



24

2012

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](http://edgeofmyseat.com) and produced by Rachel Andrew.
- Possible only with the help and dedication of our authors.

# 2012

---

During the same month that HTML5 was designated a Candidate Recommendation by the W3C, 24 ways covered issues of performance as part of responsive web design, CSS and preprocessing, responsive images (again) and design systems.

HTML5 Video Bumpers .....	5
Starting Your Project on the Right Foot (and Keeping It There) .....	18
Being Prepared To Contribute .....	26
Colour Accessibility .....	33
Responsive Responsive Design .....	51
Flashless Animation .....	61
Think First, Code Later .....	77
Giving CSS Animations and Transitions Their Place .....	86
Should We Be Reactive? .....	92
Fluent Design through Early Prototyping .....	102

Responsive Images: What We Thought We Needed.....	110
Design Systems .....	120
Redesigning the Media Query.....	130
Using Questionnaires for Design Research.....	140
A Harder-Working Class .....	158
How to Make Your Site Look Half-Decent in Half an Hour .....	174
Cut Copy Paste.....	193
Giving Content Priority with CSS3 Grid Layout .....	203
Direction, Distance and Destinations .....	230
Content Planning Demystified.....	239
Infinite Canvas: Moving Beyond the Page.....	247
Unwrapping the Wii U Browser .....	260
Monkey Business.....	278
Science! .....	284

# 1. HTML5 Video Bumpers

---

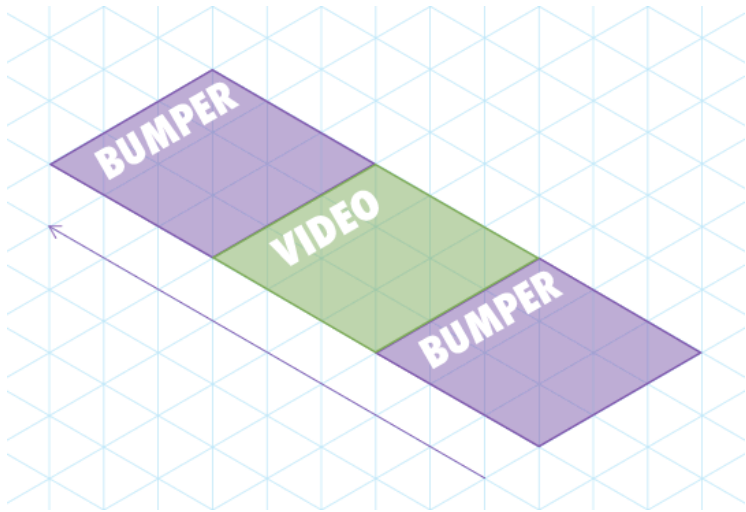
Drew McLellan

[24ways.org/201201](http://24ways.org/201201)

Video is a bigger part of the web experience than ever before. With native browser support for HTML5 video elements freeing us from the tyranny of plugins, and the availability of faster internet connections to the workplace, home and mobile networks, it's now pretty straightforward to publish video in a way that can be consumed in all sorts of ways on all sorts of different web devices.

I recently worked on a project where the client had shot some dedicated video shorts to publish on their site. They also had some five-second motion graphics produced to top and tail the videos with context and branding. This pretty common requirement is a great idea on the web, where a user might land at your video having followed a link and be viewing a page without much context.

Known as *bumpers*, these short introduction clips help brand a video and make it look a lot more professional.



## ADDING BUMPERS TO A VIDEO

The simplest way to add bumpers to a video would be to edit them on to the start and end of the video file itself. Cooking the bumpers into the video file is easy, but should you ever want to update them it can become a real headache. If the branding needs updating, for example, you'd need to re-edit and re-encode all your videos. Not a fun task.

What if the bumpers could be added dynamically? That would enable you to use the same bumper for multiple videos (decreasing download time for users who might watch more than one) and to update the bumpers whenever you wanted. You could change them seasonally,

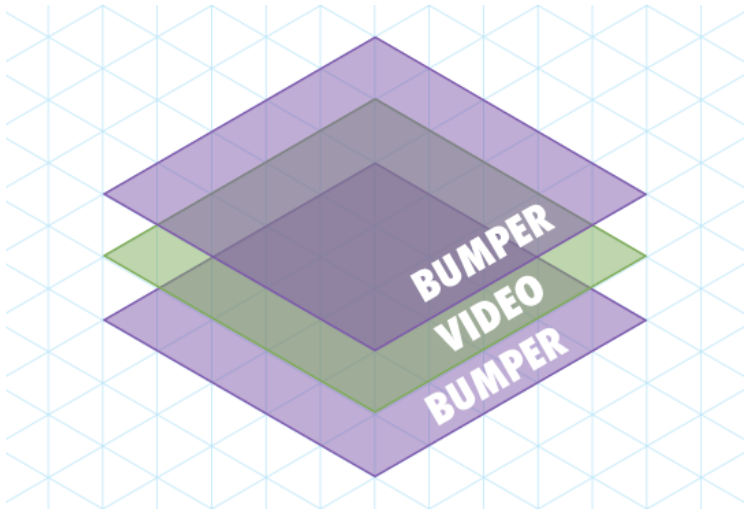
update them for special promotions, run different advertising slots, perform multivariate testing, or even target different bumpers to different users.

The trade-off, of course, is that if you dynamically add your bumpers, there's a chance that a user in a given circumstance might not see the bumper. For example, if the main video feature was uploaded to YouTube, you'd have no way to control the playback. As always, you need to weigh up the pros and cons and make your choice.

### **HTML5 BUMPERS**

If you wanted to dynamically add bumpers to your HTML5 video, how would you go about it? That was the question I found myself needing to answer for this particular client project.

My initial thought was to treat it just like an image slideshow. If I were building a slideshow that moved between images, I'd use CSS absolute positioning with `z-index` to stack the images up on top of each other in a pile, with the first image on top. To transition to the second image, I'd use JavaScript to fade the top image out, revealing the second image beneath it.



Now that video is just a native object in the DOM, just like an image, why not do the same? Stack the videos up with the opening bumper on top, listen for the video's onended event, and fade it out to reveal the main feature behind. Good idea, right?

## **WRONG**

Remember that this is the web. It's never going to be that easy. The problem here is that many non-desktop devices use native, dedicated video players. Think about watching a video on a mobile phone – when you play the video, the phone often goes full-screen in its native player, leaving the web page behind. There's no opportunity to fade or switch z-index, as the video isn't being viewed in the page. Your page is left powerless. Powerless!





So what can we do? What can we control?

Those of us with particularly long memories might recall a time before CSS, when we'd have to use JavaScript to perform image rollovers. As CSS background images weren't a practical reality, we would use lots of `<img>` elements, and perform a rollover by modifying the `src` attribute of the image.

Turns out, this old trick of modifying the source can help us out with video, too. In most cases, modifying the `src` attribute of a `<video>` element, or perhaps more likely the `src` attribute of a source element, will swap from one video to another.

## SWAPPIN' IT

Let's take a deliberately simple example of a super-basic video tag:

```
<video src="mycat.webm" controls>no fallback coz i is  
lame, innit.</video>
```

We could very simply write a script to find all video tags and give them a new src to show our bumper.

```
<script>  
  var videos, i, l;  
  videos = document.getElementsByTagName('video');  
  for(i=0, l=videos.length; i<l; i++) {  
    videos[i].setAttribute('src', 'bumper-in.webm');  
  }  
</script>
```

View the example in a browser with WebM support. You'll see that the video is swapped out for the opening bumper. Great!

## BEEFING IT UP

Of course, we can't just publish video in one format. In practical use, you need a `<video>` element with multiple `<source>` elements containing your different source formats.

```
<video controls>  
  <source src="mycat.mp4" type="video/mp4" />  
  <source src="mycat.webm" type="video/webm" />  
  <source src="mycat.ogv" type="video/ogg" />  
</video>
```

This time, our script needs to loop through the sources, not the videos. We'll use a regular expression replacement to swap out the file name while maintaining the correct file extension.

```
<script>
    var sources, i, l, orig;
    sources = document.getElementsByTagName('source');
    for(i=0, l=sources.length; i<l; i++) {
        orig = sources[i].getAttribute('src');
        sources[i].setAttribute('src',
orig.replace(/(w+).(w+)/, 'bumper-in.$2'));
        // reload the video
        sources[i].parentNode.load();
    }
</script>
```

The difference this time is that when changing the src of a <source> we need to call the .load() method on the video to get it to acknowledge the change.

See the code in action, this time in a wider range of browsers.

## **BUT, MY VIDEO!**

I guess we should get the original video playing again. Keeping the same markup, we need to modify the script to do two things:

1. Store the original src in a data- attribute so we can access it later

2. Add an event listener so we can detect the end of the bumper playing, and load the original video back in

As we need to loop through the videos this time to add the event listener, I've moved the `.load()` call into that loop. It's a bit more efficient to call it only once after modifying all the video's sources.

```
<script>
var videos, sources, i, l, orig;
sources = document.getElementsByTagName('source');
for(i=0, l=sources.length; i<l; i++) {
    orig = sources[i].getAttribute('src');
    sources[i].setAttribute('data-orig', orig);
    sources[i].setAttribute('src',
orig.replace(/(w+).(w+)/, 'bumper-in.$2'));
}
videos = document.getElementsByTagName('video');
for(i=0, l=videos.length; i<l; i++) {
    videos[i].load();
    videos[i].addEventListener('ended', function(){
        sources = this.getElementsByTagName('source');
        for(i=0, l=sources.length; i<l; i++) {
            orig = sources[i].getAttribute('data-orig');
            if (orig) {
                sources[i].setAttribute('src', orig);
            }
            sources[i].setAttribute('data-orig', '');
        }
        this.load();
        this.play();
    });
}
```

```

    });
}
</script>

```

Again, **view the example** to see the bumper play, followed by our spectacular main feature. (That's my cat, Widget. His interests include sleeping and internet marketing.)

## TIDYING THINGS UP

The final thing to do is add our closing bumper after the main video has played. This involves the following changes:

1. We need to keep track of whether the `src` has been changed, so we only play the video if it's changed. I've added the `modified` variable to track this, and it stops us getting into a situation where the video just loops forever.
2. Add an `else` to the event listener, for when the `orig` is `false` (so the main feature has been playing) to load in the end bumper. We also check that we're not already playing the end bumper. Because looping.

```

<script>
var videos, sources, i, l, orig, current, modified;
sources = document.getElementsByTagName('source');
for(i=0, l=sources.length; i<l; i++) {
    orig = sources[i].getAttribute('src');
    sources[i].setAttribute('data-orig', orig);
    sources[i].setAttribute('src',
orig.replace(/(w+).(w+)/, 'bumper-in.$2'));

```

```

}
videos = document.getElementsByTagName('video');
for(i=0, l=videos.length; i<l; i++) {
    videos[i].load();
    modified = false;
    videos[i].addEventListener('ended', function(){
        sources = this.getElementsByTagName('source');
        for(i=0, l=sources.length; i<l; i++) {
            orig = sources[i].getAttribute('data-orig');
            if (orig) {
                sources[i].setAttribute('src', orig);
                modified = true;
            }else{
                current = sources[i].getAttribute('src');
                if (current.indexOf('bumper-out')==1) {
                    sources[i].setAttribute('src',
current.replace(/([w]+).(w+)/, 'bumper-out.$2'));
                    modified = true;
                }else{
                    this.pause();
                    modified = false;
                }
            }
        }
        sources[i].setAttribute('data-orig', '');
    }
    if (modified) {
        this.load();
        this.play();
    }
});
}
</script>

```

Yo ho ho, that's a lot of JavaScript. **See it in action** – you should get a bumper, the cat video, and an end bumper.

Of course, this code works fine for demonstrating the principle, but it's very procedural. Nothing wrong with that, but to do something similar in production, you'd probably want to make the code more modular to ease maintainability. Besides, you may want to use a framework, rather than basic JavaScript.

## THE END CREDITS

One really important principle here is that of **progressive enhancement**. If the browser doesn't support JavaScript, the user won't see your bumper, but they will get the main video. If the browser supports JavaScript but doesn't allow you to modify the `src` (as was the case with older versions of iOS), the user won't see your bumper, but they will get the main video.

If a search engine or social media bot grabs your page and looks for content, they won't see your bumper, but they will get the main video – which is absolutely what you want.

This means that if the bumper is absolutely crucial, you may still need to cook it into the video. However, for many applications, running it dynamically can work quite well.

As always, it comes down to three things:

1. Measure your audience: know how people access your site
2. Test the solution: make sure it works for your audience
3. Plan for failure: it's the web and that's how things work 'round these parts

But most of all **play around with it, have fun and build something awesome.**

## **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](http://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. Starting Your Project on the Right Foot (and Keeping It There)

---

Bethany Heck

24ways.org/201202

I'm not sure if anything is as terrifying as beginning a new design project. I often spend hours trying to find the best initial footing in a design, so I've been working hard to improve my process, particularly for the earliest stages of a project. I want to smooth out the bumps that disrupt my creative momentum and focus on the emotional highs and lows I experience, and then try to minimize the lows and ride the highs as long as possible.

Design is often a struggle broken up by blissful moments of creative clarity that provide valuable force to move your work forward. Momentum is a powerful tool in creative work, and it's something we don't always maximize when we're working because of the hectic nature of our field. Obviously, every designer is going to have a different process, but I thought I'd share some of

the methods I've begun to adopt. I hope this will spark a conversation among designers who are interested in looking at process in a new way.

### **Jump-starting a project**

I cannot overstate the importance of immersing yourself in design and collecting ample amounts of inspiration when beginning a project. I make it a daily practice to visit a handful of sites (Dribbble, Graphic Exchange, Web Creme, siteInspire, Designspiration, and others) and save any examples of design that I like. I then sort them into general categories (publication design, illustration, typography, web design, and so on). Enjoying a bit of fresh design every day helps me absorb it and analyze why it's effective instead of just imitating it.

Many designers are afraid to look at too much design for fear that they'll be tempted to copy it, but I feel a steady influx of design inspiration reduces that possibility. You're much more likely to take the easy way out and rip off a design if you're scrambling for inspiration after getting stuck. If you are immersed in design from a variety of mediums, you'll engage your creative brain on multiple levels and have an easier time creating something unique for your project. Looking at good design will not make you a good designer but it will make you a better designer.

## Design is design

Try not to limit your visual research to the medium you're working in. Websites, books, posters and packaging all have their own unique limitations and challenges, and any one of those characteristics could be useful to you.

Posters need to grab the viewer and pass on a small tidbit of information; packaging needs to encourage physical interaction; and websites need to encourage exploration. If you know the challenges you'll be facing, you will know where to look for design that tackles those same problems.

I find it refreshing to look at design from the turn of the nineteenth century, when type was laid out on objects without thought to aesthetics. Many vintage packages break all sorts of modern design rules, and looking at that kind of work is a great way to spark your creativity. Pulling yourself out of the box and away from the rules of what you're working on can reveal solutions that are innovative and unique. After a little finessing, the warning label text from a 1940s hazardous chemical box from could have the exact type and icon arrangement you need for your project. There's a massive pool of design to pull from that doesn't have the limitations the web has, and exploring those design worlds will help you grow your own repertoire.

### **If all else fails, start with the footer**

The very beginning of a project is the most frustrating point in a project for me. I'm trying to figure out typeface combinations, colors and the overall voice of the design, and until I find the right solutions, I'm a wreck. I've found often that my frustration stems from trying to solve too many problems at once. The beginning of a project has a lot of moving targets, nearly endless possible solutions, and constantly changing variables. You'll knock out one problem only to discover your solution doesn't jive with something you worked out earlier — you end up designing in circles.

If you find yourself getting stuck at the beginning of a website design, try working out one specific element of the site and see what emerges. I'm going to recommend the footer. Why? Footers can easily be ignored in a design or become a dumping ground for items that couldn't be worked into the main layout. But, at the start of most projects, the minimum content requirements for the footer are usually established. There needs to be a certain number of links, social media buttons, copyright details, a search bar, and so on. It's a self-contained item within the design that has a specific purpose, and that's a great element to focus on when you're stuck in a design. Colors, typefaces, link styles, input fields and buttons can all be sketched out from just the footer. It's a very flexible

element that can be as prominent or subtle as you want, and it's a solid starting point for setting the tone and style of a site.

## **Save the details**

Designers love details. I love details. But don't let nitpicking early on in your process kill your creative momentum. Design is an emotional process, and being frustrated or defeated by a tricky problem or a graphical detail you just can't nail down can deflate your creative energies. If you hit a roadblock, set it aside and tackle another piece of the project. As you spend time engaged in a design, the style you develop will evolve according to the needs of the content, and you might arrive naturally at a solution that will work perfectly for the problem that had you stuck before.

If I find myself working on one particular element for more than a half an hour without any clear movement, I shelve it. Designers often wear their obsessive detail-oriented tendencies as a badge of honor, but there's a difference between making the design better and wasting time. If you've spent hours nudging elements around pixel by pixel and can't settle on something, it probably means what you're doing isn't making a huge improvement on the design. Don't be afraid to let it lie and come at it again

with fresh eyes. You will be better equipped to tackle the finer points of a project once you've got the broad strokes defined.

### **Have a plan when you start and stop designing**

We all know that creativity isn't something you can turn on effortlessly, and it's easy to forget the emotional process that goes along with design. If you leave a project in a place of frustration, it's going to stay with you in your free time and affect you negatively, like a dark cloud of impending disaster. Try to end each design session with a victory, a small bit of definable progress that you can take with you in your downtime. Even something as small as finding the right opacity for the interior shadow on the search bar in the header of the site is a win. Likewise, when you return to a project after a break, it can be difficult to get the ball rolling on the design again if you set it down without a clear path for the next steps. I find that I work on details best when I'm returning from downtime, when I'm fresh and re-energized and ready to dig in again. Try to pick out at least one element you'd like to fine-tune when you are winding down in a design session and use it to kick-start your next session.

## Content is king

I would argue there is nothing more crucial to the success of a design than having the content defined from the outset. Designing without content is similar to designing without an audience, and designing with vague ideas of content types and character limits is going to result in a muted design that doesn't reach its full potential. Images and language go hand in hand with design, and can take a design from functional to outstanding if you have them available from the outset. We don't always have the luxury of having content to build a design around, but fight for it whenever you can. For example, if the site you are designing is full of technical jargon, your paragraphs might need a longer line length to accommodate the longer words being used.

Often, working with content will lead to design solutions you wouldn't have come to otherwise. Design speaks to content, and content speaks to design. Lorem ipsum doesn't speak to anyone (unless you know Latin, in which case, congratulations!).

Every project has its own set of needs, and every designer has his or her own method of working. There's obviously no perfect process to design, and being dogmatic about process can be just as harmful as not having one. Exposing yourself to new design and new ways of designing is an easy way to test your skills and grow. When things are hard and you can't get any momentum going on a design,



## Starting Your Project on the Right Foot (and Keeping It There)

this is when your skill set is truly challenged. We all hope to get wonderful projects with great assets and ample creative possibilities, but you won't always be so blessed, and this is when the quality of your process is really going to shine.

### ABOUT THE AUTHOR



**Bethany Heck** is a designer working in (famously hot) Columbia, SC at the fabulous web design firm known as **Cyberwoven**. She went to Auburn University, and she's telling you this because it's got the best design program in the country. She is resisting the urge to be clever in her bio. She runs the **Eephus League of Baseball Minutiae** (she's really into sports) and shares her latest work on **Dribbble**.

# 3. Being Prepared To Contribute

---

Trent Walton

24ways.org/201203

“You’ll figure it out.” The advice my dad gives has always been the same, whether addressing my grade school homework or paying bills after college. If I was looking for a shortcut, my dad wasn’t going to be the one to provide it.

When I was a kid it infuriated the hell out of me, but what I then perceived to be a lack of understanding turned out to be a keystone in my upbringing. As an adult, I realize the value in not receiving outright solutions, but being forced to figure things out.

Even today, when presented with a roadblock while building for the web, I am tempted to get by with the help of the latest grid system, framework, polyfill, or plugin. In and of themselves these resources are harmless, but before I can drop them in, those damn words still echo in the back of my mind: “You’ll figure it out.”

I know that if I blindly implement these tools as drag and drop solutions I fail to understand the intricacies behind how and why they were built; repeatedly using them as shortcuts handicaps my skill set. When I solely rely on the tools of others, my work is at their mercy, leaving me less creative and resourceful, and, thus, less able to contribute to the advancement of our industry and community.

One of my favorite things about this community is how generous and collaborative it can be. I've loved seeing **FitVids** used all over the web and regularly improved upon at Github. I bet we can all think of a time where implementing a shared resource has benefitted our own work and sanity. Because these resources are so valuable, it's important that we continue to be a part of the conversation in order to further develop solutions and ideas. It's easy to assume there's someone smarter or more up-to-date in any one area, but with a degree of understanding and perspective, we can all participate.

This open form of collaboration is in our web DNA. After all, its primary purpose was to promote the exchange and development of new ideas.

Tim Berners-Lee proposed a global hypertext project, to be known as the World Wide Web. Based on the earlier “Enquire” work, it was designed to allow people to work together by combining their knowledge in a web of hypertext documents.

I’m delighted to find that this spirit of collaborative ingenuity is alive and well on the web today. Take the story of **Off Canvas** as an example. I was at an ATX Dribbble meet up where I met Jason Weaver and chatted to him about his recent work on the responsive layout prototype, Off Canvas. Jason said he came across a **post** by **Luke Wroblewski** outlining the idea and saw this:

If anyone is interested in building a complete example of this approach using responsive Web design techniques, let me know!

From there Luke recounts:

We went back and forth on email, with me laying out ideas and Jason doing all the hard work to see if they can be done and improving them bit by bit! Once we got to something we both liked, I wrote up an article explaining things and he hosted the examples.

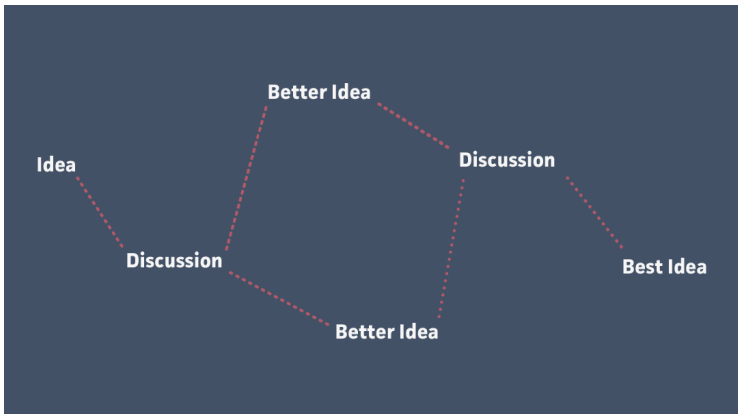
Luke took the time to clearly outline and diagram his ideas, and Jason responded with a solid proof of concept that has evolved into a tool we all have at our disposal. Victory!

