw skill, each of which must be learned and mastered.

A craftsperson has:

- An appreciation of **good** work, in both the work of others and their own. And not just good as in 'hey, that's pretty neat', I mean a goodness like a shining purity – the kind of good that feels right when you see it.
- A belief in quality at every level: every facet of the craftsperson's product is as crucial as any other, without exception, even those normally hidden from view.
- Vision: an ability to visualize their path ahead, pre-empting the obstacles that may be encountered to plan a route around them.
- A preference for simplicity: an almost Bauhausesque devotion to undecorated functionality, with no unjustifiable parts included.
- Sincerity: producing work that speaks directly to its purpose with flawless clarity.

Only when you become a custodian of such values in your work can you consider calling yourself a craftsperson. Now let's take a look at some steps we front-end developers can take on our journey of enlightenment toward craftspersonhood.

Speaking of the craftsman's journey, be sure to watch out for the video of The Standardistas' stellar talk at the Build 2011 conference titled *The Journey*, which should be online sometime soon.

BUILDING YOUR OWN TOOLBOX

My grandfather was a carpenter and trained as a young apprentice under a master. After observing and practising the many foundation theories, principles and techniques of carpentry, he was tasked with creating his own set of woodworking tools, which he would use and maintain throughout his career. By going through the process of having to create his own tools, he would be connected at the most direct level with every piece of wood he touched, his tools being his own creations and extensions of his own skilled hands. The depth of his knowledge of these tools must have surpassed the intricate as he fathered, used, cleaned and repaired them, day in and day out over many years.

And so it should be, ideally, with all crafts. We must understand our tools right down to the most fundamental level. I firmly believe that a level of true craftsmanship cannot be reached while there exists a layer that remains not wholly understood between a creator and his canvas. Of course, our tools as front-end developers are somewhat more complex than those of other crafts – it may seem reasonable to require that a carpenter create his or her own set of chisels, but somewhat less so to ask a front-end developer to code their own CSS preprocessor, or design their own computer.

However, it is still vitally important that you understand how your tools work. This is particularly critical when it comes to things like preprocessors, libraries and frameworks which aim to save you time by automating common processes and functions. For the most part, anything that saves you time is a Good Thing™ but it cannot be stressed enough that using tools like these in earnest should be avoided until you understand exactly what they are doing for you (and, to an extent, how they are doing it).

In particular, you must understand any drawbacks to using your tools, and any shortcuts they may be taking on your behalf. I'm not suggesting that you steer clear of paid work until you've studied each of jQuery's 9,266 lines of JavaScript source code but, all levity aside, it will further you on your journey to look at interesting or relevant bits of jQuery, and any other libraries you might want to use. Such libraries often directly link to corresponding sections of their source code on sites like GitHub from their official documentation. Better yet, they're almost always written in high level languages (easy to read), so there's no excuse not to don your pith helmet and go on something of an exploration. Any kind of tangential learning like this will drive you further toward becoming a true craftsperson, so keep an open mind and always be ready to step out of your comfort zone.

DOWNTIME AND TOOL HONING

With any craft, it is essential to keep your tools in good condition, and a good idea to stay up-to-date with the latest equipment. This is especially true on the web, which, as we like to tell anyone who is still awake more than a minute after asking what it is that we do, advances at a phenomenal pace. A tool or technique that could be considered best practice this week might be the subject of haughty derision in a comment thread within six months.

I have little doubt that you already spend a chunk of time each day keeping up with the latest material from our industry's finest Interblogs and Twittertubes, but do you honestly put aside time to collect bookmarks and code snippets from things you read into a slowly evolving toolbox? At @media in 2009, Simon Collison delivered a candid talk on his 'Ultimate Package'. Those of us who didn't flee the room anticipating a newfound and unwelcome intimacy with the contents of his trousers were shown how he maintained his own toolkit – a collection of files and folders all set up and ready to go for a new project. By maintaining a toolkit in this way, he has consistency across projects and a dependable base upon which to learn and improve.

The assembly and maintenance of such a personalized and familiar toolkit is probably as close as we will get to emulating the tool making stage of more traditional craft trades. Keep a master copy of your toolkit somewhere safe, making copies of it for new projects. When you learn of a way in which part of it can be improved, make changes to the master copy.

SIMPLICITY THROUGH MODULARITY

I believe that the user interfaces of all web applications should be thought of as being made up primarily of modular components. Modules in this context are patterns in design that appear repeatedly throughout the app. These can be small collections of elements, like a user profile summary box (profile picture, username, meta data), as well as atomic elements such as headings and list items.

Well-crafted front-end architectures have the ability to support this kind of repeating pattern as modules, with as close to no repetition of CSS (or JavaScript) as possible, and as close to no variations in HTML between instances as possible.

One of the most fundamental and well known tenets of software engineering is the DRY rule – don't repeat yourself. It requires that "every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

As craftspeople, we must hold this rule dear and apply it to the modules we have identified in our site designs. The moment you commit a second style definition for a module, the quality of your output (the front-end code) takes a huge hit. There should only ever be one base style definition for each distinct module or component. Keep these in a separate, sacred place in your CSS. I use a **_modules.scss** Sass include file, imported near the top of my main CSS files.

Be sure, of course, to avoid making changes to this file lightly, as the smallest adjustment can affect multiple pages (hint: keep a structure list of which modules are used on which pages). Avoid the inevitable temptation to duplicate code late in the project. Sticking to this rule becomes more important the more complex the codebase becomes.