I have also benefitted from comrades who have taken an idea of mine into development. After blogging about some concerns in regards to **maintaining hierarchy as media queries are used to shift layouts**, Jordan Moore rebounded with some **responsive demos** where he used flexbox to (re)order content as viewport sizing changes.

Similar stories can be found behind the development of things like **FitVids**, **FitText**, and **Molten Leading**. I love this pattern of collaboration because it involves a fairly specific process:

1. Initial idea or prototype is outlined or built, then shared
2. Discuss
3. Someone develops or improves it, then shares it
4. Discuss
5. Someone else develops or improves it, then shares it.
6. Infinity.

This is what the web looks like when we build it together, and I'd argue that steps 2+ are absolutely crucial. A web where everyone develops their own ideas and tools independent of one another is like a room full of people talking and no one listening.



The pattern itself mimics a literal web structure, and ideally we'd be able to follow a strand from one idea to the next and so on.

## **BLESSED ARE THE CURATORS**

Sometimes those lines aren't easy to find or follow. Thankfully, there are people who painstakingly log each experiment and index much of what's out there. Chris Coyier does this with CSS in general, and Brad Frost is doing this for responsive and multi-device design with his **Pattern Library**. Seriously, take a look at this page and imagine what it would take to find, track and organize the progression of each of these resources yourself. I'd argue that ongoing collections like these are more valuable than the sum of their parts when they are updated regularly as opposed to a top ten tips blog post format.

## HERE'S MY SOAPBOX

Here are a few things I appreciate about how things are shared and contributed online. And yes, I could do way better at all of them myself.

- Concise write-ups: honor others' time by getting to the point. Not every idea or solution needs two thousand words to convey fully. I love long-form posts, but there's a time and a place for them.
- Visual aids: if a quick illustration, screenshot, or graphic helps illustrate your point or problem, yes please.

By the way, [Luke Wroblewski](#) rules the school on both of these.

- Demo it: host it yourself, or put it on [CodePen](#) or [JS Bin](#) for others to see.
- Put it on Github: share and improve with the rest of the community. Consider, however, that because someone puts something on Github doesn't mean they're forever bound to provide support or instruction.

This isn't a call for everyone to learn everything all the time, but if you're curious or interested in something, skip the shortcut and get your hands dirty: sketch, prototype, question, debate, fork, and share. Figuring these things out on our own makes us valuable contributors to the web – the thing that ultimately we're all trying to figure out together.

## ABOUT THE AUTHOR



**Trent Walton** is founder and 1/3 of **Paravel**, a custom web design and development shop based out of the Texas Hill Country whose wife has put him on a font allowance. In his spare time, he writes about what he learns at his **blog**, and on **Twitter**.



## 4. Colour Accessibility

---

Geri Coady

24ways.org/201204

Here's a quote from Josef Albers:

In visual perception a colour is almost never seen as it really is[...] This fact makes colour the most relative medium in art.

*Josef Albers, Interaction of Color, 1963*

Albers was a German abstract painter and teacher, and published a very famous course on colour theory in 1963. Colour is very relative – not just in the way that it appears differently across different devices due to screen quality and colour management, but it can also be seen differently by different people – something we really need to be more mindful of when designing.

### **WHAT IS COLOUR BLINDNESS?**

Colour blindness very rarely means that you can't see any colour at all, or that people see things in greyscale. It's actually a decreased ability to see colour, or a decreased ability to tell colours apart from one another.

## HOW DOES IT HAPPEN?

Inside the typical human retina, there are two types of receptor cells – rods and cones. Rods are the cells that allow us to see dark and light, and shape and movement. Cones are the cells that allow us to perceive colour. There are three types of cones, each responsible for absorbing blue, red, and green wavelengths in the spectrum.

Problems with colour vision occur when one or more of these types of cones are defective or absent entirely, and these problems can either be inherited through genetics, or acquired through trauma, exposure to ultraviolet light, degeneration with age, an effect of diabetes, or other factors.

Colour blindness is a sex-linked trait and it's much more common in men than in women. The most common type of colour blindness is called deuteranomaly which occurs in 7% of males, but only 0.5% of females. That's a pretty significant portion of the population if you really stop and think about it – we can't ignore this demographic.

## WHAT DOES IT LOOK LIKE?

People with the most common types of colour blindness, like protanopia and deuteranopia, have difficulty discriminating between red and green hues. There are also forms of colour blindness like tritanopia, which

affects perception of blue and yellow hues. Below, you can see what a colour wheel might look like to these different people.



## WHAT CAN WE DO?

Here are some things you can do to make your websites and apps more accessible to people with all types of colour blindness.

## Include colour names and show examples

One of the most common annoyances I've heard from people who are colour-blind is that they often have difficulty purchasing clothing and they will sometimes need to ask another person for a second opinion on what the colour of the clothing might actually be. While it's easier to shop online than in a physical store, there are still accessibility issues to consider on shopping websites.

Let's say you've got a website that sells T-shirts. If you only show a photo of the shirt, it may be impossible for a person to tell what colour the shirt really is. For clarification, be sure to reference the name of the colour in the description of the product.



**\$24** + shipping 

Represent St. John's and the Local 709 with a **rose Cabot Tower on a Kelly Green** American Apparel 100% cotton tee. Rose UP logo on the back.

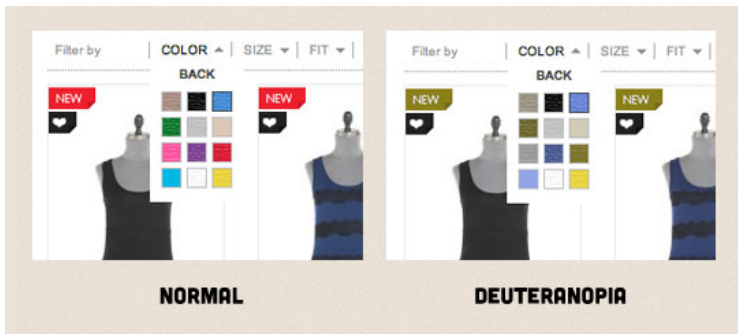
**Choose your size.** (Not sure? Check your measurements: **girls, guys**)

<b>MEN S / SOLD OUT</b>	<b>WOMEN S / SOLD OUT</b>
<b>MEN M / SOLD OUT</b>	<b>WOMEN M</b>
<b>MEN L</b>	<b>WOMEN L</b>
<b>MEN XL / SOLD OUT</b>	<b>WOMEN XL / SOLD OUT</b>
<b>MEN 2X / SOLD OUT</b>	

**ADD TO CART**

United Pixelworkers does a great job of following this rule. The St. John's T-shirt has a quirky palette inspired by the unofficial pink, white and green Newfoundland flag, and I can imagine many people not liking it.

Another common problem occurs when a colour filter has been added to a product search. Here's an example from a clothing website with unlabelled colour swatches, and how that might look to someone with deuteranopia-type colour blindness.



The colour search filter below, from the H&M website, is much better since it uses names instead.

## TOPS

[HM.COM](#) / [LADIES](#) / TOPS

▼ **HIDE FILTER**

CLEAR FILTER

### COLOUR

- |                                |                                    |                                 |
|--------------------------------|------------------------------------|---------------------------------|
| <input type="checkbox"/> White | <input type="checkbox"/> Turquoise | <input type="checkbox"/> Silver |
| <input type="checkbox"/> Black | <input type="checkbox"/> Purple    | <input type="checkbox"/> Gold   |
| <input type="checkbox"/> Green | <input type="checkbox"/> Red       | <input type="checkbox"/> Orange |
| <input type="checkbox"/> Grey  | <input type="checkbox"/> Blue      | <input type="checkbox"/> Beige  |
| <input type="checkbox"/> Pink  | <input type="checkbox"/> Yellow/   | <input type="checkbox"/> Brown  |

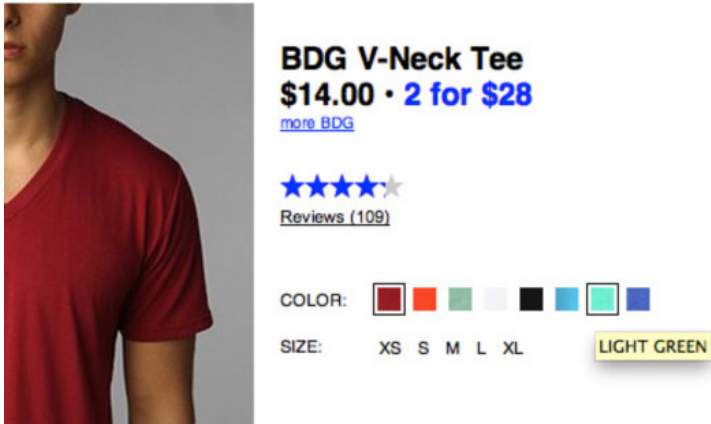
### CONCEPTS

- |                                  |                                   |
|----------------------------------|-----------------------------------|
| <input type="checkbox"/> TREND   | <input type="checkbox"/> L.O.G.G. |
| <input type="checkbox"/> &DENIM  | <input type="checkbox"/> MAMA     |
| <input type="checkbox"/> DIVIDED |                                   |

MATCHING STYLES: 67



At first glance, **Urban Outfitters** also uses unlabelled colour swatches on product pages (below), but on closer inspection, the colour name is displayed on hover. This isn't an ideal solution, because although it'll work on a desktop browser, it won't work on a touchscreen device where hovering isn't an option.



Using overly fancy colour names, like the ones you might find labelling high-end interior paint can be just as confusing as not using a colour name at all. Names like grape instead of purple don't really give the viewer any useful information about what the colour actually is on a colour wheel. Is grape supposed to be purple, or could it refer to red grapes or even green? Stick with hue names as much as possible.

### **Avoid colour-specific instructions**

When designing forms, avoid labelling required fields only with coloured text. It's safer to use a symbol cue like the asterisk which is colour-independent.

REQUIRED FIELDS ARE INDICATED IN RED	* REQUIRED FIELD
FIRST NAME <input type="text"/>	FIRST NAME <input type="text"/>
LAST NAME <input type="text"/>	LAST NAME <input type="text"/>
TELEPHONE <input type="text"/>	* TELEPHONE <input type="text"/>

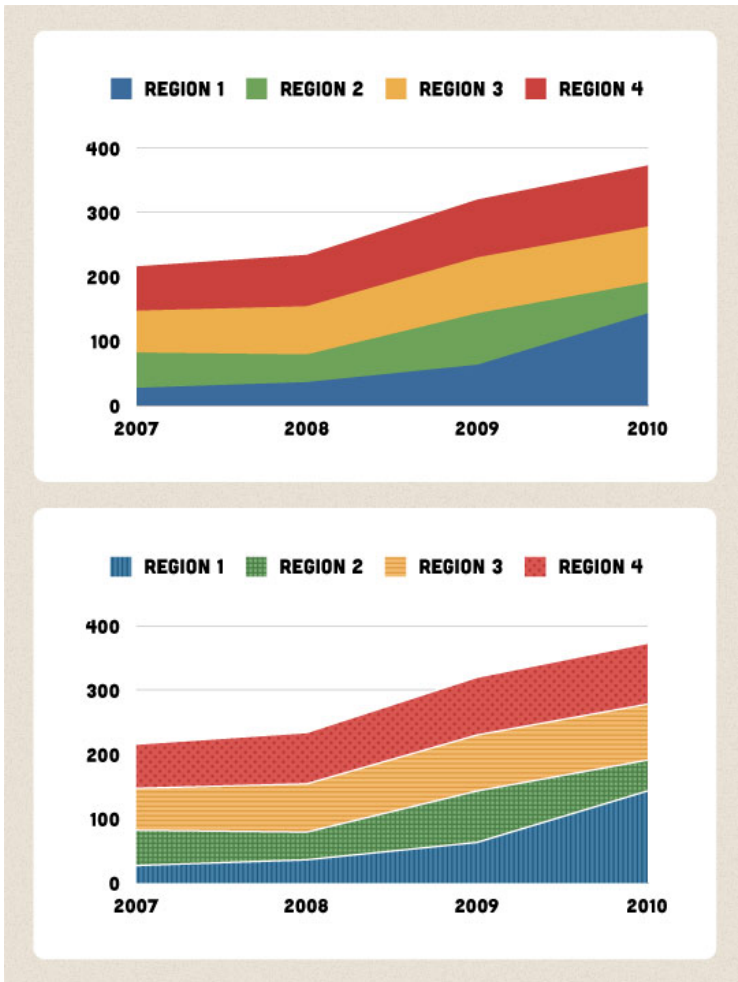
A similar example would be directing a user to click a green button to purchase a product. Label your buttons clearly and reference them in the site copy by function, not colour, to avoid confusion.

### Don't rely on colour coding

Designing accessible maps and infographics can be much more challenging.

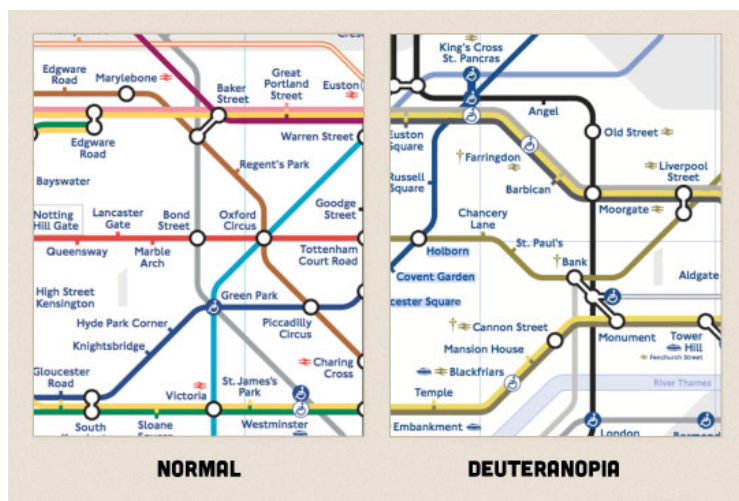
Don't rely on colour coding alone – try to use a combination of colour and texture or pattern, along with precise labels, and reflect this in the key or legend. Combine a blue background with a crosshatched pattern, or a pink background with a stippled dot – your users will always have two pieces of information to work with.



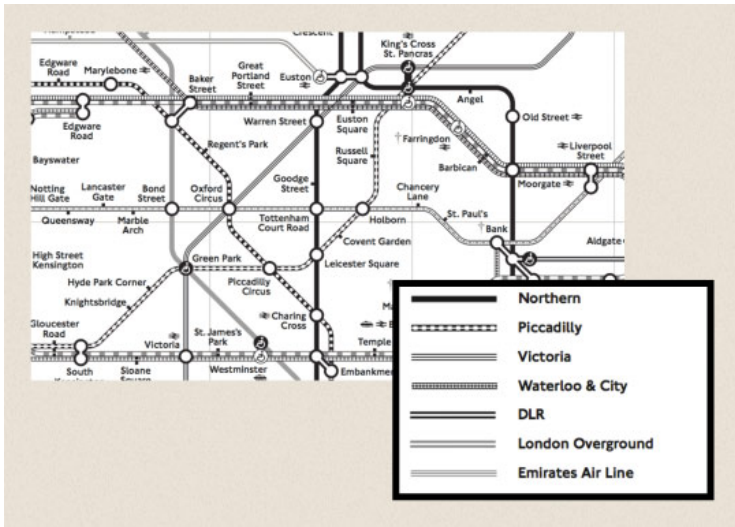


The map of the London subway system is an iconic image not just in London, but around the world. Unfortunately, it contains some colours that are indistinguishable from each other to a person with a vision problem. This is true

not only for the London underground, but also for any other wayfinding system that relies on colour coding as the only key in a legend.



There are printable versions of the map available online in black and white, using patterns and shades of black and grey that are distinguishable, but the point is that there would be no need for such a map if it were designed with accessibility in mind from the beginning. And, if you're a person who has a physical disability as well as a vision problem, the "Step-Free" guide map which shows stations is based on the original coloured map.



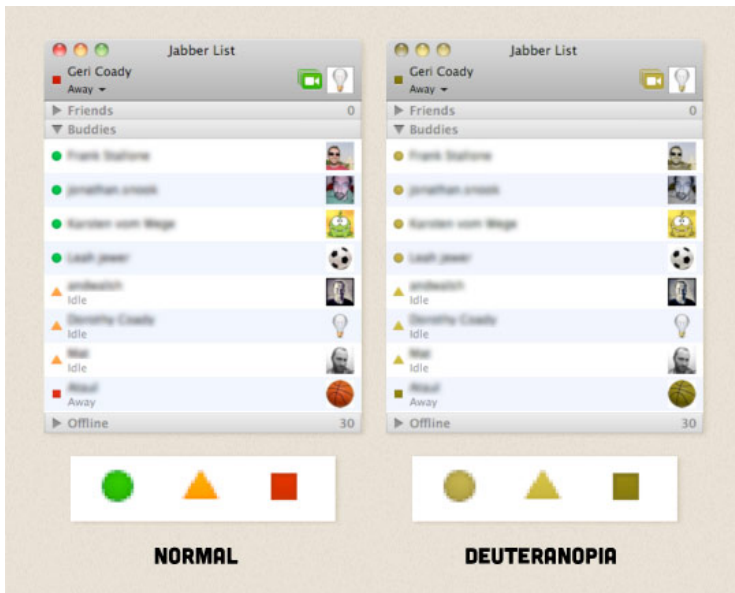
## Provide alternatives and customization

While it's best to consider these issues and design your app to be accessible by default, sometimes this might not be possible. Providing alternative styles or allowing users to edit their own colours is a feature to keep in mind.

The developers of the game **Faster Than Light** created an alternate colour-blind mode and asked for public feedback to make sure that it passed the test. Not much needed to be done, but you can see they added stripes to the red zones and changed some outlines to blue.



iChat is also a good example. Although by default it uses coloured bubbles to indicate a user's status (available for chat, away or idle, or busy), included in the preferences is a "User Shapes to Indicate Status" option, which changes the shape of the standard circles to green circles, yellow triangles and red squares.



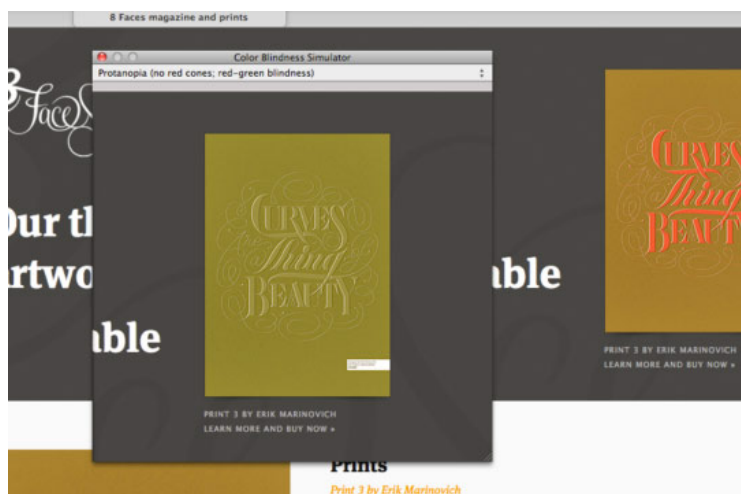
### Pay attention to contrast

Colours that are similar in value but different in hue may be easy to distinguish between for a user with good vision, but a person who suffers from colour blindness may not be able to tell them apart at all. Proofing your work in greyscale is a quick way to tell if there's enough contrast between the most important information in your design.

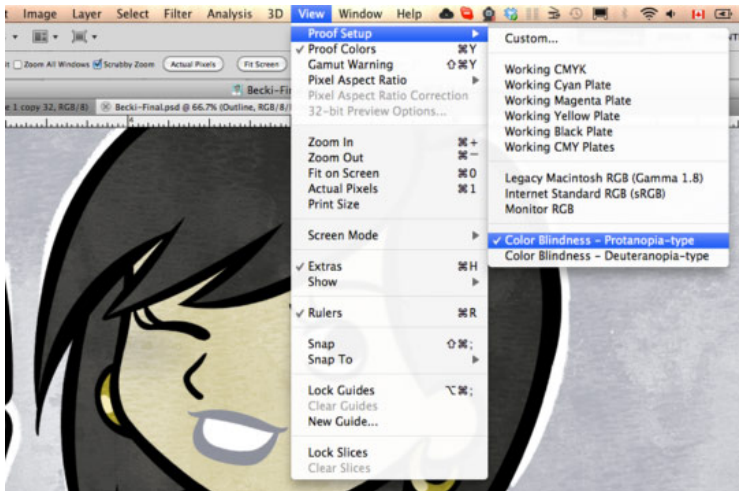
### Check with a simulator

There are many tools out there for simulating different types of colour blindness, and it's worth checking your design to catch any potential problems up front.

One is called **Sim Daltonism** and it's available for Mac OS X. It'll show a pop-up preview next to your cursor and you can choose which type of colour blindness you want to test from a drop-down menu.



You can also proof for the two most common types of colour blindness right in Photoshop or Illustrator (CS4 and later) while you're designing.



The colour contrast check tool from designer and developer Jonathan Snook gives you the option to enter a colour code for a background, and a colour code for text, and it'll tell you if the colour contrast ratio meets the Web Content Accessibility Guidelines 2.0. You can use the built-in sliders to adjust your colours until they meet the compliant contrast ratios. This is a great tool to test your palette before going live.

The screenshot shows a color contrast checker interface. On the left, there are two columns for color selection: 'Foreground Colour' (hex #819EB1) and 'Background Colour' (hex #2A2C54). Each column has sliders for Red, Green, Blue, Hue (°), Saturation (%), and Value (%). To the right is a 'Results' table with the following data:

This is example text. Some of it bolded. Some of it italicized.	
Brightness Difference: (>= 125)	104.13
Colour Difference: (>= 500)	296
Are colours compliant?	NO
Contrast Ratio	4.72
WCAG 2 AA Compliant	YES
WCAG 2 AA Compliant (18pt+)	YES
WCAG 2 AAA Compliant	NO
WCAG 2 AAA Compliant (18pt+)	YES

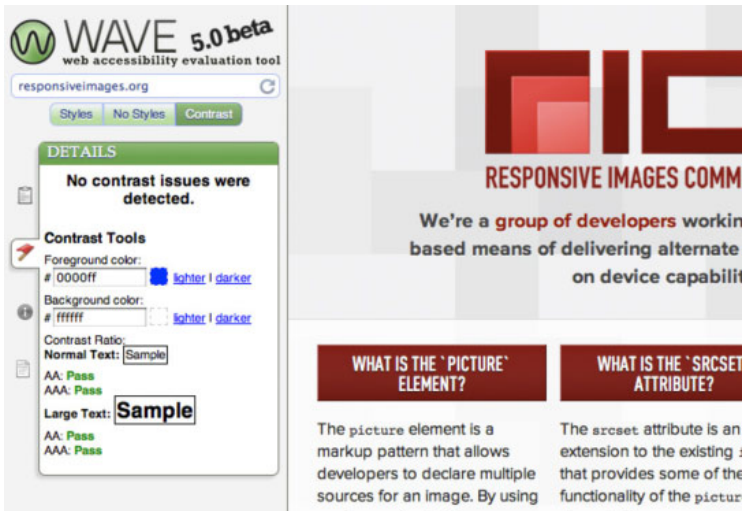
Below the results is a dark blue box containing the text: "This is example text. Some of it bolded. Some of it italicized." Below this box, the tool displays the following results:

<b>CONTRAST RATIO</b>	<b>4.72</b>
<b>ARE COLOURS COMPLIANT?</b>	<b>NO</b>

For live websites, you can use the accessibility tool called **WAVE**, which also has a contrast checker. It's important to keep in mind, though, that while **WAVE** can identify contrast errors in text, other things can slip through, so a site that passes the test does not automatically mean it's accessible in reality.

For example, the contrast checker here doesn't notice that our red link in the introduction isn't underlined, and therefore could blend into the surrounding paragraph text.





I know that once I started getting into the habit of checking my work in a simulator, I became more mindful of any potential problem areas and it was easier to avoid them up front. It's also made me question everything I see around me and it sends red flags off in my head if I think it's a serious colour blindness fail. Understanding that colour is relative in the planning stages and following these tips will help us make more accessible design for all.

## ABOUT THE AUTHOR



**Geri Coady** is a colour-obsessed illustrator and designer from Newfoundland, Canada. She is a former Art Director at a Canadian advertising agency and is now pursuing her own clients through her website at [hellogeri.com](http://hellogeri.com). Geri loves chatting about nerdy things on **Twitter** and has shared her thoughts in publications such as *net* magazine, *The Pastry Box Project*, and *Digital Arts*. She's the author of the *Pocket Guide to Colour Accessibility* from *Five Simple Steps*, a sometimes-illustrator for *A List Apart*, and was voted *Net Magazine's Designer of the Year* in 2014.

## 5. Responsive Responsive Design

---

Tim Kadlec

24ways.org/201205

Now more than ever, we're designing work meant to be viewed along a gradient of different experiences. Responsive web design offers us a way forward, finally allowing us to "design for the ebb and flow of things."

With those two sentences, Ethan closed the article that introduced the web to responsive design. Since then, responsive design has taken the web by storm. Seemingly every day, some company is touting their new responsive redesign. Large brands such as Microsoft, Time and Disney are getting in on the action, blowing away the once common criticism that responsive design was a technique only fit for small blogs.

Certainly, this is a good thing. As Ethan and **John Allsopp** before him, were right to point out, the inherent flexibility of the web is a feature, not a bug. The web's unique ability

to be consumed and interacted with on any number of devices, with any number of input methods is something to be embraced.

But there's one part of the web's inherent flexibility that seems to be increasingly overlooked: the ability for the web to be interacted with on any number of networks, with a gradient of bandwidth constraints and latency costs, on devices with varying degrees of hardware power.

A few months back, Stephanie Rieger **tweeted**

“Shoot me now...responsive design has seemingly become confused with an opportunity to reduce performance rather than improve it.”

I would love to disagree, but unfortunately the evidence is damning. Consider the size and number of requests for four highly touted responsive sites that were launched this year:

- 74 requests, 1,511kb
- 114 requests, 1,200kb
- 99 requests, 1,298kb
- 105 requests, 5,942kb

And those numbers were for the small screen versions of each site!

These sites were praised for their visual design and responsive nature, and rightfully so. They're very easy on the eyes and a lot of thought went into their appearance. But the numbers above tell an inconvenient truth: for all the time spent ensuring the visual design was airtight, seemingly very little (if any) attention was given to their performance.

It would be one thing if these were the exceptions, but unfortunately they're not. Guy Podjarny, who has done a lot of research around responsive performance, discovered that 86% of the responsive sites he tested were either **the same size or larger on the small screen as they were on the desktop.**

The reality is that high performance should be a requirement on any web project, not an afterthought. Poor performance has been tied to a **decrease in revenue, traffic, conversions, and overall user satisfaction.** Case study after case study shows that improving performance, even marginally, will impact the bottom line. The situation is no different on mobile where **71% of people say they expect sites to load as quickly or faster on their phone when compared to the desktop.**

The bottom line: **performance is a fundamental component of the user experience.**

So, given it's extreme importance in the success of any web project, why is it that we're seeing so many bloated responsive sites?

First, I adamantly disagree with the belief that poor performance is inherent to responsive design. That's not a rule – it's a cop-out. It's an example of blaming the technique when we should be blaming the implementation. This argument also falls flat because it ignores the fact that the trend of fat sites is increasing on the web in general. While some responsive sites are the worst offenders, it's hardly an issue resigned to one technique.

To fix the issue, we need to stop making excuses and start making improvements instead. Here, then, are some things we can do to start improving the state of responsive performance, and performance in general, right now.

## **CREATE A CULTURE OF PERFORMANCE**

If you understand just how important performance is to the success of a project, the natural next step is to start creating a culture where high performance is a key consideration.

One of the things you can do is set a baseline. Determine the maximum size and number of requests you are going to allow, and don't let a page go live if either of those numbers is exceeded. The BBC does this with its responsive mobile site.

A variation of that, which Steve Souders discussed in a recent podcast is to create a performance budget based on those numbers. Once you have that baseline set, if someone comes along and wants to add something to the page, they have to make sure the page remains under budget. If it exceeds the budget, you have three options:

1. Optimize an existing feature or asset on the page
2. Remove an existing feature or asset from the page
3. Don't add the new feature or asset

The idea here is that you make performance part of the process instead of something that may or may not get tacked on at the end.

## **EMBRACE THE PAIN**

This troubling trend of web bloat can be blamed in part on the lack of pain associated with poor performance. Most of us work on high-speed connections with low latency. When we fire up a 4Mb site, it doesn't feel so bad.

When I tested the previously mentioned 5,942kb site on a 3G network, it took over 93 seconds to load. A minute and a half just staring at a white screen. Had anyone working on that project experienced that, you can bet the site wouldn't have launched in that state.

Don't just crunch numbers. Fire up your site on a slower network and see what it feels like to wait. If you don't have access to a slow network, simulate one using a tool like **Slowly**, **Throttle** or the **Network Conditioner** found in Mac OS X 10.7.

## **WATCH FOR LOW-HANGING FRUIT**

There are a bunch of general performance improvements that apply to any site (responsive or not) but often aren't made. A great starting point is to refer to **Yahoo!**'s **list of rules**.

Some of this might sound complicated or intimidating, but it doesn't have to be. You can grab an **.htaccess file** from **HTML 5 Boilerplate** or use **Sergey Chernyshev's drop-in .htaccess file**. You can use tools like **SpriteMe** to simplify the creation of sprites, and **ImageOptim** to compress images.

Just by implementing these simple optimizations you will achieve a noticeable improvement in terms of weight and page load time.



## BE CAREFUL WITH IMAGES

The most common offender for poor responsive performance is downloading unnecessarily large images, or worse yet, multiple sizes of the same image.

For background images, simply being careful with **where and how you include the image** can ensure you don't get caught in the trap of multiple background images being downloaded without being used. Don't count on `display:none` to help. While it may hide elements from displaying on screen, those images will still be requested and downloaded.

Content images can be a little trickier. Whatever you do, don't serve a large image that works on a large screen display to small screens. It's wasteful, not only in terms of adding weight to the page, but also in wasting precious memory. Instead, use a tool like **Adaptive Images** or **src.sencha.io** to make sure only appropriately sized images are being downloaded.

The new `<picture>` element that has been so often discussed is another excellent solution if you're feeling particularly future-oriented. A **picture polyfill** exists so that you can start using the element now without any worries about support.

## CONDITIONAL LOADING

Don't load any more than you absolutely need to. If a script isn't needed at certain sizes, use the `matchMedia polyfill` to ensure it only loads when needed. Use `eCSSential` to do the same for unnecessary CSS files.

Last year on 24 ways, Jeremy Keith wrote an [article about conditional loading of content](#) in a responsive design based on the screen width. The technique was later refined by the Filament Group into what they dubbed the [Ajax-Include Pattern](#). It's a powerful and simple way to lighten the load on small screens as well as reduce clutter.

## GO VANILLA?

If you take a look at the [HTTP Archive](#) you'll see that other than image size, JavaScript is the heaviest asset on a page weighing in at 215kb on average. It also boasts the fifth highest correlation to load time as well as the second highest correlation to render time.

Much of the weight can be attributed to our industry's increasing reliance on frameworks. This is especially a concern on mobile devices. [PPK recently exclaimed](#) that current JavaScript libraries are just "too heavy for mobile". "Research from [Stoyan Stefanov](#) on [parse times](#) supports this. On some Android and iOS devices, it can take as long as 200-300ms just to `parse` jQuery.

There's nothing *wrong* about using a framework, but the problem is that they've become the default. Before dropping another framework or plugin into a page, we should stop to consider the value it adds and whether we could accomplish what we need to do using a combination of vanilla JavaScript and CSS instead. (This is a great example of a scenario where a performance budget could help.)

## **START THINKING BEYOND VISUAL AESTHETICS**

We love to tout the web's universality when discussing the need for responsive design. But that universality is not limited simply to screen size. Networks and hardware capabilities must factor in as well.

The web is an incredibly dynamic and interactive medium, and designing for it demands that we consider more than just visual aesthetics. Let's not forget to give those other qualities the attention they deserve.

## ABOUT THE AUTHOR



**Tim Kadlec** is a developer living in a tiny town in the north woods of Wisconsin. He's very passionate about the Web and can frequently be found speaking about what he's learned at a variety of web conferences.

Tim is the author of *Implementing Responsive Design: Building sites for an anywhere, everywhere web* (New Riders, 2012) and was a contributing author for the *Web Performance Daybook Volume 2* (O'Reilly, 2012). He writes sporadically at [timkadlec.com](http://timkadlec.com) and you can find him sharing his thoughts in a briefer format on Twitter at [@tkadlec](https://twitter.com/tkadlec).

## 6. Flashless Animation

---

Rachel Nabors

[24ways.org/201206](http://24ways.org/201206)

### ANIMATION IN A FLASHLESS WORLD

When I splashed down in web design four years ago, the first thing I wanted to do was animate a cartoon in the browser. I'd been drawing comics for years, and I've wanted to see them come to life for nearly as long. Flash animation was still riding high, but I didn't want to learn Flash. I wanted to learn JavaScript!

Sadly, animating with JavaScript was limiting and resource-intensive. My initial foray into an infinitely looping background did more to burn a hole in my CPU than amaze my friends (although it still looks pretty cool). And there was still no simple way to incorporate audio. The browser technology just wasn't there.

Things are different now. CSS3 transitions and animations can do most of the heavy lifting and HTML5 audio can serve up the music and audio clips. You can do a lot without leaning on JavaScript at all, and when you lean on JavaScript, you can do so much more!

In this project, I'm going to show you how to animate a simple walk cycle with looping audio. I hope this will inspire you to do something really cool and impress your friends. I'd love to see what you come up with, so please send your creations my way at [rachelnabors.com](http://rachelnabors.com)!

*Note: Because every browser wants to use its own prefixes with CSS3 animations, and I have neither the time nor the space to write all of them out, I will use the W3C standard syntaxes; that is, going prefix-less. You can implement them out of the box with something like **Prefixfree**, or you can add prefixes on your own. If you take the latter route, I recommend using **Sass** and **Compass** so you can focus on your animations, not copying and pasting.*

## THE WALK CYCLE

Walk cycles are the “Hello world” of animation. One of the first projects of animation students is to spend hours drawing dozens of frames to complete a simple loopable animation of a character walking.

Most animators don't have to draw every frame themselves, though. They draw a few **key frames** and send those on to production animators to work on the **between frames** (or tween frames). This is meticulous, grueling work requiring an eye for detail and natural movement. This is also why so much production animation gets shipped overseas where labor is cheaper.

Luckily, we don't have to worry about our frame count because we can set our own frames-per-second rate on the fly in CSS3. Since we're trying to impress friends, not animation directors, the inconsistency shouldn't be a problem. (Unless your friend is an animation director.)

This is a simple walk cycle I made of **my comic character Tuna** for my CSS animation talk at **CSS Dev Conference** this year:

The magic lies here:

```
animation: walk-cycle 1s steps(12) infinite;
```

Breaking those properties down:

```
animation: <name> <duration> <timing-function>  
<iteration-count>;
```

`walk-cycle` is a simple `@keyframes` block that moves the background sprite on `.tuna` around:

```
@keyframes walk-cycle {
  0% {background-position: 0 0; }
  100% {background-position: 0 -2391px;}
}
```

The background sprite has exactly twelve images of Tuna that complete a full walk cycle. We're setting it to cycle through the entire sprite every second, infinitely. So why isn't the background image scrolling down the `.tuna` container? It's all down to the timing function `steps()`. Using `steps()` let us tell the CSS to make jumps instead of the smooth transitions you'd get from something like `linear`. Chris Mills at dev.opera wrote in his excellent [intro to CSS3 animation](#) :

Instead of giving a smooth animation throughout, `[steps()]` causes the animation to jump between a set number of steps placed equally along the duration. For example, `steps(10)` would make the animation jump along in ten equal steps. There's also an optional second parameter that takes a value of `start` or `end`. `steps(10, start)` would specify that the change in property value should happen at the start of each step, while `steps(10, end)` means the change would come at the end.

(Seriously, go read his full article. I'm not going to touch on half the stuff he does because I cannot improve on the basics any more than he already has.)



## THE BACKGROUND

A cat walking in a void is hardly an impressive animation and certainly your buddy one cube over could do it if he chopped up some of those cat GIFs he keeps using in group chat. So let's add a **parallax** background! Yes, yes, all web designers signed a peace treaty to not abuse parallax anymore, but this is its true calling—treaty be damned.

And to think we used to need JavaScript to do this! It's still pretty CPU intensive but much less complicated. We start by splitting up the page into different layers, `.foreground`, `.midground`, and `.background`. We put `.tuna` in the `.midground`.

`.background` has multiple background images, all set to repeat horizontally:

```
background-image:
    url(background_mountain5.png),
    url(background_mountain4.png),
    url(background_mountain3.png),
    url(background_mountain2.png),
    url(background_mountain1.png);
background-repeat: repeat-x;
```

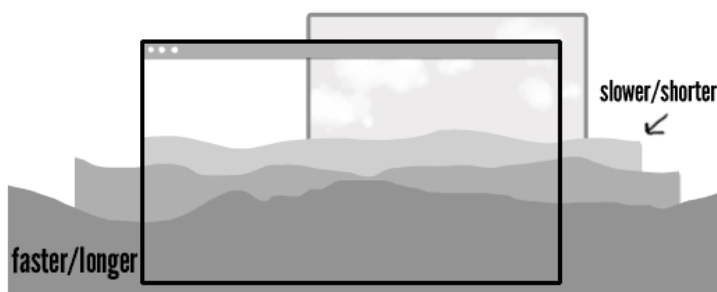
With parallax, things in the foreground move faster than those in the background. Next time you're driving, notice how the things closer to you move out of your field of vision faster than something in the distance, like a mountain or a large building. We can imitate that here by

making the background images on top (in the foreground, closer to us) wider than those on the bottom of the stack (in the distance).

The different lengths let us use one animation to move all the background images at different rates in the same interval of time:

```
animation: parallax_bg linear 40s infinite;
```

The shorter images have less distance to cover in the same amount of time as the longer images, so they move slower.



Let's have a look at the background's animation:

```
@keyframes parallax_bg {  
  0% {  
    background-position: -2400px 100%, -2000px 100%,  
-1800px 100%, -1600px 100%, -1200px 100%;  
  }  
  100% {  
    background-position: 0 100%, 0 100%, 0 100%, 0 100%,  
0 100%;  
  }  
}
```

```
0 100%;
  }
}
```

At 0%, all the background images are positioned at the negative value of their own widths. Then they start moving toward `background-position: 0 100%`. If we wanted to move them in the reverse direction, we'd remove the negative values at 0% (so they would start at `2400px 100%`, `2000px 100%`, etc.). Try changing the values in the codepen above or changing `background-repeat` to `none` to see how the images play together.

`.foreground` and `.midground` operate on the same principles, only they use single background images.

## THE MUSIC

After finishing the first draft of my original walk cycle, I made a GIF with it and **posted it on YTMND** with some music from the movie *Paprika*, specifically the track “The Girl in Byakkoya.” After showing it to some colleagues in my community, it became clear that this was a winning combination sure to drive away dresscode blues. So let's use HTML5 to get a clip of that music looping in there!

Warning, there is sound. Please adjust your volume or apply headphones as needed.



We're using HTML5 audio's loop and autoplay abilities to automatically play and loop a sound file on page load:

```
<audio loop autoplay>  
  <source src="http://music.com/clip.mp3" />  
</audio>
```

Unfortunately, you may notice there is a small pause between loops. HTML5 audio, thou art half-baked still. Let's hope one day the **Web Audio API** will be able to help us out, but until things improve, we'll have to hack our way around these shortcomings.

Turns out there's a handy little script called **seamlessLoop.js** which we can use to patch this. Mind you, if we were really getting crazy with the Cheese Whiz, we'd want to get out **big guns** like **sound.js**. But that'd be overkill for a mere loop (and explaining the Web Audio API might bore, rather than impress your friends)!

Installing `seamlessLoop.js` will get rid of the pause, and now our walk cycle is complete.

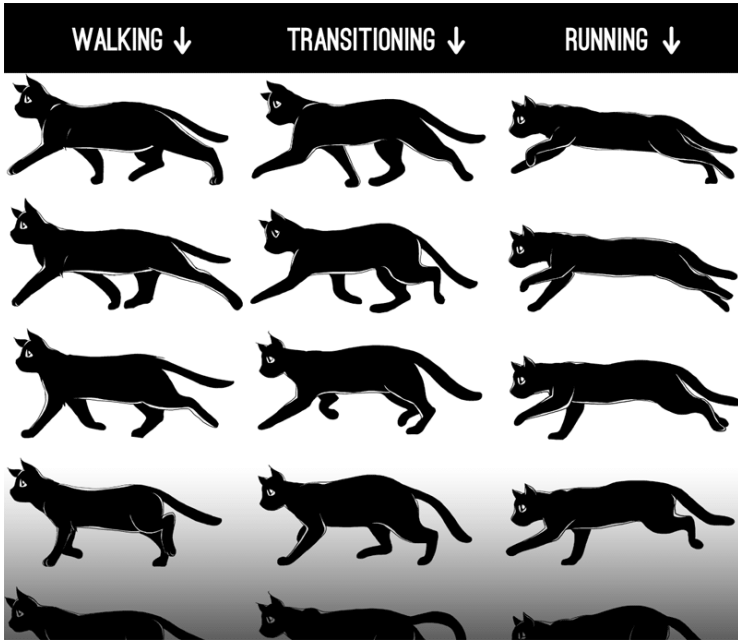
(I've done some very rough sniffing to see if the browser can play MP3 files. If not, we fall back to using .ogg formatted clips (Opera and Firefox users, you're welcome).)

## **REALLY IMPRESS YOUR FRIENDS BY ADDING A RUN CYCLE**

So we have music, we have a walk cycle, we have parallax. It will be a snap to bring them all together and have a simple, endless animation. But let's go one step further and knock the socks off our viewers by adding a run cycle.

### **The run cycle**

Tacking a run cycle on to our walk cycle will require a third animation sequence: a transitional animation of Tuna switching from walking to running. I have added all these to the sprite:



Let's work on getting that transition down. We're going to use multiple animations on the same `.tuna` div, but we're going to kick them off at different intervals using `animation-delay`—no JavaScript required! Isn't that magical?

It requires a wee bit of math (not much, it doesn't hurt) to line them up. We want to:

1. Loop the walk animation twice

2. Play the transitional cycle once (it has a finite beginning and end perfectly drawn to pick up between the last frame of the walk cycle and the first frame of the run cycle—no looping this baby)
3. RUN FOREVER.

Using the pattern animation: <name> <duration>  
<timing-function> <delay> <iteration-count>, here's what that looks like:

```
animation:
  walk-cycle 1s steps(12) 2,
  walk-to-run .75s steps(12) 2s 1,
  run-cycle .75s steps(13) 2.75s infinite;
```

I played with the times to get make the movement more realistic. You may notice that the running animation looks smoother than the walking animation. That's because it has 13 keyframes running over .75 second instead of 12 running in one second. Remember, professional animation studios use super-high frame counts. This little animation isn't even up to PBS's standards!

### **The music: extended play with HTML5 audio sprites**

My favorite part in the *The Girl in Byakkoya* is when the calm opening builds and transitions into a bouncy motif. I want to start with Tuna walking during the opening, and then loop the running and bounciness together for infinity.

1. The intro lasts for 24 seconds, so we set our 1 second walk cycle to run for 24 repetitions:  
`walk-cycle 1s steps(12) 24`
2. We delay `walk-to-run` by 24 seconds so it runs for .75 seconds before...
3. We play `run-cycle` at 24.75 seconds and loop it infinitely

For the music, we need to think of it as two parts: the intro and the bouncy loop. We can do this quite nicely with audio sprites: using one HTML5 audio element and using JavaScript to change the play head location, like skipping tracks with a CD player. Although this technique will result in a small gap in music shifts, I think it's worth using here to give you some ideas.

```
// Get the audio element
var byakkoya = document.querySelector('audio');
// create function to play and loop audio
function song(a){
    //start playing at 0
    a.currentTime = 0;
    a.play();
    //when we hit 64 seconds...
    setTimeout(function(){
        // skip back to 24.5 seconds and keep playing...
        a.currentTime = 24.55;
        // then loop back when we hit 64 again, or every
        // 59.5 seconds.
        setInterval(function(){
            a.currentTime = 24.55;
```



```

    }, 39450);
  }, 64000);
}

```

## The load screen

I've put it off as long as I can, but now that the music and the CSS are both running on their own separate clocks, it's imperative that both images and music be fully downloaded and ready to run when we kick this thing off. So we need a load screen (also, it's nice to give people a heads-up that you're about to blast them with music, no matter how wonderful that music may be).