If you can stick to this rule, using sensible class names and consistent HTML, you can reach a joyous, self-fulfilling plateau stage in each project where you are assembling each interface from your own set of carefully crafted building blocks.

OLD SCHOOL MARKUP

Let’s take a step back. Before we fret about creating a divinely pure modular CSS framework, we need to **know** the site’s design and what it is made of. The best way to gain this knowledge is to go old school. Print out every comp, mockup, wireframe, sketch or whatever you have. If there are sections of pages that are hidden until some user action takes place, or if the page has multiple states, be sure that you have everything that could become visible to the user on paper.

Once you have your wedge of paper designs, lay out all the pages on the floor, or stick them to the wall if you can, arranging them logically according to the site hierarchy, by user journey, or whatever guidelines make most sense to you. Once you have the site laid out before you, study it for a while, familiarizing yourself with every part of every interface. This will eliminate nasty surprises late in the project when you realize you’ve duplicated something, or left an interface on the drawing board altogether.

Now that you know the site like it’s your best friend, get out your pens or pencils of choice and attack it. Mark it up like there’s no tomorrow. Pretend you’re a spy trying to identify communications from an enemy network hiding their messages in newspapers. Look for patterns and similarities, drawing circles around them. These are your modules. Start also highlighting the differences between each instance of these modules, working out which is the most basic or common type that will become the base definition from which all other representations are extended.

This simple but empowering exercise will equip you for your task of actually crafting, instead of just building, the front-end. Without the knowledge gained from this kind of research phase, you will be blundering forward, improvising as best you can, but ultimately making quality-compromising mistakes that could have been avoided.

For more on this theme, read Anna Debenham's Front-end Style Guides which recommends a similar process, and the sublime idea of extending this into a guide to refer to during development and beyond.

DESIGN HOMOGENEITY

Moving forward again, you now have your modules defined and things are looking good. I mentioned that many instances of these modules will carry minor differences. These differences must be given significant thinking time, and discussion time with your designer(s).

It should be common knowledge by now that successful software projects are not the product of distinct design and build phases with little or no bidirectional feedback. The crucial nature of the designer-developer relationship has been covered in depth this year by Paul Robert Lloyd, and a joint effort from both teams throughout the project lifecycle is pivotal to your ability to craft and ship successful products.

This relationship comes into play when you're well into the development of the site, and you start noticing these differences between instances of modules (they'll start to stand out very clearly to you and your carefully regimented modular CSS system). Before you start overriding your base styles, question the differences with the designer to work out why they exist. Perhaps they are required and are important to their context, but perhaps they were oversights from earlier design revisions, or simple mistakes.

THE CRAFTSPERSON'S GLAND

As you grow towards the levels of expertise and experience where you can proudly and honestly consider yourself a craftsperson, you will find that you steadily develop what initially feels like a kind of sixth sense. I think of it more as a new hormonal gland, secreting into your bloodstream a powerful messenger chemical that can either reward or punish your brain. This gland is connected directly to your core understanding of what good quality work looks and **feels** like, an understanding that itself improves with experience.

This gland will make itself known to you in two ways. First, when you solve a problem in a beautifully elegant way with clean and unobtrusive code that looks good and just feels right, your craftsperson's gland will ooze something delicious that makes your brain and soul glow from the inside out. You will beam triumphantly at the succinct lines of code on your computer display before bounding outside with a spring in your step to swim up glittering rainbows and kiss soft fluffy puppies.

The second way that you may become aware of your craftsperson's gland, though, is somewhat less pleasurable. In an alternate reality, your parallel self is faced with the same problem, but decides to take a shortcut and get around it by some dubious means – the kind of technical method that the words hack, kludge and bodge are reserved for. As soon as you have done this, or even as you are doing it, your craftsperson's gland will damn well let you know that you took the wrong fork in the road. As your craftsperson's gland begins to secrete a toxic pus, you will at first become entranced into a vacant stare at the monstrous mess you are considering unleashing upon your site's visitors, before writhing in the horrible agony of an itch that can never be scratched, and a feeling of being coated with the devil's own deep and penetrating filth that no shower will ever cleanse.

Perhaps I exaggerate slightly, but it is no overstatement to suggest that you will find yourself being guided by proverbial angels and demons perched on opposite shoulders, or a whispering voice inside your head. If you harness this sense, sharpening it as if it were another tool in your kit and letting it guide or at least advise your decision making, you will transcend the rocky realm of random trial and error when faced with problems, and tend toward the right answers instinctively.

This gland can also empower your ability to assess your own work, becoming a judge before whom all your work is cross-examined. A good craftsperson regularly takes a step back from their work, and questions every facet of their product for its precise alignment with their core values of quality and sincerity, and even the very necessity of each component.

THE WRAPPING

By now, you may be thinking that I take this kind of thing far too seriously, but to terrify you further, I haven't even shared the half of it. Hopefully, though, this gives you an idea of the kind of levels of professionalism and dedication that it should take to get you on your way to becoming a craftsperson. It's a level of accomplishment and ability toward which we all should strive, both for our personal fulfilment and the betterment of the products we use daily. I look forward to seeing your finely crafted work throughout 2012.

ABOUT THE AUTHOR

Ben Bodien is Co-Founder of Neutron Creations and a front-end development journeyman, and sometimes dabbler in interface design (when there are no grown-ups around to stop him). His employers and clients have ranged from video game companies to hedge funds and from bedroom and VC backed startups to publicly listed multinationals. His other loves include coffee, jazz and cocktails, often consumed in combination. You can observe him quizzically from a safe distance on Twitter @bbodien.

Photo: Stefan Nitzsche