Since the two timers are so closely linked, we'd best not run the animations until we run the music:

```
* { animation-play-state: paused; }
```

`animation-play-state` can be set to `paused` or `running`, and it's the most useful thing you will learn today.

First we use an event listener to see when the browser thinks we can play through from the beginning to end of the music without pause for buffering:

```
byakkoya.addEventListener("canplaythrough",
function () { });
```

(More on HTML5 audio's media events at [HTML5doctor.com](http://HTML5doctor.com))

Inside our event listener, I use a bit of jQuery to add class of `.playable` to the body when we're ready to enable the play button:

```
$( "body" ).addClass( "playable" );
    $( "#play-me" ).html( "Play me." ).click( function() {
        song( byakkoya );
        $( "body" ).addClass( "playing" );
    } );
```

That `.playing` class is special because it turns on the animations at the same time we start playing the song:

```
.playing * { animation-play-state: running; }
```

## The background

We're almost done here! When we add the background, it needs to speed up at the same time that Tuna starts running. The music picks up speed around 24.75 seconds in, and so we're going to use `animation-delay` on those backgrounds, too.

This will require some math. If you try to simply shorten the animation's duration at the 24.75s mark, the backgrounds will, mid-scroll, jump back to their initial background positions to start the new animation! Argh! So let's make a new `@keyframe` and calculate where the background position would be *just before* we speed up the animation.

Here's the formula:

**new 0% value = delay ÷ old duration × length of image**

**new 100% value = new 0% value + length of image**

Here's the formula put to work on a smaller scale:

### **Voilà! The finished animation!**

I've always wanted to bring my illustrations to life. Then I woke up one morning and realized that I had all the tools to do so in my browser and in my head. Now I have fallen in love with Flashless animation.

I'm sure there will be detractors who say HTML wasn't meant for this and it's a gross abuse of the DOM! But I say that these explorations help us expand what we expect from devices and software and challenge us in good ways as artists and programmers. The browser might not be the most appropriate place for animation, but is certainly a *fun* place to start.

There is so much you can do with the spec implemented today, and so much of the territory is still unexplored. I have not yet begun to show you everything. In eight months I expect this demo will represent the norm, not the bleeding edge. I look forward to seeing the wonderful things you create.

*(Also, someone, please, do something about that gappy HTML5 audio looping. It's a crying shame!)*

## ABOUT THE AUTHOR



**Rachel Nabors** is an interaction developer and award-winning cartoonist. She travels the world, speaking and training people in the art of web animation. When not biking around her home city of Portland, she makes interactive comics at her company **Tin Magpie**. You can catch her as **@rachelnabors** on Twitter and at **rachelnabors.com**.

## 7. Think First, Code Later

---

Stephen Fulljames

[24ways.org/201207](http://24ways.org/201207)

This is a story that's best told from the end, and it's probably one you're all familiar with.

You, or someone just like you, have been building a website, probably as part of a skilled and capable team. You're a front-end developer, focusing on JavaScript – it's either your sole responsibility or shared around. It's quite a big job, been going on for months, and at last it feels like you're reaching the end of it.

But, in a brief moment of downtime, you step back and take a look at the code as a whole. You notice that the folder called “jQuery plugins” suddenly looks rather full, and maybe there's evidence of several methods of doing the same thing; there are loads of little niggly fixes in the bug tracker; and every place you use Ajax the structure of the data is slightly different. You sigh, and your shoulders droop slightly, and you think “Yeah, we'll do that more cleanly next time.”

The thing is, you probably already know how to rewrite the start of this story to make the ending work better. This situation is not really anyone's fault – it's just an accumulation of all the things you decided along the way, all the things you agreed you'd fix later that have disappeared into the black hole of technical debt, and accomodating all the "can we just...?" requests from around the team and the client.

So, the solution to this is easy, right? More interminable planning meetings, more tightly controlled and documented specifications, less freedom to innovate, to try out new ideas and enjoy what you're doing.

Wait, that sounds even less fun than the old way.

## **MINIMUM VIABLE PLANNING**

Actually, planning and specifications are exactly what you need, but the way you go about them can make a real difference, both to the quality of your code, and the quality of your life as a developer. It can be as simple as being a little more thoughtful before starting on any new piece of functionality. Involve your whole team if possible, or at least those working on what you're doing. Canvass opinions and work out what the solution to the problem might look like first, rather than coding speculatively to find out.

There are easy ways you can get into this habit of putting the thought and design up front, and it doesn't have to mean spending more time on the project as a whole. It also doesn't have to result in reams of functional specifications. Instead, let the code itself form the specification.

As JavaScript applications become more complex, unit testing is becoming ever more important. So embrace it, whether you prefer **QUnit**, or **Mocha**, or any of the other JavaScript testing frameworks out there. The TDD (or test-driven development) pattern is all about writing the tests first and then writing functional code to pass those tests; or, if you prefer, code that meets the specification given by the tests.

Sounds like a hassle at first, but once you get into the rhythm of it you should find that the time spent writing tests up front is no greater, and often significantly less, than the time you would have spent fixing bugs afterwards.

If what you're working on requires an API between client and server (usually Ajax but this can apply to any method of sending or receiving data) then spend a bit of time with the back-end developer to design the data contracts, before either of you cut any code. Work out what the API endpoints are going to be, and what the data structure you'll get back from a certain endpoint looks like. A mock

JSON object documented on a wiki is enough and it can be atomic. Don't worry about planning the entire project at once, just plan enough to get on with your current tasks.

Definition in this way doesn't have to make your API immutable – change is still fine – but if you know roughly where you're heading, then not only can your team's efforts become more parallel, but you're far more likely to have an easier time making it all work. And again, you have a specification – the shape of the data – to write your JavaScript against.

Putting everything together, you end up with a logical flow of development, from the specification agreed with the client (your backlog), to the specification agreed with your team (the API contract design), to the specification agreed with your code (your unit tests). Hopefully, there will be ample clues in all of this to inform your front-end library choices, because by then you should have a better picture of what you're going to need.

## **WHAT THE FRAMEWORK?**

As a JavaScript developer predominantly, these are the choices I'm particularly interested in – how and why you use JavaScript libraries and frameworks, both what you expect from them and what you actually get.



If we look back at how web development, and specifically JavaScript development has progressed – from the earliest days of using lines and lines of Dreamweaver code-barf to make an image rollover effect, to today’s large frameworks that handle working with the DOM, Ajax communication and visual effects all in one hit – the purpose of it is clear: to smooth over the inconsistent bumps between browsers and give a solid, reliable, predictable base on which to put our desired functionality.

Understanding what we expect the language as a specification to do, and matching that to what we observe browsers actually doing, and then smoothing out the differences, is a big job. Since the language and the implementations are also changing as we go along, it also feels like a never-ending job. So make full use of this valuable effort. Use jQuery or YUI or anything else you’re comfortable with, but it still pays to think early on about what you need your library to do and what the best choice is to meet that need.

I’ve come in to projects as a fixer and found, to take a recent example, that jQuery UI was being used just to provide a date picker and a modal effect. That’s a lot of code weight to provide two fairly simple pieces of functionality that could easily be covered by smaller plugins. Which isn’t to say that jQuery UI itself is a bad choice, but I could see that it had been included late on

just to do those things, whereas a more considered approach would have been to put the library in early and use it more universally.

There are other choices, too. If you automatically throw in jQuery (or whatever your favourite main library is) to a small site with limited functionality, you might only touch a tiny fraction of its scope. In my own development I started looking at what I actually needed from a JavaScript library. For a simple project like **What the Framework?**, all jQuery needed to do was listen for `.ready()` and then perform some light DOM selection before handing over to a client-side MVC framework. So perhaps there was another way to go about this while still avoiding the cross-browser headaches.

## DELETING JQUERY

But the jQuery pattern is compelling and familiar. And once you're comfortable with something, it's a bit of an effort to force yourself out of that comfort zone and learn. But looking back at my whole career, I realised that I've relearned pretty much everything I do, probably several times, since I started out. So it's worth keeping in mind that learning and trying new things is how development has advanced to where it is now, and how it will keep advancing in the future.

In the end this led me to **Ender**, which is billed as an NPM-style package manager for the browser, letting you search for and manage small, loosely coupled modules and their dependencies, and compile them to one file with a common API.

For What the Framework I ended up with a set of DOM tools, **Underscore** and **Knockout**, all minified into 25kb of JavaScript. This compares really well with 32kb minified for jQuery on its own, and Ender's use of the dollar variable and the jQuery-like syntax in many modules makes switching over a low-friction experience.

On more complex projects, where you're really going to use all the features of something like jQuery, but want to minimise the loading of other dependencies when you don't need them, I've recently started looking at **Jam**. This uses the **RequireJS** pattern to compile commonly used code into a library file and then manage dependencies and bring in others on a per-page basis depending on how you need it. Again, it all comes down to thinking about what you need and using it only when you need it. And the configurability of tools like Ender or Jam allow you to be responsive to changing requirements as your project grows.

## THERE IS NO RIGHT ANSWER

That's not to say this way of working automatically makes things easier. It doesn't. On a large, long-running project or one where future functionality is unknown, it's still hard to predict and plan for everything – at least until crystal balls as a service come about. But by including strong engineering practices in your front-end, and trying to minimise technical debt, you're at least giving yourself a decent safety net to guard against the “can we just...?” tendencies that are a fact of life.

So, really, this is not an advocacy of using a particular technology or framework, because I can't tell you what works for you or your team. But what I can tell you is that working this way round has done wonders for my productivity and enthusiasm, both for code quality and for trying out new libraries. Give it a go, you might like it!

## ABOUT THE AUTHOR



**Stephen Fulljames** is a freelance interface developer working mostly (these days) in Javascript and Node. Having started out writing about video games in the late 1990s, he was nominated to build the magazine's first website because "you know a bit about HTML" and it's kind of grown from there. Stephen lives near Brighton with his wife, Nat, and two kids and fills his spare time making atoms for railway modellers at **Narrow Planet**. You can find him on Twitter [@fulljames](#).

Photo: Nathalie Fulljames

# 8. Giving CSS Animations and Transitions Their Place

---

Val Head

24ways.org/201208

CSS animations and transitions may not sit squarely in the realm of the behaviour layer, but they're stepping up into this area that used to be pure JavaScript territory. Heck, CSS might even perform better than its JavaScript equivalents in some cases. That's pretty serious! With CSS's new tricks blurring the lines between presentation and behaviour, it can start to feel bloated and messy in our CSS files. It's an uncomfortable feeling.

Here are a pair of methods I've found to be pretty helpful in keeping the potential bloat and wire-crossing under control when CSS has its hands in both presentation and behaviour.

## **SAME EGGS, MORE BASKETS**

Structuring your CSS to have separate files for layout, typography, grids, and so on is a fairly common approach these days. But which one do you put your transitions and animations in? The initial answer, as always, is “it depends”.

Small effects here and there will likely sit just fine with your other styles. When you move into more involved effects that require multiple animations and some logic support from JavaScript, it’s probably time to choose none of the above, and create a separate CSS file just for them.

Putting all your animations in one file is a huge help for code organization. Even if you opt for a name less literal than `animations.css`, you’ll know exactly where to go for anything CSS animation related. That saves time and effort when it comes to editing and maintenance. Keeping track of which animations are still currently used is easier when they’re all grouped together as well. And as an added bonus, you won’t have to look at all those horribly unattractive and repetitive prefixed `@-keyframe` rules unless you actually need to.

An `animations.css` file might look something like the snippet below. It defines each animation’s keyframes and defines a class for each variation of that animation you’ll be using. Depending on the situation, you may also want

to include transitions here in a similar way. (I've found defining transitions as their own class, or mixin, to be a huge help in past projects for me.)

```
// defining the animation
@keyframes catFall {
  from { background-position: center 0;}
  to {background-position: center 1000px;}
}
@-webkit-keyframes catFall {
  from { background-position: center 0;}
  to {background-position: center 1000px;}
}
@-moz-keyframes catFall {
  from { background-position: center 0;}
  to {background-position: center 1000px;}
}
@-ms-keyframes catFall {
  from { background-position: center 0;}
  to {background-position: center 1000px;}
}

...

// class that assigns the animation

.catsBackground {
  height: 100%;
  background: transparent url(../endlessKittens.png) 0 0
  repeat-y;
  animation: catFall 1s linear infinite;
  -webkit-animation: catFall 1s linear infinite;
```



```
-moz-animation: catFall 1s linear infinite;  
-ms-animation: catFall 1s linear infinite;  
}
```

### IF WE DON'T NEED IT, WHY LOAD IT?

Having all those CSS animations and transitions in one file gives us the added flexibility to load them only when we want to. Loading a whole lot of things that will never be used might seem like a bit of a waste.

While CSS has us impressed with its motion chops, it falls flat when it comes to the logic and fine-grained control. JavaScript, on the other hand, is pretty good at both those things. Chances are the content of your `animations.css` file isn't acting alone. You'll likely be adding and removing classes via JavaScript to manage your CSS animations at the very least. If your CSS animations are so entwined with JavaScript, why not let them hang out with the rest of the behaviour layer and only come out to play when JavaScript is supported?

Dynamically linking your `animations.css` file like this means it will be completely ignored if JavaScript is off or not supported. No JavaScript? No additional behaviour, not even the parts handled by CSS.

```
<script>  
document.write('<link rel="stylesheet" type="text/css"  
href="animations.css">');  
</script>
```

This technique comes up in **progressive enhancement techniques** as well, but it can help here to keep your presentation and behaviour nicely separated when more than one language is involved. The aim in both cases is to avoid loading files we won't be using.

If you happen to be doing something a bit fancier – like 3-D transforms or critical animations that require more nuanced fallbacks – you might need something like **modernizr** to step in to determine support more specifically. But the general idea is the same.

## **SUMMING IT ALL UP**

Using a couple of simple techniques like these, we get to pick where to best draw the line between behaviour and presentation based on the situation at hand, not just on what language we're using. The power of when to separate and how to reassemble the individual pieces can be even greater if you use preprocessors as part of your process. We've got a lot of options! The important part is to make forward-thinking choices to save your future self, and even your current self, unnecessary headaches.

## ABOUT THE AUTHOR



**Val Head** is totally into design, type and code. She is a designer currently based in Pittsburgh where she works with agencies and small businesses to keep the web fun. She speaks internationally at conferences and leads workshops on web design and creative coding.

Every year she and **Jason** bring a swarm of web designers to Pittsburgh for **Web Design Day**. She also runs the local creative coding meet up, Loop and helps keep **Refresh Pittsburgh** going strong. She likes people. Val **tweets too much**, occasionally **dribbles**, and **blogs** somewhat inconsistently.

# 9. Should We Be Reactive?

---

Dan Donald

[24ways.org/201209](http://24ways.org/201209)

## EVOLUTION

Looking at the evolution of the web and the devices we use should help remind us that the times we're adjusting to are just another step on a journey. These times seem to be telling us that we need to embrace flexibility.

Imagine an HTML file containing nothing but text. It's viewable on any web-capable device and reasonably readable: the notion of the universality of the web was very much a founding principle. Right from the beginning, browser vendors understood that we'd want text to reflow (why wouldn't we?), so I consider the first websites to have been fluid.

As we attempted to exert more control through our designs in the early days of the web, debates about whether we should produce fixed or fluid sites raged. We could create fluid designs using tables, but what we didn't have then was a wide range of web capable devices or the

ability to control this fluidity. The biggest changes occurred when stats showed enough people using a different screen resolution we could cater for.

To me, the techniques of responsive web design provide the control we were missing. Combining new approaches to layout and images with media queries empowered us to learn how to embrace the inherent flexibility of the web in ways to suit our work and the devices used by our audience.

Perhaps another kind of flexibility might be found in how we use context to affect how we present our content; to consider how we might use the information we can access from people, browsers and devices to provide web experiences – effectively creating sites that react to initial or changing circumstances in the relationship between people and our content.

## **EMBRACING FLEXIBILITY**

So what is context? Put simply, you could think of it as a secondary piece of information that helps clarify the meaning of the first. It helps set a scene or describe circumstances. I think that Cennydd Bowles has summed it up really well through talks he's given recently, in which he's arrived at the acronym DETAILS (**D**evice, **E**nvironment, **T**ime, **A**ctivity, **I**ndividual, **L**ocation, **S**ocial) – I encourage you to keep an eye out for his next book due

in the new year where he'll explore this idea much further. This clarity over what context could mean in terms of what we do on the web is fundamental, directing us towards ways we might use it.

When you stop to think about it, we've been using some basic pieces of this information right from the beginning, like bits of JavaScript or Java applets that serve an appropriate greeting to your site's visitors, or show their location, or even local weather. But what if we think of this from the beginning of our projects?

We should think about our content first. Once we know this and have a direction, perhaps then we can think about what context, or even multiple contexts, might help us to communicate more effectively.

## **THE REAL WORLD**

There's always been a disconnect between the real world and the web, which is to be expected. But the world around us is a sea of data; every fundamental building block: people, places, events and things have information waiting to be explored.

For sites based around physical objects or locations, this divide is really apparent. We don't ordinarily take the time to describe in code the properties of a place, or consider

whether your relationship to the place in the real world should have any impact on your relationship with a site about it.

When I think about local businesses, they have such rich properties to draw on and yet we don't really explore them in any meaningful way, even through something as simple as opening hours.

### **NOW WE HAVE DATA...**

We've long had access to the current time both on server- and client-sides. The use of geolocation is easier than ever, but when we look at the range of information we could glean to help us make some choices, maybe there's some help on the horizon from projects like the **W3C Device APIs Working Group**. This might prove useful to help make us aware of network and battery conditions of a device, along with the potential to gain data from other sensors, which could tell us about lighting conditions, ambient noise levels and temperature depending on the capabilities of the device.

It may be that our sites have some form of login or access to your profile from another site. Along with data from our devices and browsers, this should give us a sense of how best to talk to our audience in certain situations. We don't necessarily need to know any personal details, just enough to make decisions about how to present our sites.

## THE REACTIVE WEB?

So why reactive web design? I'm hoping that a name might help us to have a common vocabulary not only about what we mean when we talk about context, but how it could be considered through our projects, right from the early stages. How could this manifest itself?

A simple example might be a location-aware panel on your site. Perhaps the space is a little down in your content hierarchy but serves a perfectly valid purpose by default. To visitors outside the country perhaps this works fine, but within your country maybe this panel could be used to communicate more effectively. Further still, if we knew the visitors were in the vicinity, we could talk to them more directly.

What if both time and location were relevant? This space could work as before but you could consider how time could intersect with your local audience. If you know they're local and it's a certain time of day, you could communicate directly with them.

This example isn't beyond what banner ads often do and uses easily accessible information. There are more unusual combinations we may be able to find, such as movement and presence. Perhaps a site that tells a story, which changes design and content based on whether you're moving, how long you've been on the site and how far you've travelled. This isn't what we typically expect



from websites, but we should bear in mind that what websites are now will not be what they become in the future.

You could do much of this contextual presentation through native apps, of course. **The Silent History**, an app novel written and designed for iPad and iPhone, uses an exploration element, providing “hundreds of location-based stories across the U.S. and around the world. These can be read only when your device’s GPS matches the coordinates of the specified location.” But considering the universality of the web, we could redefine what web-based experiences should be like. Not all methods would work well on the web, but that’s a decision that has to be made for a specific project.

By thinking more broadly about any web-capable device, we can use what we know to provide relevant experiences for our site’s visitors. We need to be sure what we mean by relevant, of course!

## **REALITY BITES**

While there are incredible possibilities, from a simple panel on a site to something bordering on living sites that evolve and change with our circumstances, we need to act with a degree of pragmatism and understand how much of what we could do is based on assumptions and the bias of our own experiences.

We could go wild with changing the way our content is presented based on contextual information, but if we're not careful what we end up with confuses and could provide a very fractured experience. As much as possible we need to think more ethnographically, observe and question people in the situations we think may be relevant, and test our assumptions as early as we can. Even on small projects, there may be ways we can validate our assumptions and test with our audience. The key to applying contextual content or cues is not to break the experience between contextual views (as I think we now wouldn't when hiding content on a mobile view).

It's another instance of progressive enhancement – as we know certain pieces of information, we can enhance the experience. Also, if you do change content, how can you not make a more cumbersome experience for your visitors?

## **IT'S ALL ABOUT COMMUNICATION**

Content is at the core of what we do, but if we consider context we need to understand the impact on that. The effect could be as subtle as an altered hierarchy, involve swapping out panels of content, or in extreme instances perhaps all of your content might change. In some ways, this extends the notion of **adaptive content** that Karen McGrane has been talking about, to how we write and store the content we create. Thinking about the the

impact of context may require us to re-evaluate our site structure, too. Whatever we decide, we have to be clear what will happen and manage the expectations of our users.

## THE BOTTOM LINE

What I'm proposing isn't that we go crazy and end up with a confused, disjointed set of experiences across the web. What I hope is that starting right from the beginning of a project, we think about what context is and could be, and see what relevance it might have to what we're trying to communicate. This strategic process leads us to think about design.

We are slowly adapting to what it means to be flexible through responsive and adaptive processes. What does thinking about contextual states mean to us (or designing for state in general)? Does this highlight again how difficult it'll be for our tools to keep up with our processes and output?

In terms of code, the vast majority of this data comes from the client-side through JavaScript. While we can progressively enhance, this could lead to a lot of code bloat through feature or capability detection, and potentially a lot of conditional loading of scripts. It's a real shame we don't get much we can rely on from the server-side – we know how unreliable user agents are!

We need to understand why we'd do this. Are we trying to communicate well and be useful, or doing it to show off? Underneath it all, what do we base our decisions on? Do we have actual insight or are we proceeding from our assumptions and the bias of our own experiences? **Scott Jensen** summed it up best for me: (to paraphrase) the pain we put people through has to be greatly outweighed by the value we offer.

I see that this could be another potential step in our evolution on the web; continuing this exploration of the flexibility the web allows us. It's amazing we can do such incredible things from what is essentially a set of disparate, linked documents.

## ABOUT THE AUTHOR



**Dan Donald**, based up near Manchester, currently tinkers with web things at BBC Sport, tries his hand from time to time at speaking, and tweets nonsense as [@hereinthehive](#). He hopes to finally get big-picture-web-journal-thing **Break the Page** launched at some point before the end of the world. When not webbing it up he makes noise in **Mark of 1000 Evils** and stacks up side projects he'll never get to.

# 10. Fluent Design through Early Prototyping

---

Rebecca Cottrell

[24ways.org/201210](http://24ways.org/201210)

There's a small problem with wireframes. They're not good for showing the kind of interactions we now take for granted – transitions and animations on the web, in Android, iOS, and other platforms. There's a belief that early prototyping requires a large amount of time and effort, and isn't worth an early investment. But it's not true!

It's still normal to spend a significant proportion of time working in wireframes. Given that wireframes are high-level and don't show much detail, it's tempting to give up control and responsibility for things like transitions and other things sidelined as visual considerations. These things aren't expressed well, and perhaps not expressed at all, in wireframes, yet they critically influence the quality of a product. Rapid prototyping early helps to bring sidelined but significant design considerations into focus.

## **Speaking fluent design**

Fluency in a language means being able to speak it confidently and accurately. The Latin root means *flow*.

By design fluency, I mean using a set of skills in order to express or communicate an idea. Prototyping is a kind of fluency. It takes designers beyond the domain of grey and white boxes to consider all the elements that make up really good product design.

Designers shouldn't be afraid of speaking fluent design. They should think thoroughly about product decisions beyond their immediate role — not for the sake of becoming some kind of power-hungry design demigod, but because it will lead to better, more carefully considered product design.

## **Wireframes are incomplete sentences**

Wireframes, once they've served their purpose, are a kind of self-imposed restriction.

Mostly made out of grey and white boxes, they deliberately express the minimum. Important details — visuals, nuanced transitions, sounds — are missing. Their appearance bears little resemblance to the final thing. Responsibility for things that traditionally didn't matter (or exist) is relinquished. Animations and transitions in particular are increasingly relevant to the mobile

designer's methods. And rather than being fanciful and superfluous visual additions to a product, they help to clarify designs and provide information about context.

Wireframes are useful in the early stages. As a designer trying to persuade stakeholders, clients, or peers, sometimes it will be in your interests to only tell half the story. They're ideal for gauging whether a design is taking the right direction, and they're the right medium for deciding core things, such as the overall structure and information architecture.

But spending a long time in wireframes means delaying details to a later stage in the project, or to the end, when the priority is shifted to getting designs out of the door. This leaves little time to test, finesse and perfect things which initially seemed to be less important. I think designers should move away from using wireframes as primary documentation once the design has reached a certain level of maturity.

### **A prototype is multiple complete sentences**

Paragraphs, even.

Unlike a wireframe, a prototype is a persuasive storyteller. It can reveal the depth and range of design decisions, not just the layout, but also motion: animations and transitions. If it's a super-high-fidelity prototype, it's a perfect vessel for showing the visual design as well. It's all



of these things that contribute to the impression that a product is good... and useful, and engaging, and something you'd like to use.

A prototype is *impressive*. A good prototype can help to convince stakeholders and persuade clients. With a compelling demo, people can more easily imagine that this thing could actually exist. "Hey", they're thinking. "This might actually be pretty good!"

### **How to make a prototype in no time and with no effort**

Now, it does take time and effort to make a prototype. However, good news! It used to require a lot *more* effort. There are tools that make prototyping much quicker and easier.

If you're making a mobile prototype (this seems quite likely), you will want to test and show this on the actual device. This sounds like it could be a pain, but there are a few ways to do this that are quite easy.

Keynote, Apple's presentation software, is an unlikely candidate for a prototyping tool, but surprisingly great and easy for creating prototypes with transitions that can be shown on different devices.

Keynote enables you to do a few useful, excellent things. You can make each screen in your design a slide, which can be linked together to allow you to click through the

prototype. You can add customisable transitions between screens. If you want to show a panel that can slide open or closed on your iPad mockup, for example, transitions can also be added to individual elements on the screen. The design can be shown on tablet and mobile devices, and interacted with like it's a real app. Another cool feature is that you can export the prototype as a video, which works as another effective format for demoing a design.

Overall, Keynote offers a very quick, lightweight way to prototype a design. Once you've learned the basics, it shouldn't take longer than a few hours – at most – to put together a respectable clickable prototype with transitions.

### Download the interactive MOV example

Holly icon by Megan Sheehan from The Noun Project

This is a Quicktime movie exported from Keynote. This version is animated for demonstration purposes, but **download the interactive original** and you can click the screen to move through the prototype. It demonstrates the basic interactivity of an iPhone app. This anonymised example was used on a project at **Fjord** to create a master example of an app's transitions.

## **Prototyping drawbacks, and perceived drawbacks**

If prototyping is so great, then why do we leave it to the end, or not bother with it at all? There are multiple misconceptions about prototyping: they're too difficult to make; they take too much time; or they're inaccurate (and dangerous) documentation.

A prototype is a preliminary model. There should always be a disclaimer that it's not the real thing to avoid setting up false expectations.

A prototype doesn't have to be the main deliverable. It can be a key one that's supported by visual and interaction specifications. And a prototype is a lightweight means of managing and reflecting changes and requirements in a project.

An actual drawback of prototyping is that to make one too early could mean being gung-ho with what you thought a client or stakeholder wanted, and delivering something inappropriate. To avoid this, communicate, iterate, and keep things simple until you're confident that the client or other stakeholders are happy with your chosen direction.

The key throughout any design project is iteration. Designers build iterative models, starting simple and becoming increasingly sophisticated. It's a process of iterative craft and evolution. There's no perfect methodology, no magic recipe to follow.

## What to do next

Make a prototype! It's the perfect way to impress your friends.

It can help to advance a brilliant idea with a fraction of the effort of complete development. Sketches and wireframes are perfect early on in a project, but once they've served their purpose, prototypes enable the design to advance, and push thinking towards clarifying other important details including transitions.

For Keynote tutorials, **Keynotopia** is a great resource. **Axure** is standard and popular prototyping software many UX designers will already be familiar with; it's possible to create transitions in Axure. **POP** is an iPhone app that allows you to design apps on paper, take photos with your phone, and turn them into interactive prototypes. **Ratchet** is an elegant iPhone prototyping tool aimed at web developers.

There are perhaps hundreds of different prototyping tools and methods. My final advice is not to get bogged down in (or limited by) any particular tool, but to remember you're making quick and iterative models. Experiment and play!

Prototyping will push you and your designs to a scary place without limitations. No more grey and white boxes, just possibilities!

## ABOUT THE AUTHOR



**Rebecca Cottrell** is an interaction designer at Fjord. She likes prototyping, living in London (most of the time), and unusual-looking animals.

# 11. Responsive Images: What We Thought We Needed

---

Paul Lloyd

24ways.org/201211

If you were to read a web designer's Christmas wish list, it would likely include a solution for displaying images responsively. For those concerned about users downloading unnecessary image data, or serving images that look blurry on high resolution displays, finding a solution has become a frustrating quest.

Having experimented with **complex** and sometimes **devilish** hacks, consensus is forming around defining new standards that could solve this problem. Two approaches have emerged.

The `<picture>` **element** markup pattern was **proposed** by **Mat Marquis** and is now being developed by the **Responsive Images Community Group**. By providing a means of declaring multiple sources, authors could use media queries to control which version of an image is displayed and under what conditions:

## Responsive Images: What We Thought We Needed

```
<picture width="500" height="500">  
  <source media="(min-width: 45em)" src="large.jpg">  
  <source media="(min-width: 18em)" src="med.jpg">  
  <source src="small.jpg">  
    
  <p>Accessible text</p>  
</picture>
```

A second proposal put forward by Apple, the **srcset attribute**, uses a more concise syntax intended for use with the `<img>` element, although it could be compatible with the `<picture>` element too. This would allow authors to provide a set of images, but with the decision on which to use left to the browser:

```

```

## ENTER SCROOGE

Men's courses will foreshadow certain ends, to which, if persevered in, they must lead.

*Ebenezer Scrooge*

Given the complexity of this issue, there's a heated debate about which is the best option. Yet code belies a certain truth. That both feature verbose and opaque syntax, I'm not sure either should find its way into the browser – especially as alternative approaches have yet to be fully explored.

So, as if to dampen the festive cheer, here are five reasons why I believe both proposals are largely redundant.

## **1. WE NEED BETTER FORMATS, NOT MORE MARKUP**

As we move away from designs defined with fixed pixel values, bitmap images look increasingly unsuitable. While simple images and iconography can use scalable vector formats like SVG, for detailed photographic imagery, raster formats like GIF, PNG and JPEG remain the only suitable option.

There is scope within current formats **to account for varying bandwidth** but this requires cooperation from browser vendors. Newer formats like JPEG2000 and WebP generate higher quality images with smaller file sizes, but aren't widely supported.

While it's tempting to try to solve this issue by inventing new markup, the crux of it remains at the file level.

Daan Jobsis's experimentation with image compression strengthens this argument. He discovered that by increasing the dimensions of a JPEG image while simultaneously reducing its quality, a smaller files could be produced, with the resulting image looking just as good on both standard and high-resolution displays.



This may be a hack in lieu of a more permanent solution, but it's applied in the right place. Easy to accomplish with existing tools and without compatibility issues, it has few downsides. Further experimentation in this area should be encouraged, with standardisation efforts more helpful if focused on developing new image formats or, preferably, extending existing ones.

## 2. ART DIRECTION DOESN'T BELONG IN MARKUP

A desired benefit of the `<picture>` markup pattern is to allow for **greater art direction**. For example, rather than scaling down images on smaller displays to the point that their content is hard to discern, we could present closer crops instead:



Original Image



Scaled down



Cropped

This can be achieved with CSS of course, although with a download penalty for those parts of an image not shown. This point may be negligible, however, since in the context of adaptable layouts, these hidden areas may end up being revealed anyway.

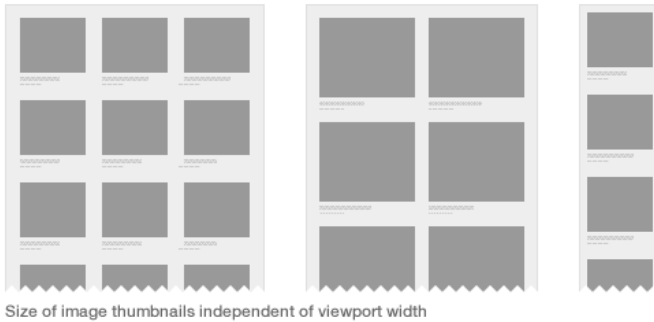
Art direction concerns design, not content. If we wish to maintain a separation of concerns, including presentation within our markup seems misguided.

### **3. THE SIZE OF A DISPLAY HAS LITTLE RELATION TO THE SIZE OF AN IMAGE**

By using media queries, the `<picture>` element allows authors to choose which characteristics of the screen or viewport to query for different images to be displayed.

In developing sites at **Clearleft**, we have noticed that the viewport is essentially arbitrary, with the size of an image's containing element more important. For example, look at how this grid of images may adapt at different viewport widths:

## Responsive Images: What We Thought We Needed



As we build more modular systems, components need to be adaptable in and of themselves. There is a case to be made for developing more contextual methods of querying, rather than those based on attributes of the display.

### 4. WE HAVEN'T LIVED WITH THE PROBLEM LONG ENOUGH

A key strength of the web is that the underlying platform can be continually iterated. This can also be problematic if snap judgements are made about what constitutes an improvement.

The early history of the web is littered with such examples, be it the perceived need for blinking text or inline typographic styling. To build a platform for the future, additions to it should be carefully considered. And

if we want more consistent support across browsers, burdening vendors with an ever increasing list of features seems counterproductive.

Only once the need for a new feature is sufficiently proven, should we look to standardise it. Before we could declare hover effects, rounded corners and typographic styling in CSS, we used JavaScript as a polyfill. Sure, doing so was painful, but use cases were fully explored, and the CSS specification better reflected the needs of authors.

## 5. IMAGES AND THE WEB AESTHETIC

The srcset proposal has emerged from a company that markets its phones as being able to browse the real – yet squashed down, tapped and zoomable – web. Perhaps Apple should make its own website responsive before suggesting how the rest of us should do so.

Converserly, while the <picture> proposal has the backing of a few respected developers and designers, it was born out of the work Mat Marquis and Filament Group did for the **Boston Globe**. As the first large-scale responsive design, this was a landmark project that ignited the responsive web design movement and proved its worth. But it was the first.

Its design shares a vernacular to that of contemporary newspaper websites, with a columnar, image-laden and densely packed layout. Compared to more recent

## Responsive Images: What We Thought We Needed

examples – Quartz, The Next Web and the New York Times Skimmer – it feels out of step with the future direction of news sites. In seeking out a truer aesthetic for the web in which software interfaces have greater influence, we might discover that the need for responsive images isn't as great as originally thought.



NYTimes website



NYTimes webapp

## BUILDING FOR THE FUTURE

With responsive design, we've accepted the idea that a fully fluid layout, rather than a set of fixed layouts, is best suited to the web's unpredictable nature. Current responsive image proposals are antithetical to this approach.

We need solutions that lack complexity, are device-agnostic and work within existing workflows. Any proposal that requires different versions of the same image to be created, is likely to have to acquiesce under the pressure of reality.

While it's easy to get distracted about the size and quality of an image, and how we might choose to serve it, often the simplest solution is not to include it at all. After years of gluttonous design practice, in which fast connections and expansive display sizes were an accepted norm, we have got use to filling pages with needless images and countless items of page furniture.

To design more adaptable experiences, the presence of every element needs to be questioned, for its existence requires additional data to be downloaded or further complexity within a design system. Conditional loading techniques mean that the inclusion of images is no longer a binary choice, but can instead appear in a progressively enhanced manner.

So here is my proposal. Instead of spending the next year worrying about responsive images, let's embrace the constraints of the medium, and seek out new solutions that can work within them.

## 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**.

# 12. Design Systems

---

Laura Kalbag

24ways.org/201212

The most important part of responsive web design is that, no matter what the viewport width, the content is accessible in an optimum display. The *best* responsive designs are those that allow you to go from one optimised display to another, but with the feeling that these experiences are part of a greater product whole.

## **RESPONSIVE DESIGN: WHERE WE'VE BEEN GOING WRONG**

Responsive web design was a shock to my web designer system. Those of us who had already been designing sites for mobile probably had the biggest leap to make. We might have been detecting user agents in order to deliver a mobile-specific site, or using the slightly more familiar **Bushido** technique to deliver sites optimised for device type and viewport size, but either way our focus was on devices. A site was optimised for either a mobile phone or a desktop.



Responsive web design brought us back to pre-table layout fluid sites that expanded or contracted to fit the viewport. This was a big difference to get our heads around when we were so used to designing for fixed-width layouts. Suddenly, an element could be any width or, at least, we needed to consider its maximum and minimum widths. Pixel perfection, while pretty, became wholly unrealistic, and a whole load of designers who prided themselves in detailed and precise designs got a bit scared.

Hanging on to our previous processes and typical deliverables led us to continue to optimise our sites for particular devices and provide pixel-perfect mockups for those device widths.

With all this we were concentrating on devices, not content, deliverables and not process, and making assumptions about users and their devices based on nothing but the width of the viewport.

I don't think this is a crime, I think it was inevitable.

We can be up to date with our principles and ideals, but it's never as easy in practice. That's why it's more important than ever to share our successful techniques and processes. Let's drag each other into modern web design.

## DESIGN SYSTEMS: THE PRINCIPLES

### What are design systems?

A visual design system is built out of the core components of typography, layout, shape or form, and colour. When considering the design of a whole product, a design system should also include patterns in user flow, content strategy, copy, and tone of voice. These concepts, design decisions or rules, created around the core components are used consistently across your product to create a cohesive feel, whether it's from one element to another, page to page, or viewport width to viewport width.

Responsive design is one of the most important considerations in the components of a design system. For each component, you must decide what will unite the design across the viewports to maintain that consistent feel, and what parts of the design will differentiate in order to provide a flexible and optimal experience for different viewport sizes.

### COMPONENTS YOU MIGHT KEEP THE SAME ACROSS VIEWPORTS

- typeface
- base unit
- colour
- shape/form

## COMPONENTS YOU MIGHT DIFFERENTIATE ACROSS VIEWPORTS

- grids
- layout
- font size
- measure (line length)
- leading (line height)

## CONTENT: IT MUST ALWAYS BE THE SAME

The focus of a design system is the optimum display of content. As Mark Boulton put it, designing “content out, not canvas in.” Chris Armstrong puts the emphasis on **not designing for viewports but for content** – “we need to build on what we *do* know: content.” In order to do this, we must share the same content across all devices and focus on how best to display and represent content through design system components.

## THE PRACTICAL: CORE VISUAL COMPONENTS

### Typography first

When you work with a lot of text content, typography is the easiest way to set the visual tone of the design across all viewport widths. It’s likely that you’ll choose one or two typefaces to use across the whole system, but you

might change the most legible font size, balanced with the most comfortable measure, as the viewport width changes.

## **Where typography meets layout**

The unit on which you choose to base the grid and layout design, font sizes and leading could be based on the typeface, an optimal reading size, or something more arbitrary. Sometimes I'll choose a unit based on multiples of ten because it makes the maths in the CSS easier. Tim Brown suggests trying a **modular scale**. Chris Armstrong suggests basing it on **your ideal measure, or the width of a fixed item of content** such as an ad unit.

## **Grids and layouts**

Sensible grid design can be a flexible yet solid foundation for your design system layout component. But you must be wary in responsive design that a grid might not work across all widths: even four columns could make for very cramped content and one-word measures on smaller screens.

Maybe the grid columns are something you differentiate across widths, but you can keep the concept of the grid consistent. If the content has blocks in groups of three, you might decide on a three-column grid which folds down to one column for narrow viewports. If the grid

focuses on the idea of symmetry and has a four-column grid on larger viewports, it might fold down to two columns for narrower viewports. These consistencies may seem subtle, not at all obvious to many except the designer, but it's all these little constants and patterns across the whole of the design system that makes design decisions easier to make (as they adhere to the guiding concepts of your system), and give the product a uniform feel no matter what the device.

### **Shape or form**

The shape or form components are concepts you already use in fixed-width web design for a strong, consistent look and feel.

Since CSS `border-radius` became widely supported by browsers, a lot of designs feature circle themes. These are very distinctive and can be used across viewport widths giving them the same united feel, even if they're not used in the same way. This could also apply to border styles, consistent shadows and any number of decorative details and textures. These are the elements that make up the shape or form of a design system.

## Colour

Colour is the most basic way to reinforce a brand and unite experiences across viewports. The same hex colour used system-wide is instantly recognisable, no matter what the viewport width.

## THE PROCESS

While using a design system isn't necessarily attached to any particular process, it does lend itself to some process ideals.

### **Detaching design considerations from viewport widths**

A design system allows you to focus separately on the components that make up the system, disconnecting the look and feel from the layout. This helps prevent us getting stuck in the rut of the Apple breakpoints (brilliantly coined by **Simon Foster**) of mobile, tablet and desktop. It also forces us to design for variation in viewport experiences side by side, not one after the other.

### **Design in the browser**

I can't start off designing in the browser – it just doesn't seem to bring out my creative side (and I'm incredibly envious of you if you can; I just *have* to start on paper) – but static mock-ups aren't the only alternative. **Style**

**guides** and **style tiles** are perfect for expressing the concepts of your design system. **Pattern libraries** could also work well.

### **Mock-ups and breakpoints**

At some point, whether it's to test your system ideas, or because a client needs help visualising how your system might work, you may end up producing some static mock-ups. It's not the end of the world, but you must ensure that these consider all the viewports, not just those of the iDevices, or even the devices currently on the market. You need to decide the breakpoints where the **states of your design** change. The blocks within your content will always have optimum points for their display (based on their hierarchy, density, width, or type of interaction) and so your breakpoints should be based around these points.

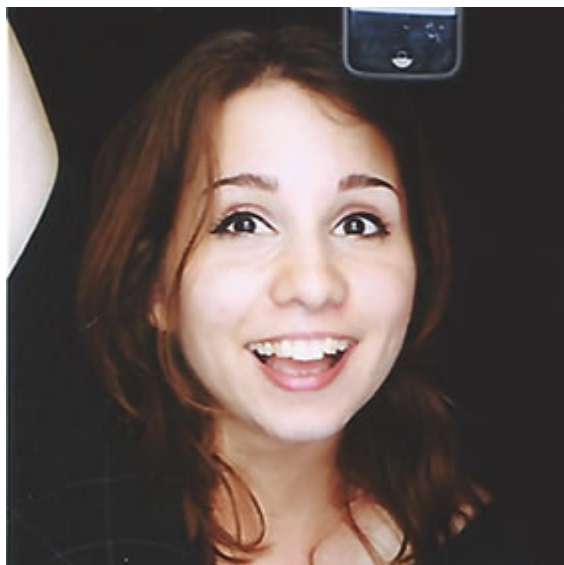
These are probably the ideal points at which to produce static mockups; treat them as snapshots. They're not necessarily mock-ups, so much as a way of capturing how your design system would be interpreted when frozen at that particular viewport width.

## **THE FUTURE**

Creating design systems will give us the flexibility we need for working with the unknown devices of the future. It may be a change in process, but it shouldn't be too much

of a difference in thinking. The pioneers in responsive design have a hard job. Some of these problems may have already been solved in other technologies or industries, but it's up to the pioneers to find those connections and help us formulate solutions and standards that will make responsive design the best it can possibly be. We need to keep experimenting and communicating, particularly in the area of design, as good user experiences are the true sign of whether our products are a success.

## **ABOUT THE AUTHOR**





**Laura Kalbag** is a designer easily excited by web design and development. Among her list of ever-changing pet subjects are responsive web, semantic web, and web fonts, but she's really fascinated by anything in the areas of web, mobile and design.

Laura has been a **freelancer** for the whole of her professional life. She revels in working with small and meaningful clients, creating websites, apps, icons, illustrations and the odd logo.

# 13. Redesigning the Media Query

---

Les James

[24ways.org/201213](http://24ways.org/201213)

Responsive web design is showing us that designing content is more important than designing containers. But if you've given RWD a serious try, you know that shifting your focus from the container is surprisingly hard to do. There are many factors and instincts working against you, and one culprit is a perpetrator you'd least suspect.

The media query is the ringmaster of responsive design. It lets us establish the rules of the game and gives us what we need most: control. However, like some kind of evil double agent, the media query is actually working against you.

Its very nature diverts your attention away from content and forces you to focus on the container.

The very act of choosing a media query value means choosing a screen size.

Look at the history of the media query—it's always been about the container. Values like `screen`, `print`, `handheld` and `tv` don't have anything to do with content. The modern media query lets us choose screen dimensions, which is great because it makes RWD possible. But it's still the act of choosing something that is completely unpredictable.

*Content* should dictate our breakpoints, not the container. In order to get our focus back to the only thing that matters, we need a reengineered media query—one that frees us from thinking about screen dimensions. A media query that works for your content, not the window. Fortunately, **Sass 3.2** is ready and willing to take on this challenge.

## THINKING IN COLUMNS

Fluid grids never clicked for me. I feel so disoriented and confused by their squishiness. Responsive design demands their use though, right?

I was ready to surrender until I found a grid that turned my world upright again. The **Frameless Grid** by Joni Korpi demonstrates that column and gutter sizes can stay fixed. As the screen size changes, you simply add or remove columns to accommodate. This made sense to me and

armed with this concept I was able to give Sass the first component it needs to rewrite the media query: fixed column and gutter size variables.

```
$grid-column: 60px;  
$grid-gutter: 20px;
```

We're going to want some resolution independence too, so let's create a function that converts those nasty pixel values into ems.

```
@function em($px, $base: $base-font-size) {  
  @return ($px / $base) * 1em;  
}
```

We now have the components needed to figure out the width of multiple columns in ems. Let's put them together in a function that will take any number of columns and return the fixed width value of their size.

```
@function fixed($col) {  
  @return $col * em($grid-column + $grid-gutter)  
}
```

With the math in place we can now write a mixin that takes a column count as a parameter, then generates the perfect media query necessary to fit that number of columns on the screen. We can also build in some left and right margin for our layout by adding an additional gutter value (remembering that we already have one gutter built into our fixed function).

```
@mixin breakpoint($min) {
  @media (min-width: fixed($min) + em($grid-gutter)) {
    @content
  }
}
```

And, just like that, we've rewritten the media query. Instead of picking a minimum screen size for our layout, we can simply determine the number of columns needed. Let's add a wrapper class so that we can center our content on the screen.

```
@mixin breakpoint($min) {
  @media (min-width: fixed($min) + em($grid-gutter)) {
    .wrapper {
      width: fixed($min) - em($grid-gutter);
      margin-left: auto; margin-right: auto;
    }
    @content
  }
}
```

Designing content with a column count gives us nice, easy, whole numbers to work with. Sizing content, sidebars or widgets is now as simple as specifying a single-digit number.

```
@include breakpoint(8) {
  .main { width: fixed(5); }
  .sidebar { width: fixed(3); }
}
```

Those four lines of Sass just created a responsive layout for us. When the screen is big enough to fit eight columns, it will trigger a fixed width layout. And give widths to our main content and sidebar. The following is the outputted CSS...

```
@media (min-width: 41.25em) {  
  .wrapper {  
    width: 38.75em;  
    margin-left: auto; margin-right: auto;  
  }  
  .main { width: 25em; }  
  .sidebar { width: 15em; }  
}
```

## Demo

I've created a [Codepen demo](#) that demonstrates what we've covered so far. I've added to the demo some grid classes based on [Griddle](#) by Nicolas Gallagher to create a floatless layout. I've also added a CSS gradient overlay to help you visualize columns. Try changing the column variable sizes or the breakpoint includes to see how the layout reacts to different screen sizes.

## RESPONSIVE IMAGES

Responsive images are a serious problem, but I'm excited to see the community talk so passionately about a solution. Now, there are some excellent stopgaps while

we wait for something official, but these solutions require you to mirror your breakpoints in JavaScript or HTML. This poses a serious problem for my Sass-generated media queries, because I have no idea what the real values of my breakpoints are anymore. For responsive images to work, JavaScript needs to recognize which media query is active so that proper images can be loaded for that layout.

What I need is a way to label my breakpoints. Fortunately, people much smarter than I have figured this out. Jeremy Keith **devised a labeling method** by using CSS-generated content as the storage method for breakpoint labels. We can use this technique in our breakpoint mixin by passing a label as another argument.

```
@include breakpoint(8, 'desktop') { /* styles */ }
```

Sass can take that label and use it when writing the corresponding media query. We just need to slightly modify our breakpoint mixin.

```
@mixin breakpoint($min, $label) {  
  @media (min-width: fixed($min) + em($grid-gutter)) {  
  
    // label our mq with CSS generated content  
    body::before { content: $label; display: none; }  
  
    .wrapper {  
      width: fixed($min) - em($grid-gutter);  
      margin-left: auto; margin-right: auto;  
    }  
  }  
}
```

```
@content
  }
}
```

This allows us to label our breakpoints with a user-friendly string. Now that our media queries are defined and labeled, we just need JavaScript to step in and read which label is active.

```
// get css generated label for active media query
var label = getComputedStyle(document.body,
  '::before')['content'];
```

JavaScript now knows which layout is active by reading the label in the current media query—we just need to match that label to an image. I prefer to store references to different image sizes as data attributes on my image tag.

```
<img class="responsive-image" data-mobile="mobile.jpg"
data-desktop="desktop.jpg" />
<noscript></noscript>
```

These data attributes have names that match the labels set in my CSS. So while there is some duplication going on, setting a keyword like ‘tablet’ in two places is much easier than hardcoding media query values. With matching labels in CSS and HTML our script can marry the two and load the right sized image for our layout.



```
// get css generated label for active media query
var label = getComputedStyle(document.body,
  '::before')['content'];

// select image
var $image = $('img.responsive-image');

// create source from data attribute
$image.attr('src', $image.data(label));
```

### Demo

With some slight additions to our previous Codepen demo you can see this **responsive image technique** in action. While the above JavaScript will work it is not nearly robust enough for production so the demo uses a **jQuery plugin** that can accommodate multiple images, reloading on screen resize and fallbacks if something doesn't match up.

## CREATING A FRAMEWORK

This media query mixin and responsive image JavaScript are the center piece of a **front end framework** I use to develop websites. It's a fluid, mobile first foundation that uses the breakpoint mixin to structure fixed width layouts for tablet and desktop. Significant effort was focused on making this framework completely cross-browser. For example, one of the problems with using media queries is that essential desktop structure code ends up being

hidden from legacy Internet Explorer. `Respond.js` is an excellent polyfill, but if you're comfortable serving a single desktop layout to older IE, we don't need JavaScript. We simply need to capture layout code outside of a media query and sandbox it under an IE only class name.

```
// set IE fallback layout to 8 columns
$ie-support = 8;

// inside of our breakpoint mixin (but outside the media
query)
@if ($ie-support and $min <= $ie-support) {
  .lt-ie9 { @content; }
}
```

## PERSPECTIVE REGAINED

Thinking in columns means you are thinking about content layout. How big of a screen do you need for 12 columns? Who cares? Having Sass write media queries means you can use intuitive numbers for content layout. A fixed grid means more layout control and less edge cases to test than a fluid grid. Using CSS labels for activating responsive images means you don't have to duplicate breakpoints across separations of concern.

It's a harmonious blend of approaches that gives us something we need—responsive design that feels intuitive. And design that, from the very outset, focuses on what matters most. Just like our kindergarten teachers taught us: It's what's inside that counts.

## ABOUT THE AUTHOR



**Les James** was born with the heart of an artist but the brain of a developer. Fortunately for him, front-end development is the perfect intersection between the art and science of crafting a website and he gets to pursue his passion for design at **Capstrat** by transforming ideas into code. Les is always thirsty for knowledge, so why don't you drop some on him at [@lesjames](#).

# 14. Using Questionnaires for Design Research

---

Emma Boulton

**How do  
you ask**

[24ways.org/201214](http://24ways.org/201214)

**the right questions?**

In this article, I share a bunch of tips and practical advice on how to write and use your own surveys for design research.

I'm an audience researcher – I'm not a designer or developer. I've spent much of the last thirteen years working with audience data both in creative agencies and on the client-side. I'm also a member of the **Market Research Society**. I run user surveys and undertake user research for our clients at the design studio I run with my husband – Mark Boulton Design.

## **SO LET'S GET STARTED!**

### **Who are you designing for?**

Good web designers and developers appreciate the importance of understanding the audience they are designing or building a website or app for. I'm assuming

that because you are reading a quality publication like 24 ways that you fall into this category, and so I won't begin this article with a lecture.

Suffice it to say, it's a good idea to involve research of some sort during the life cycle of every project you undertake. I don't just mean visual or competitor research, which of course is also very important. I mean looking at or finding your own audience or user data. Whether that be auditing existing data or research available from the client, carrying out user interviews, A/B testing, or conducting a simple questionnaire with users, *any* research is better than none. If you create personas as a design tool, they should always be based on research, so you will need to have plenty of data to hand for that.

### **Where do I start?**

In the initial kick-off stages of a project, it's a good idea to start by asking your client (when working in-house you still have a client – you might even be the client) what research or audience data they have available. Some will have loads – analytics, surveys, focus groups and insights – from talking to customers. Some won't have much at all and you'll be hard pressed to find out much about the audience. It's best to review existing research first without rushing headlong into doing new research. Get a picture of what the data tells you and perhaps get this into

a document – who, what, why and how are they using this website or app? What gaps are there in existing research? What else do you need to know? Then you can decide what else you need to do to plug these gaps. Think about the information first before deciding on the methodology. The rest of my article talks mostly about running **self-completion online surveys**. You can of course do face-to-face surveys, self-completion written questionnaires or phone polls, but I won't cover those here. That's for another article.

### **Why run a survey?**

Surveys are great for getting a broad picture of your audience. As long as they are designed carefully, you can create an overview of them, how they use the site and their opinions of it, with an idea of which parts of this picture are more important than others. By using a limited amount of open-ended questions, you can also get some more qualitative feedback or insights on your website or app. The clients we work with surprisingly often don't have much in the way of audience research available, even basic analytics, so I will often suggest running a short survey, just to create a picture of who is out there.

## OK, WHAT SHOULD I DO FIRST?

Before you rush into writing questions, stop and think about what you're trying to find out. Remember being in school when you studied science and you had to propose a hypothesis? This could be a starting point – something to prove or disprove. Or, even better, write a research brief. It doesn't have to be long; it can be just a sentence that encapsulates what you're trying to do, like a good creative brief. For the purposes of this article, I created a short, slightly silly survey on **Christmas and beliefs in Father Christmas**.

My research brief was:

To find out more about people's beliefs about Father Christmas and their experiences of Christmas.

Inevitably, as you start thinking of what questions to ask, you will find that you go off at tangents or your client will want you to add in everything but the kitchen sink. In order for your questionnaire not to get too long and lose focus, you could write lists of what it is and what it's not. This is how I'd apply it to my Christmas questionnaire example:

### WHAT IT IS ABOUT

- How people communicate with Father Christmas

- If someone's background has affected their likelihood of believing in Father Christmas

### WHAT IT IS NOT ABOUT

- What colour to change Father Christmas's coat to
- Father Christmas's elves

Let's get down to business: the questions.

### KINDS OF QUESTIONS

There are two basic kinds of questions: open-ended and closed. Closed questions limit answers by giving the respondent a number of predefined lists of options to choose from. Typically, these are multiple-choice questions with a list of responses. You can either select one or tick all that apply. Another useful type of closed question I often use is a rating scale, where a respondent can assess a situation along a continuum of values. These can also be useful as a measure of advocacy or strength of feeling about something. There is a standard measure called the **Net Promoter score**, which measures how likely someone is to recommend your product or service to a friend or acquaintance. It's a useful benchmark as you can compare your scores to others in a similar sector.



## Using Questionnaires for Design Research

7. How often do you communicate with Father Christmas in the two months before Christmas?

Daily  
 Once a week or more often  
 Once a month  
 Rarely  
Other (please specify)

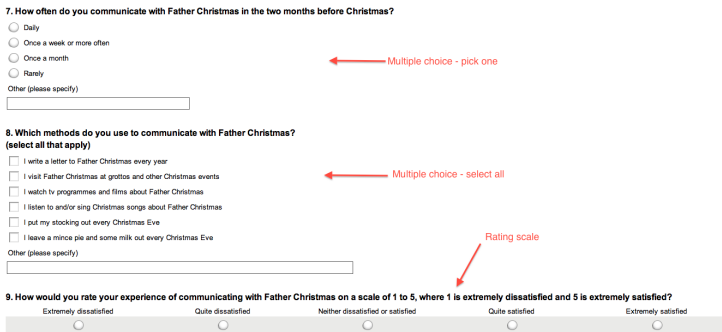
8. Which methods do you use to communicate with Father Christmas?  
(select all that apply)

I write a letter to Father Christmas every year  
 I visit Father Christmas at grottos and other Christmas events  
 I watch tv programmes and films about Father Christmas  
 I listen to and/or sing Christmas songs about Father Christmas  
 I put my stocking out every Christmas Eve  
 I leave a mince pie and some milk out every Christmas Eve  
Other (please specify)

9. How would you rate your experience of communicating with Father Christmas on a scale of 1 to 5, where 1 is extremely dissatisfied and 5 is extremely satisfied?

Extremely dissatisfied    Quite dissatisfied    Neither dissatisfied or satisfied    Quite satisfied    Extremely satisfied

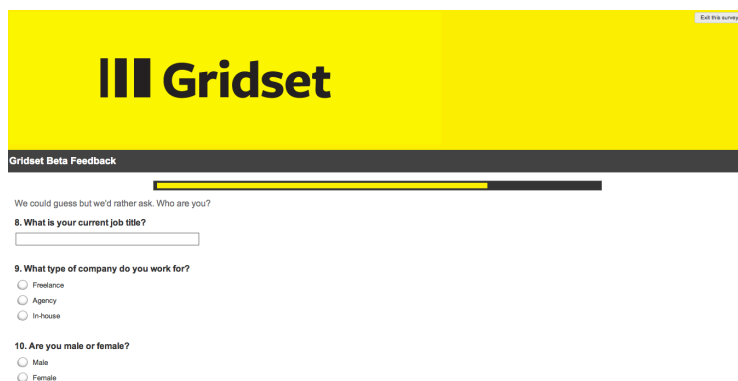


Open-ended questions often take the form of a statement which requires a response. Generally, respondents are given a text box to fill in. It's useful to limit this in some way so that people have an idea of how long the expected response should be; for example, a single line for an email address (Q18), or a larger text area for a longer response (Q6).

If you plan to send your survey out to a large number of people, I would suggest using mostly closed questions, unless you want to spend a long time wading through comments and hand-coded responses. I'd always advise adding a general request at the end of a survey ('Is there anything else you'd like to tell us?'). You'd be surprised how many interesting and insightful comments people will add.

There are times when it's better to provide an open-ended text box rather than a predefined list makes assumptions about your audience's groupings. For

example, we ran a short survey for our Gridset beta testers and rather than assume we knew who they were, we decided to ask an open-ended question: “What is your current job title?”



The screenshot shows a survey interface with a yellow header containing the Gridset logo and the text 'Gridset Beta Feedback'. Below the header, there is a progress bar and a question: '8. What is your current job title?' with a text input field. The next question is '9. What type of company do you work for?' with radio button options for 'Freelance', 'Agency', and 'In-house'. The final question is '10. Are you male or female?' with radio button options for 'Male' and 'Female'. There is an 'Exit this survey' link in the top right corner.

The analysis took quite a bit longer than responses using a predefined list, but it meant that we were able to make sure we didn't miss anyone. And next time we run a survey for Gridset, I can use the responses gathered from this survey to help create a predefined list to make analysis easier.

## WHAT TO ASK

The questions to ask depend on what you want to know, but your brief and lists of what the survey is and isn't should help here. I always ask the design team and client to give me ideas of what they are interested in finding out, and combine this with a mix of new and standard

questions I have used in other surveys. I find **Survey Monkey's question bank** a very useful source of example questions and help with tricky wording.

I always include simple demographics so I can compare my results to the population at large or internet users as a whole – just going on age, gender and location can be quite illuminating. For example, with the Christmas survey, I can see that the respondents were typical of the online design and dev community, mainly young and male.

If appropriate, I add questions on disability, ethnic background, religion and community of interest. Questions about ethnicity, religion, sexual preference, disability and other sensitive subjects can feel awkward and difficult to ask. This is not a good reason to not ask them. Perhaps you're working for a public sector client, like a local council, so it's likely you will need to consider groups of people who maybe under-represented, who may have differing views to others, or who you need to look at specifically as a subset.

### HOW TO ASK

Although they may seem clunky and wordy, it's often best to use the census wording or professional body wording for such demographic questions. For example, I used the **UK census 2011** wording for Wales on my Christmas questionnaire in my questions on **religion** [PDF] (Q16) and

ethnicity [PDF] (Q17). I had to adapt them slightly for the Survey Monkey format – self-completion online, rather than pen and paper – which is why “White Welsh” came up as the first option for the ethnicity question. For similar questions for US audiences, try the **Census Bureau** website.

When conducting a survey for a project that has a global audience, you need to consider who your primary audience is. For example, I recently created a questionnaire for a global news website. A large proportion of its audience is based in the USA, so I was careful to word things in a way Americans would find familiar. I used the US ethnic background census question wording and options, and looked at data for US competitor news websites to decide which to include.

You should also consider people whose first language isn't English. Working as an audience researcher at **BBC Wales**, every survey we did was bilingual. I've also recently run a user survey in Arabic using **Google Forms**. During this project, we found that while Survey Monkey supports different languages, including Arabic, the text ran left to right with no option to change it to right to left – an essential when it comes to reading Arabic! If research is a deliverable in a client project, and you know you'll need to conduct it in a foreign language, always build in extra time for translation at both the questionnaire design and analysis stages. Make sure you

also allow for plenty of checks. In this case we had to change to Google Forms after initially creating our survey with Survey Monkey to get the functionality we needed.

## LOOK AND FEEL

Think about the survey as another way your audience will experience your brand. Take care getting the tone of voice right. There are plenty of great articles and books out there about tone of voice – try *Letting Go of the Words* by Ginny Redish for starters, or *Brand Language* by Liz Doig. The basic rule of thumb is to sound like a human, and use clear and friendly language. If, like me, you are lucky enough to work with journalists or copy editors, you should ask for their help, particularly in the preamble, linking text and closing statements. I find it helpful to break my questions down into sections and to have a page for each. I then have an introductory piece of text for each section to guide the respondent through the survey.

You should also make sure you check with your designers how your survey looks – use a company logo and branding, and make the typography legible. Many survey apps like Survey Monkey and Google Forms have a progress bar. This is helpful for users to see how far through your survey they are. I generally time the survey and give an indication in the preamble: “This survey will only take five minutes of your time.”

You also need to think about how you will technically serve the questionnaire. For example, will it be via email, social media, a pop-up or lightbox on your website, or (not recommended but possible) in an ad space?

## **ETHICAL CONSIDERATIONS**

Something else to think about are any local laws that govern how you collect and store data, such as the **Data Protection Act** in the UK. As a member of the Market Research Society, I am also obliged to consider its guidelines, but even if you're not, it's always a good idea to deal with personal data ethically.

If you collect personal data that can identify individuals, you must ask their permission to share it with others, and store it securely for no longer than two years. If you want to contact people afterwards, you must ask for their permission. If you ask for email addresses, as I did in question 18, you have a ready-made sample for a further survey, interviews or focus groups. Remember, you shouldn't survey people under sixteen years old without the permission of their parents or legal guardians, so if you know your website is likely to be used by children, you must ask for verification of age early on, and your survey should close someone answers that they are under sixteen. The **ESOMAR guidelines for online research**

[PDF] are well worth reading, as they go into detail about such issues, as well as privacy guidelines – using cookies, storing IP addresses, and so on.

## TOOLS

Unless you work in-house and have proprietary software, or at a market research agency and you're using specialist software such as **Snap** or **IBM SPSS Statistics** (previously just SPSS), you will need to use a good tool to run your survey, collect your responses and, ideally, help with the analysis. I like **Survey Monkey** because of the question bank and analysis tools. The software graphs your results and does simple cross-tabbing and filtering. What this means is you can slice the data in more interesting ways and delve a bit deeper. For example, in the Gridset questionnaire I mentioned earlier, I cross-tabbed responses to questions against whether a person worked in-house, for an agency or as a freelancer.

Other well known online tools that I also use from time to time are **Wufoo** and **Google Forms**. **Smart Surveys** is a similar service to Survey Monkey and it's used by many leading brands in the UK. Snap Surveys mentioned above is a well-established player in the market research scene, used a lot for face-to-face surveys and also on tablets and smartphones.

## ANALYSIS

Analysis is often overlooked but is as important as the design of the questionnaire. Don't just rely on looking at the summary report and charts generated as standard by your form or survey software. Spend time with your data. Spend at least a week now and then if you can, looking at the data. Keep coming back to it and tweaking or cutting it a different way to see if there are any different pictures. Slice it up in different ways to reveal new insights. **Here is the data** from my dummy survey (apart from the open-ended responses).

For open-ended questions, you can analyse collaboratively. Print and cut out the open-ended responses and do a cluster analysis or affinity sort with a colleague.





Discussing the comments helps you to understand them. You will also find the design team are more likely to buy into the research as they have uncovered the insights for

themselves. Always make sure to treat open-ended responses sensitively and don't share anything publicly in a way that identifies the respondent.

## **Write a report**

Never hand over a dataset to your client without a summary of the findings. Data on its own can be skewed to suit the reader's needs, and not everyone is able to find the story in a dataset. Even if it's not a deliverable, it's always a good idea to capture your findings in a report of some sort. Use graphs sparingly to show really interesting things or to aid the reader's understanding. I have written a quick dummy report using the data from the Christmas questionnaire so you can see how it's done.

I highly recommend Brian Suda's book *A Practical Guide to Designing with Data* for tips on how to present data effectively, but that's a subject that benefits a whole article (indeed book) in itself.

I am not a designer. I am a researcher, so I never write design recommendations in a report unless they have been talked about or suggested by the designers I work with. More often, I write up the results and we talk about them and what impact they have on the project or design. Often they lead to more questions or further research.

So that's it: a brief introduction to using questionnaires for design research. Here's a quick summary to remind you what I have talked about, and a list of resources if you're interested in reading further.

### **TOP 10 THINGS TO REMEMBER WHEN USING QUESTIONNAIRES FOR DESIGN RESEARCH:**

1. Start by auditing existing research to identify gaps in data.
2. Write a research brief. Work out exactly what you're trying to find out – what is the survey about, and what is it not about?
3. The two basic kinds of questions are open-ended and closed.
4. Closed questions limit responses by giving the respondent a number of predefined lists of options to choose from (multiple choice, rating scales, and so on).
5. Open-ended questions are often in the form of a statement which requires a response. Always ask one at the end of a questionnaire.
6. Always include simple demographics to enable you to compare your sample against the population in general.
7. It's best to use official census or professional body wording for questions on ethnicity, disability and religion.
8. Be sure to think carefully about your tone of voice and the look of your questionnaire.

9. Pay attention to guidelines and laws on storing personal data, cookies and privacy.
10. Invest plenty of time in analysis and report writing. Don't just look at the obvious – dig deep for more interesting insights.

## **SOME USEFUL RESOURCES FOR FURTHER STUDY**

### **Online research**

- *Design Research: Methods and Perspectives* edited by Brenda Laurel
- *Online Research Essentials* by Brenda Russell and John Purcell
- *Handbook of Online and Social Media Research* by Ray Poynter
- ESOMAR guidelines for online research [PDF]
- Online questionnaires

### **Market research books on questionnaire design**

- *Using Questionnaires in Small-Scale Research: A Beginner's Guide* by Pamela Munn
- *Questionnaire Design* by A N Oppenheim
- *Developing a Questionnaire* by Bill Gillham

## ABOUT THE AUTHOR



**Emma** has been helping clients understand their audiences for the better part of the last 14 years. She cut her research teeth in the brave new world of online advertising, before moving to the Audiences team at the BBC. She's now the Research Director at **Mark Boulton Design** and works as part of a small but exceptional team creating great web experiences for clients such as CERN, Al Jazeera and Global Witness. Emma is also the Editor in Chief of indie publisher, **Five Simple Steps**.

# 15. A Harder-Working Class

---

Nathan Ford

[24ways.org/201215](http://24ways.org/201215)

Class is only becoming more important. Focusing on its original definition as an attribute for grouping (or classifying) as well as linking HTML to CSS, recent front-end development practices are emphasizing class as a vessel for structured, modularized style packages. These patterns reduce the need for repetitive declarations that can seriously bloat file sizes, and instil human-readable understanding of how the interface, layout, and aesthetics are constructed.

In the next handful of paragraphs, we will look at how these emerging practices – such as **object-oriented CSS** and **SMACSS** – are pushing the relevance of class. We will also explore how HTML and CSS architecture can be further simplified, performance can be boosted, and CSS utility sharpened by combining class with the attribute selector.

## A PRIMER ON ATTRIBUTE SELECTORS

While attribute selectors were introduced in the CSS 2 spec, they are still considered rather exotic. These well-established and well-supported features give us vastly improved flexibility in targeting elements in CSS, and offer us opportunities for smarter markup. With an attribute selector, you can directly style an element based on any of its unique – or uniquely shared – attributes, without the need for an ID or extra classes. Unlike pseudo-classes, pseudo-elements, and other exciting features of CSS3, attribute selectors do not require any browser-specific syntax or prefix, and are even supported in Internet Explorer 7.

For example, say we want to target all anchor tags on a page that link to our homepage. Where otherwise we might need to manually identify and add classes to the HTML for these specific links, we could simply write:

```
[href=index.html] { }
```

This selector reads: target every element that has an href attribute of “index.html”.

Attribute selectors are more faceted, though, as they also give us some very simple regular expression-like logic that helps further narrow (or widen) a selector’s scope. In our previous example, what if we wanted to also give indicative styles to any anchor tag linking to an external

site? With no way to know what the exact href value would be for every external link, we need to use an expression to match a common aspect of those links. In this case, we know that all external links need to start with “http”, so we can use that as a hook:

```
[href^=http] { }
```

The selector here reads: target every element that has an href attribute that begins with “http” (which will also include “https”). The ^= means “starts with”. There are a few other simple expressions that give us a lot of flexibility in targeting elements, and I have found that a deep understanding of **these and other selector types** to be very useful.

## THE CLASS-ATTRIBUTE SELECTOR

By matching classes with the attribute selector, CSS can be pushed to accomplish some exciting new feats. What I call a class-attribute selector combines the advantages of classes with attribute selectors by targeting the class attribute, rather than a specific class. Instead of selecting `.urgent`, you could select `[class*=urgent]`. The latter may seem like a more verbose way of accomplishing the former, but each would actually match two subtly different groups of elements.



Eric Meyer first explored the possibility of using classes with attribute selectors **over a decade ago**. While his interest in this technique mostly explored the different facets of the syntax, I have found that using class-attribute selectors can have distinct advantages over either using an attribute selector or a straightforward class selector.

First, let's explore some of the subtleties of why we would target class before other attributes:

- Classes are ubiquitous. They have been supported since the HTML 4 spec was released in 1999. Newer attributes, such as the custom data attribute, have only recently begun to be adopted by browsers.
- Classes have multiple ways of being targeted. You can use the class selector or attribute selector (`.classname` or `[class=classname]`), allowing more flexible specificity than resorting to an ID or `!important`.
- Classes are already widely used, so adding more classes will usually require less markup than adding more attributes.
- Classes were designed to abstractly group and specify elements, making them the most appropriate attribute for styling using object-oriented methods (as we will learn in a moment).

Also, as Meyer pointed out, we can use the class-attribute selector to be more strict about class declarations. Of these two elements:

```
<h2 class="very urgent">
```

```
<h2 class="urgent">
```

...only the second h2 would be selected by `[class=urgent]`, while `.urgent` would select both. The use of `=` matches any element with the exact class value of "urgent". Eric explores these nuances further in his series on attribute selectors, but perhaps more dramatic is the added power that class-attribute selectors can bring to our CSS.

## **MORE OBJECT-ORIENTED, MORE SCALABLE AND MODULAR**

Nicole Sullivan has been pushing abstracted, object-oriented thinking in CSS development for years now. She has shared **stacks of knowledge** on how behemoth sites have seen impressive gains in maintenance overhead and CSS file sizes by leaning heavier on classes derived from common patterns. **Jonathan Snook** also speaks, writes and is genuinely passionate about improving our markup by using more stratified and modular class name conventions. With SMACSS, he shows this to be highly useful across sites – both complex and simple – that exhibit repeated design patterns. Sullivan and Snook both

push the use of class for styling over other attributes, and many front-end developers are fast advocating such thinking as best practice.

With class-attribute selectors, we can further abstract our CSS, pushing its scalability. In his [chapter on modules](#), Snook gives the example of a `.pod` class that might represent a certain set of styles. A `.pod` style set might be used in varying contexts, leading to CSS that might normally look like this:

```
.pod { }  
form .pod { }  
aside .pod { }
```

According to Snook, we can make these styles more portable by targeting more verbose classes, rather than context:

```
.pod { }  
.pod-form { }  
.pod-sidebar { }
```

...resulting in the following HTML:

```
<div class="pod">  
<div class="pod pod-form">  
<div class="pod pod-sidebar">
```

This divorces the `<div>`'s styles from its context, making it applicable to any situation in which it is needed. The markup is clean and portable, and the classes are imbued with meaning as to what module they belong to.

Using class-attribute selectors, we can simplify this further:

```
[class*=pod] { }  
.pod-form { }  
.pod-sidebar { }
```

The `*` tells the browser to look for any element with a class attribute containing “pod”, so it matches “pod”, “pod-form”, “pod-sidebar”, etc. This allows only one class per element, resulting in simpler HTML:

```
<div class="pod">  
  <div class="pod-form">  
    <div class="pod-sidebar">
```

We could further abstract the concept of “form” and “sidebar” adjustments if we knew that each of those alterations would always need the same treatment.

```
/* Modules */  
[class*=pod] { }  
[class*=btn] { }  
  
/* Alterations */  
[class*=-form] { }  
[class*=-sidebar] { }
```

In this case, all elements with classes appended “-form” or “-sidebar” would be altered in the same manner, allowing the markup to stay simple:

```
<form>
  <h2 class="pod-form">
  <a class="btn-form" href="#">

<aside>
  <h2 class="pod-sidebar">
  <a class="btn-sidebar" href="#">
```

## 50+ SHADES OF SPECIFICITY

Classes are just powerful enough to override element selectors and default styling, but still leave room to be trumped by IDs and !important styles. This makes them more suitable for object-oriented patterns and helps avoid messy specificity issues that can not only be a pain for developers to maintain, but can also affect a site’s performance. As Sullivan notes, “In almost every case, classes work well and have fewer unintended consequences than either IDs or element selectors”.

**Proper use of specificity** and cascade is crucial in building straightforward, efficient CSS.

One interesting aspect of attribute selectors is that they can be compounded for increasing levels of specificity. Attribute selectors are assigned a specificity level of ten,

the same as class selectors, but both class and attribute selectors can be chained together, giving them more and more specificity with each link. Some examples:

```
.box { }  
/* Specificity of 10 */
```

```
.box.promo { }  
/* Specificity of 20 */
```

```
[class*=box] { }  
/* Specificity of 10 */
```

```
[class*=box][class*=promo] { }  
/* Specificity of 20 */
```

You can chain both types together, too:

```
.box[class*=promo] { }  
/* Specificity of 20 */
```

I was amused to find, though, that you can chain the exact same class and attribute selectors for **infinite levels of specificity**

```
.box { }  
/* Specificity of 10 */
```

```
.box.box { }  
/* Specificity of 20 */
```

```
.box.box.box { }  
/* Specificity of 30 */
```

```
[class*=box] { }  
/* Specificity of 10 */  
  
[class*=box][class*=box] { }  
/* Specificity of 20 */  
  
[class*=box][class*=box][class*=box] { }  
/* Specificity of 30 */  
  
.box[class*=box].box[class*=box] { }  
/* Specificity of 40 */
```

To override `.box` styles for `promo`, we wouldn't need to add an ID, change the order of `.promo` and `.box` in the CSS, or resort to an `!important` style. Granted, any issue that might need this fine level of specificity tweaking could probably be better solved with clever cascades, but having options never hurts.

## SMARTER CSS

One of the most powerful aspects of the class-attribute selector is its ability to expand the simple logic found in CSS. When developing **Gridset** (an online tool for building grids and outputting them as CSS), I realized that with the right class name conventions, class-attribute selectors would allow the CSS to be smart enough to automatically adjust for column offsets without the need for extra

classes. This imbued the CSS output with logic that other frameworks lacked, and makes a developer's job much easier.

Say you need an element that spans column five (c5) to column six (c6) on your grid, and is preceded by an element spanning column one (c1) to column three (c3). The CSS can anticipate such a scenario:

```
.c1-c3 + .c5-c6 {  
    margin-left: 25%; /* ...or the width of column four plus  
    two gutter widths */  
}
```

...but to accommodate all of the margin offsets that could span that same gap, we would need to write a rather protracted list for just a six column grid:

```
.c1-c3 + .c5-c6,  
.c1-c3 + .c5,  
.c2-c3 + .c5-c6,  
.c2-c3 + .c5,  
.c3 + .c5-c6,  
.c3 + .c5 {  
    margin-left: 25%;  
}
```

Now imagine how the verbosity compounds when we repeat this type of declaration for every possible margin in a grid. The more columns added to the grid, the longer this selector list would get, too, making the CSS harder for



the developer to maintain and slowing the load time. Using class-attribute selectors, though, this can be much simpler:

```
[class*=c3] + [class*=c5] {  
  margin-left: 25%;  
}
```

I've detailed how we extract as much logic as possible from as little CSS as needed on the [Gridset blog](#).

## MORE FLEXIBLE SELECTORS

In a recent project, I was working with **Drupal**-generated classes to change styles for certain special pages on a site. Without being able to change the code base, I was left trying to find some specific aspect of the generated HTML to target. I noticed that every special page was given a prefixed class, unique to the page, resulting in CSS like this:

```
.specialpage-about,  
.specialpage-contact,  
.specialpage-info,  
...
```

...and the list kept growing with each new special page. Such bloat would lead to problems down the line, and add development overhead to editorial decisions, which was a situation we were trying to avoid. I was easily able to fix this, though, with a concise class-attribute selector:

[class\*=specialpage-]

The CSS was now flexible enough to accommodate both the editorial needs of the client, and the development restrictions of the CMS.

## SELECTOR PERFORMANCE

As Snook tells us in his chapter on **Selector Performance**, selectors are read by the browser from right to left, matching every element that adheres to each rule (or part of the selector). The more specific we can make the right-most rules – and every other part of your selectors – the more performant your CSS will be. So this selector:

```
.home-page .promo .main-header
```

...would be more performant than:

```
.home-page div header
```

...because there are likely many more header and div elements on the page, but not so many elements with those specific classes.

Now, the class-attribute selector could be more general than a class selector, but not by much. I ran numerous tests based on the work of Steve Souders (and a few others) to test a class-attribute selector against a normal class selector. Given that Javascript will freeze during style rendering, I created a script that will add, then

remove, a stylesheet on a page 5000 times, and measure only the time that elapses during the rendering freeze. The script runs four tests, essentially: one where a class selector and class-attribute Selector match a single element, and one they match multiple elements on the page.

After running the test over 100 times and averaging the results, I have not seen a significant difference in rendering times. (As of this writing, the class-attribute selector has been 0.398% slower on average.) **View the results here.**

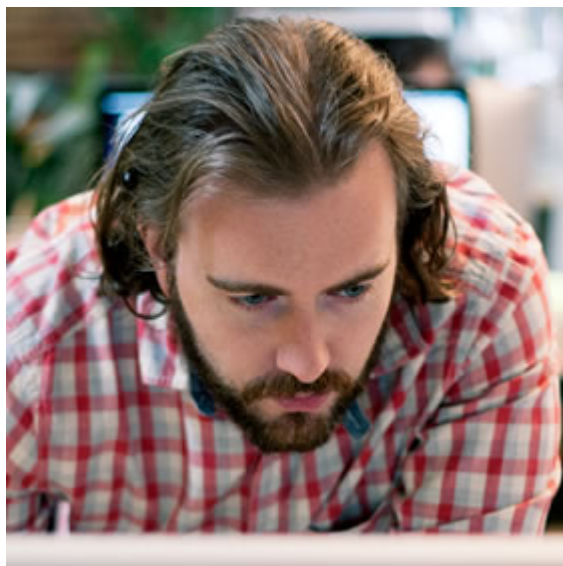
Given the sheer amount of bytes potentially saved by reducing selector lists, though, I am confident class-attribute selectors could shorten load times on larger sites and, at the very least, save precious development time.

## CONCLUSION

With its flexibility and broad remit, class has at times been derided as too lenient, allowing CMSes and lazy developers to fill its values with presentational hacks or verbose gibberish. There have even been calls for an **early retirement**. Class continues, though, to be one of our most crucial tools.

Front-end developers are rightfully eager to expand production abilities through innovations such as **Sass** or **LESS**, but this should not preclude us from honing the tools we already know as well. Every technique demonstrated in this article was achievable over a decade ago and most of the same thinking could be applied to IDs, `rels`, or any other attribute (though the reasons listed above give class an edge). The recent advent of methods such as object-oriented CSS and SMACSS shows there is still much room left to expand what simple HTML and CSS can accomplish. Progress may not always be found in the innovation of our tools, but through sharpening our understanding of them.

## **ABOUT THE AUTHOR**



**Nathan Ford** is Creative Director at **Mark Boulton Design** where he helps a small, talented team of designers build beautiful experiences for a queue of clients such as Al Jazeera, ESPN, and CERN. He is also lead developer on **Gridset**, an online tool for building grid systems. Read more of his infrequent writing on his blog, **Art=Work**, where he shares thoughts and tools to make working on the web a bit easier (hopefully), or follow **@nathan\_ford** on Twitter.

# 16. How to Make Your Site Look Half-Decent in Half an Hour

---

Anna Powell-Smith

[24ways.org/201216](http://24ways.org/201216)

Programmers like me are often intimidated by design – but a little effort can give a huge return on investment. Here are one coder’s tips for making any site quickly look more professional.

I am a programmer. I am not a designer. I have a degree in computer science, and I don’t mind Comic Sans. (It looks cheerful. Move on.)

But although I am a programmer, I want to make my sites look attractive. This is partly out of vanity, and partly realism. Vanity because I want people to think my work is good, and realism because the research shows that people won’t think a site is credible unless it also looks attractive.

For a very long time after I became a programmer, I was scared of design. Design seemed to consist of complicated rules that weren't written down anywhere, plus an unlearnable sense of taste, possessed only by a black-clad elite.

But a little while ago, I decided to do my best to hack what it took to make my own projects look vaguely attractive. And although this doesn't come close to the effect a professional designer could achieve, gathering these resources for improving a site's look and feel has been really helpful.

If I hadn't figured out some basic design shortcuts, it's unlikely that a **weekend hack of mine** would have ended up on **page three of the Daily Mail**. And too often now, I see excellent programming projects that don't reach the audience they deserve, simply because their design doesn't match their execution.

So, if you are a developer, my Christmas present to you is this: my own collection of hacks that, used rightly, can make your personal programming projects look professional, quickly. None are hard to learn, most are free, and they let you focus on writing code.

One thing to note about these tips, though. They are a personal, pragmatic compilation. They are suggestions, not a definitive guide. You will definitely get better results by working with a professional designer, and by studying design more deeply.

If you are a designer, I would love to hear your suggestions for the best tools that I've missed, and I'd love to know how we programmers can do things better.

With that, on to the tools...

## 1. USE BOOTSTRAP

If you're not already using **Bootstrap**, start now. I really think that Bootstrap is one of the most significant technical achievements of the last few years: it democratizes the whole process of web design.

Essentially, Bootstrap is a grid system, with a bunch of common elements. So you can lay your site out how you want, drop in simple elements like forms and tables, and get a good-looking, consistent result, without spending hours fiddling with CSS. You just need HTML.

Another massive upside is that it makes it easy to make any site responsive, so you don't have to worry about writing media queries. Go, **get Bootstrap** and **check out**



the examples. To keep your site lightweight, you can customize your download to include only the elements you want.

If you have more time, then Mark Otto's article on why and how Bootstrap was created at Twitter is well worth a read.

## 2. PIMP BOOTSTRAP

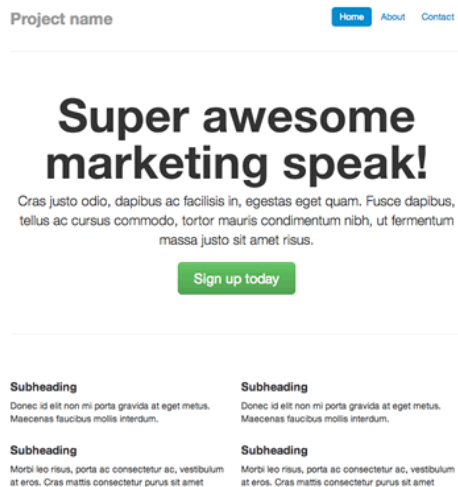
Using Bootstrap is already a significant advance on not using Bootstrap, and massively reduces the tedium of front-end development. But you also run the risk of creating Yet Another Bootstrap Site, or Hack Day Design, as it's known.

If you're really pressed for time, you could buy a theme from **Wrap Bootstrap**. These are usually created by professional designers, and will give a polish that we can't achieve ourselves. Your site won't be unique, but it will look good quickly.

Luckily, it's pretty easy to make Bootstrap not look too much like Bootstrap – using fonts, CSS effects, background images, colour schemes and so on. Most of the rest of this article covers different ways to achieve this.

We are going to customize this Bootstrap example page.

This already has some custom CSS in the <head>. We'll pull it all out, and create a new CSS file, `custom.css`. Then we add a reference to it in the header. Now we're ready to start customizing things.



### 3. FONTS

Web fonts are one of the quickest ways to make your site look distinctive, modern, and less Bootstrappy, so we'll start there.

First, we can add some sweet fonts, from **Google Web Fonts**. The intimidating bit is choosing fonts that look nice together. Luckily, there are plenty of suggestions around

the web: we're going to use one of **DesignShack's suggested free Google Fonts pairings**. Our fonts are **Corben** (for headings) and **Nobile** (for body copy).

Then we add these files to our <head>.

```
<link href="http://fonts.googleapis.com/
css?family=Corben:bold" rel="stylesheet" type="text/css">
<link href="http://fonts.googleapis.com/
css?family=Nobile" rel="stylesheet" type="text/css">
```

...and this to custom.css:

```
h1, h2, h3, h4, h5, h6 {
    font-family: 'Corben', Georgia, Times, serif;
}
p, div {
    font-family: 'Nobile', Helvetica, Arial, sans-serif;
}
```

Now our example looks like this. It's not going to win any design awards, but it's immediately better:

# Super awesome marketing speak!

Cras justo odio, dapibus ac facilisis in, egestas eget quam. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

Sign up today

## Subheading

Donec id elit non mi porta gravida at eget metus. Maecenas faucibus mollis interdum.

## Subheading

Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras mattis consectetur purus sit amet fermentum.

## Subheading

Donec id elit non mi porta gravida at eget metus. Maecenas faucibus mollis interdum.

## Subheading

Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras mattis consectetur purus sit amet fermentum.

I also recommend the web font services [Fontdeck](#), or [Typekit](#) – these have a wider selection of fonts, and are worth the investment if you regularly need to make sites look good. For more font combinations, [Just My Type](#) suggests appealing pairings from Typekit. Finally, you can experiment with type pairing ideas at [Type Connection](#). For the design background on pairing fonts, [Typekit's post](#) is worth a read.

## 4. TEXTURES

An instant way to make a site look classy is to use textures. You know the grey, stripy, indefinably elegant background on [24ways.org](#)? That.

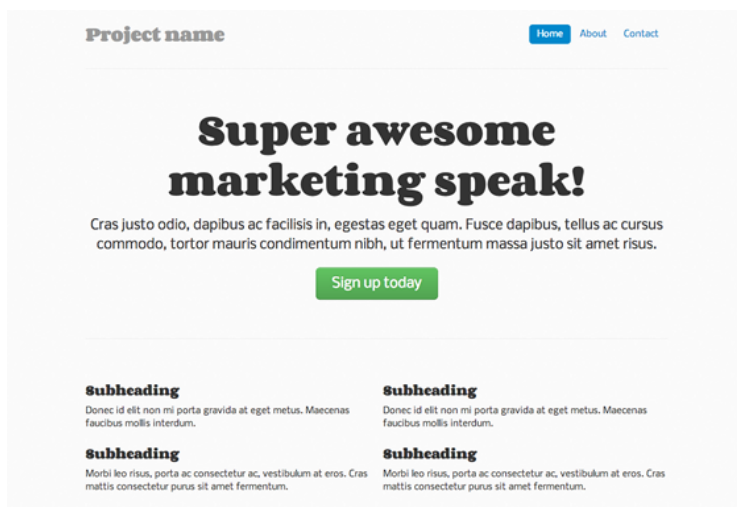
## How to Make Your Site Look Half-Decent in Half an Hour

If only there was a superb resource listing attractive, free, ready-to-use textures... Oh wait, there's Atle Mo's **Subtle Patterns**.

We're going to use **Cream Dust**, for an effect that can only be described as subtle. We download the file to a new `/img/` directory, then add this to the CSS file:

```
body {  
    background: url(/img/cream_dust.png) repeat 0 0;  
}
```

Bang:



For some design background on patterns, I recommend reading through **Smashing Magazine's guidelines on textures**. (TL;DR version: use textures to enhance beauty, and clarify the information architecture of your site; but don't overdo it, or inadvertently obscure your text.)

Still more to do, though. Onwards.

## 5. ICONS

Last year's 24 ways taught us to use icon fonts for our site icons.

This is great for the time-pressed coder, because icon fonts don't just cut down on HTTP requests – they're a lot quicker to set up than image-based icons, too.

Bootstrap ships with an extensive, free for commercial use icon set in the shape of **Font Awesome**. Its icons are safe for screen readers, and can even be made to work in IE7 if needed (we're not going to bother here).

To start using these icons, just download Font Awesome, and add the `/fonts/` directory to your site and the `font-awesome.css` file into your `/css/` directory. Then add a reference to the CSS file in your header:

```
<link rel="stylesheet" href="/css/font-awesome.css">
```

Finally, we'll add a truck icon to the main action button, as follows. Why a truck? Why not?

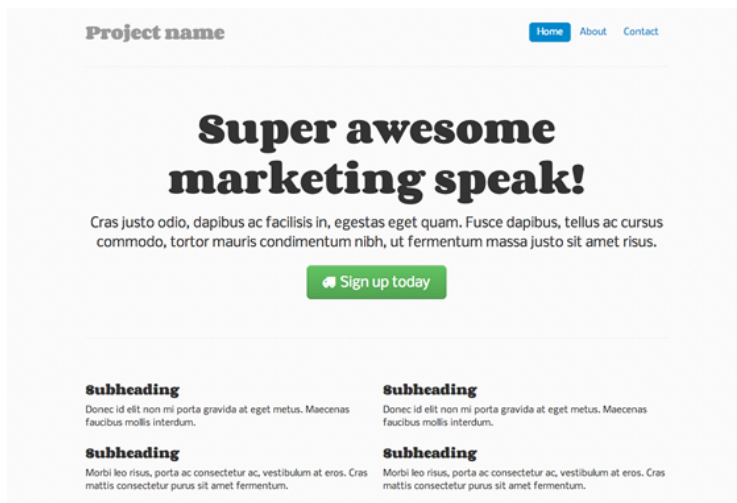
## How to Make Your Site Look Half-Decent in Half an Hour

```
<a class="btn btn-large btn-success" href="#"><i class="icon-truck"></i> Sign up today</a>
```

We'll also tweak our CSS file to stop the icon nudging up against the button text:

```
.jumbotron .btn i {  
    margin-right: 8px;  
}
```

And this is how it looks:



Not the most exciting change ever, but it livens up the page a bit. The licence is CC-BY-3.0, so we also include a mention of Font Awesome and its URL in the source code.

If you'd like something a little more distinctive, **Shifticons** lets you pay a few cents for individual icons, with the bonus that you only have to serve the icons you actually use, which is more efficient. Its icons are also friendly to screen readers, but won't work in IE7.

## 6. CSS3

The next thing you could do is add some CSS3 goodness. It can really help the key elements of the site stand out.

If you are pressed for time, just adding **box-shadow** and **text-shadow** to emphasize headings and standouts can be useful:

```
h1 {
    text-shadow: 1px 1px 1px #ccc;
}

```

We have a little more time though, so we're going to do something more subtle. We'll add a radial gradient behind the main heading, using an [online gradient editor](#).

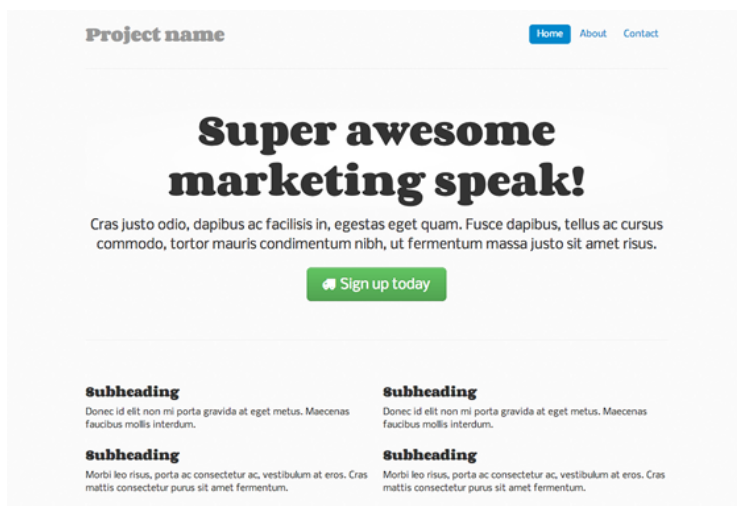
The output is hefty, but you can see it in the CSS. Note that we also need to add the following to our HTML, for IE9 support:



## How to Make Your Site Look Half-Decent in Half an Hour

```
<!--[if gte IE 9]>
  <style type="text/css">
    .gradient {
      filter: none;
    }
  </style>
<![endif]-->
```

And the effect – I don't know what a designer would think, but I like the way it makes the heading pop.



For a crash course in useful modern CSS effects, I highly recommend CodeSchool's online course in **Functional HTML5 and CSS3**. It costs money (\$25 a month to subscribe), but it's worth it for the time you'll save. As a bonus, you also get access to some excellent JavaScript, Ruby and GitHub courses.

(Incidentally, if you find yourself fighting with basic `float` and `display` attributes in CSS – and there’s no shame in it, CSS layout is not intuitive – I recommend the **CSS Cross-Country** course at CodeSchool.)

## 7. ADD A TWIST

We could leave it there, but we’re going to add a background image, and give the site some personality.

This is the area of design that I think many programmers find most intimidating. How do we create the graphics and photographs that a designer would use? The answer is **iStockPhoto** and its competitors – online image libraries where you can find and pay for images. They won’t be unique, but for our purposes, that’s fine.

We’re going to use a **Christmassy image**. For a twist, we’re going to use **Backstretch** to make it responsive.

We must pay for the image, then download it to our `/img/` directory. Then, we set it as our `<body>`’s `background-image`, by including a JavaScript file with just the following line:

```
$.backstretch("/img/winter.jpg");
```

We also reset the subtle pattern to become the background for our container image. It would look much better transparent, so we can use **this technique** in GIMP to make it see-through:

## How to Make Your Site Look Half-Decent in Half an Hour

```
.container-narrow {  
    background: url(/img/cream_dust_transparent.png)  
    repeat 0 0;  
}
```

We also fiddle with the padding on body and .container-narrow a bit, and this is the result:



(Aside: If this were a real site, I'd want to buy images in multiple sizes and ensure that Backstretch chose the most appropriately sized image for our screen, perhaps using responsive images.)

How to find the effects that make a site interesting? I keep a set of bookmarks for interesting JavaScript and CSS effects I might want to use someday, from realistic shadows to animating grids. The *JavaScript Weekly* newsletter is a great source of ideas.

## 8. COLOUR SCHEMES

We're just about getting there – though we're probably past half an hour now – but that button and that menu still both look awfully Bootstrappy.

Real sites, with real designers, have a colour palette, carefully chosen to harmonize and match the brand profile. For our purposes, we're just going to borrow some colours from the image. We use Gimp's colour picker tool to identify the hex values of the blue of the snow. Then we can use **Color Scheme Designer** to find contrasting, but complementary, colours.

Finally, we use those colours for our central button. There are lots of tools to help us do this, such as **Bootstrap Buttons**. The new HTML is quite long, so I won't paste it all here, but you can find it in the CSS file.

We also reset the colour of the pills in the navigation menu, which is a bit easier:

## How to Make Your Site Look Half-Decent in Half an Hour

```
.nav-pills > .active > a, .nav-pills > .active > a:hover  
{  
  background-color: #FF9473;  
}
```

I'm not sure if the result is great to be honest, but at least we've lost those Bootstrap-blue buttons:



Another way to do it, if you didn't have an image to match, would be to borrow an attractive colour scheme.

Colourlovers is a community where people create and share ready-made colour palettes.

The key thing is to find a palette with an open licence, so you can legitimately use it. Unfortunately, you can't search for palettes by licence type, but many do have open licences. Here's a popular palette with a CC-BY-SA licence that allows reuse with attribution.

As above, you can use the hex values from the palette in your custom CSS, and bask in the newly colourful results.

## 9. READ ON

With the above techniques, you can make a site that is starting to look slightly more professional, pretty quickly.

If you have the time to invest, it's well worth learning some design principles, if only so that design seems less intimidating and more like fun. As part of my design learning, I read a few introductory design books aimed at coders. The best I found was David Kadavy's *Design for Hackers: Reverse-Engineering Beauty*, which explains the basic principles behind choosing colours, fonts, typefaces and layout.

In the introduction to his book, David writes:

No group stands to gain more from design literacy than hackers do... The one subject that is exceedingly frustrating for hackers to try to learn is design. Hackers know that in order to compete against corporate behemoths with just a few lines of code, they need to have good, clear design, but the resources with which to learn design are simply hard to find.

Well said. If you have half a day to invest, rather than half an hour, I recommend getting hold of David's book.

And the journey is over. Perhaps that took slightly more than half an hour, but with practice, using the above techniques can become second nature. What useful tools have I missed? Designers, how would you improve on the above? I would love to know, so please give me your views in the comments.

## ABOUT THE AUTHOR



**Anna Powell-Smith** is a freelance web developer, with a background in literature and computer science. She likes Python, JavaScript, data visualisation and online mapping. Currently, she is excited about **CartoDB** and **D3.js**.

Earlier this year, her interactive visualisation of women's clothing sizes, **What Size Am I?**, was featured in the **Guardian**, **Daily Mail**, and the **Mirror**. Another interactive on baby names was featured in the **Guardian** and the **Sun**. Anna created the only freely available online copy of **Domesday Book**, **Open Domesday**.

Anna tweets as **@darkgreener**. You can see her work at <http://anna.ps>.



# 17. Cut Copy Paste

---

Brendan Dawes

[24ways.org/201217](http://24ways.org/201217)

Long before I got into this design thing, I was heavily into making my own music inspired by the likes of Coldcut and Steinski. I would scour local second-hand record shops in search of obscure beats, loops and bits of dialogue in the hope of finding that killer sample I could then splice together with other things to make a huge hit that everyone would love. While it did eventually lead to a record contract and getting to release a few 12" singles, ultimately I knew I'd have to look for something else to pay the bills.

I may not make my own records any more, but the approach I took back then – finding (even stealing) things, cutting and pasting them into interesting combinations – is still at the centre of how I work, only these days it's pretty much bits of code rather than bits of vinyl. Over the

years I've stored these little bits of code (some I've found, some I've created myself) in Evernote, ready to be dialled up whenever I need them.

So when Drew got in touch and asked if I'd like to do something for this year's 24 ways I thought it might be kind of cool to share with you a few of these snippets that I find really useful. Think of these as a kind of coding mix tape; but remember – don't just copy as is: play around, combine and remix them into other wonderful things.

Some of this stuff is dirty; some of it will make hardcore programmers feel ill. For those people, remember this – while you were complaining about the syntax, I made something.

### **Create unique colours**

Let's start right away with something I stole. Well, actually it was given away at the time by **Matt Biddulph** who was then at Dopplr before Nokia destroyed it. Imagine you have thousands of words and you want to assign each one a unique colour. Well, Matt came up with a crazily simple but effective way to do that using an MD5 hash. Just encode said word using an MD5 hash, then take the first six characters of the string you get back to create a hexadecimal colour representation.

I can't guarantee that it will be a harmonious colour palette, but it's still really useful. The thing I love the most about this technique is the left-field thinking of using an encryption system to create colours! Here's an example using JavaScript:

```
// requires the MD5 library available at  
http://pajhome.org.uk/crypt/md5
```

```
function MD5Hex(str){  
    result = MD5.hex(str).substring(0, 6);  
    return result;  
}
```

## Make something breathe using a sine wave

I never paid attention in school, especially during double maths. As a matter of fact, the only time I received corporal punishment – several strokes of the ruler – was in maths class. Anyway, if they had shown me then how beautiful mathematics actually is, I might have paid more attention. Here's a little example of how a sine wave can be used to make something appear to breathe.

I recently used this on an **Arduino** project where an LED ring surrounding a button would gently breathe. Because of that it felt much more inviting. I love mathematics.

```
for(int i = 0; i<360; i++){  
    float rad = DEG_TO_RAD * i;  
    int sinOut = constrain((sin(rad) * 128) + 128, 0, 255);
```

```
    analogWrite(LED, sinOut);  
    delay(10);  
}
```

## Snap position to grid

This is so elegant I love it, and it was shown to me by Gary Burgess, or Boom Boom as myself and others like to call him. It snaps a position, in this case the X-position, to a grid. Just define your grid size (say, twenty pixels) and you're good.

```
snappedXpos = floor( xPos / gridSize) * gridSize;
```

## Calculate the distance between two objects

For me, interaction design is about the relationship between two objects: you and another object; you and another person; or simply one object to another. How close these two things are to each other can be a handy thing to know, allowing you to react to that information within your design. Here's how to calculate the distance between two objects in a 2-D plane:

```
deltaX = round(p2.x-p1.x);  
deltaY = round(p2.y-p1.y);  
diff = round(sqrt((deltaX*deltaX)+(deltaY*deltaY)));
```

## Find the X- and Y-position between two objects

What if you have two objects and you want to place something in-between them? A little bit of interruption and disruption can be a good thing. This small piece of code will allow you to place an object in-between two other objects:

```
// set the position: 0.5 = half-way

float position = 0.5;
float x = x1 + (x2 - x1) *position;
float y = y1 + (y2 - y1) *position;
```

## Distribute objects equally around a circle

More fun with maths, this time adding cosine to our friend sine. Let's say you want to create a circular navigation of arbitrary elements (yeah, **Jakob**, you heard), or you want to place images around a circle. Well, this piece of code will do just that. You can adjust the size of the circle by changing the `distance` variable and alter the number of objects with the `numberOfObjects` variable. Example below is for use in **Processing**.

```
// Example for Processing available for free download at
processing.org

void setup() {

    size(800,800);
    int numberOfObjects = 12;
```

```

int distance = 100;
float inc = (TWO_PI)/numberOfObjects;
float x,y;
float a = 0;

for (int i=0; i < numberOfObjects; i++) {
    x = (width/2) + sin(a)*distance;
    y = (height/2) + cos(a)*distance;
    ellipse(x,y,10,10);
    a += inc;
}
}

```

## Use modulus to make a grid

The modulus operator, represented by %, returns the remainder of a division. Fallen into a coma yet? Hold on a minute – this seemingly simple function is very powerful in lots of ways. At a simple level, you can use it to determine if a number is odd or even, great for creating alternate row colours in a table for instance:

```

boolean checkForEven(numberToCheck) {
    if (numberToCheck % 2 == 0)
        return true;
    } else {
        return false;
    }
}

```

That's all well and good, but here's a use of modulus that might very well blow your mind. Construct a grid with only a few lines of code. Again the example is in **Processing** but can easily be ported to any other language.

```
void setup() {  
  
  size(600,600);  
  int numItems = 120;  
  int numOfColumns = 12;  
  int xSpacing = 40;  
  int ySpacing = 40;  
  int totalWidth = xSpacing*numOfColumns;  
  
  for (int i=0; i < numItems; i++) {  
  
    ellipse(floor((i*xSpacing)%totalWidth), floor((i*xSpacing)/totalWidth)*  
  
  }  
}
```

Not all the bits of code I keep around are for actual graphical output. I also have things that are very utilitarian, but which I still consider part of the design process. Here's a couple of things that I've found really handy lately in my design workflow. They may be a little specific, but I hope they demonstrate that it's not about working harder, it's about working smarter.

## Merge CSV files into one file

Recently, I've had to work with huge – about 1GB – CSV text files that I then needed to combine into one master CSV file so I could then process the data. Opening up each text file and then copying and pasting just seemed really dumb, not to mention slow, so I thought there must be a better way. After some Googling I found this command line script that would combine .txt files into one file and add a new line after each:

```
awk 1 *.txt > finalfile.txt
```

But that wasn't what I was ideally after. I wanted to merge the CSV files, keeping the first row of the first file (the column headings) and then ignore the first row of subsequent files. Sure enough I found the answer after some Googling and it worked like a charm. Apologies to the original author but I can't remember where I found it, but you, sir or madam, are awesome. Save this as a shell script:

```
FIRST=
```

```
for FILE in *.csv
do
    exec 5<"$FILE" # Open file
    read LINE <&5 # Read first line
    [ -z "$FIRST" ] && echo "$LINE" # Print it only
from first file
    FIRST="no"
```



```
cat <&5 # Print the rest directly to standard
output
exec 5<&- # Close file
# Redirect stdout for this section into file.out

done > file.out
```

## Create a symbolic link to another file or folder

Oftentimes, I'll find myself hunting through a load of directories to load a file to be processed, like a CSV file. Use a symbolic link (in the Terminal) to place a link on your desktop or wherever is most convenient and it'll save you loads of time. Especially great if you're going through a Java file dialogue box in Processing or something that doesn't allow the normal Mac dialog box or aliases.

```
cd /DirectoryYouWantShortcutToLiveIn
ln -s /Directory/You/Want/ShortcutTo/ TheShortcut
```

## You can do it, in the mix

I hope you've found some of the above useful and that they've inspired a few ideas here and there. Feel free to tell me better ways of doing things or offer up any other handy pieces of code. Most of all though, collect, remix and combine the things you discover to make lovely new things.

## ABOUT THE AUTHOR



Ever since his first experiences with the humble ZX81 back in the early eighties, **Brendan Dawes** has continued to explore the interplay of people, code, design and art through his work on [brendandawes.com](http://brendandawes.com) where he publishes ideas, toys and projects created from an eclectic mix of digital and analog objects. On top of all that his Mum says he's good with computers.

# 18. Giving Content Priority with CSS3 Grid Layout

---

Rachel Andrew

[24ways.org/201218](http://24ways.org/201218)

Browser support for many of the modules that are part of CSS3 have enabled us to use CSS for many of the things we used to have to use images for. The rise of mobile browsers and the concept of responsive web design has given us a whole new way of looking at design for the web. However, when it comes to layout, we haven't moved very far at all. We have talked for years about separating our content and source order from the presentation of that content, yet most of us have had to make decisions on source order in order to get a certain visual layout.

Owing to some interesting specifications making their way through the W3C process at the moment, though, there is hope of change on the horizon. In this article I'm going to look at one CSS module, the **CSS3 grid layout module**, that enables us to define a grid and place elements on to it. This article comprises a practical demonstration of the basics of grid layout, and also a discussion of one way in which we can start thinking of content in a more adaptive way.

Before we get started, it is important to note that, at the time of writing, **these examples work only in Internet Explorer 10**. CSS3 grid layout is a module created by Microsoft, and implemented using the `-ms` prefix in IE10. My examples will all use the `-ms` prefix, and not include other prefixes simply because this is such an early stage specification, and by the time there are implementations in other browsers there may be inconsistencies. The implementation I describe today may well change, but is also there for your feedback.

If you don't have access to IE10, then one way to view and test these examples is by signing up for an account with **Browserstack** – the free trial would give you time to have a look. I have also included screenshots of all relevant stages in creating the examples.

## WHAT IS CSS3 GRID LAYOUT?

CSS3 grid layout aims to let developers divide up a design into a grid and place content on to that grid. Rather than trying to fabricate a grid from floats, you can declare an actual grid on a container element and then use that to position the elements inside. Most importantly, the source order of those elements does not matter.

### Declaring a grid

We declare a grid using a new value for the display property: `display: grid`. As we are using the IE10 implementation here, we need to prefix that value: `display: -ms-grid`;

Once we have declared our grid, we set up the columns and rows using the `grid-columns` and `grid-rows` properties.

```
.wrapper {  
    display: -ms-grid;  
    -ms-grid-columns: 200px 20px auto 20px 200px;  
    -ms-grid-rows: auto 1fr;  
}
```

In the above example, I have declared a grid on the `.wrapper` element. I have used the `grid-columns` property to create a grid with a 200 pixel-wide column, a 20 pixel gutter, a flexible width `auto` column that will stretch to fill the available space, another 20 pixel-wide gutter and a

final 200 pixel sidebar: a flexible width layout with two fixed width sidebars. Using the `grid-rows` property I have created two rows: the first is set to `auto` so it will stretch to fill whatever I put into it; the second row is set to `1fr`, a new value used in grids that means one *fraction unit*. In this case, one fraction unit of the available space, effectively whatever space is left.

## Positioning items on the grid

Now I have a simple grid, I can pop items on to it. If I have a `<div>` with a class of `.main` that I want to place into the second row, and the flexible column set to `auto` I would use the following CSS:

```
.content {  
  -ms-grid-column: 3;  
  -ms-grid-row: 2;  
  -ms-grid-row-span: 1;  
}
```

If you are old-school, you may already have realised that we are essentially creating an HTML table-like layout structure using CSS. I found the concept of a table the most helpful way to think about the grid layout module when trying to work out how to place elements.

## CREATING GRID SYSTEMS

As soon as I started to play with CSS3 grid layout, I wanted to see if I could use it to replicate a flexible grid system like this **fluid 16-column 960 grid system**.

I started out by defining a grid on my wrapper element, using fractions to make this grid fluid.

```
.wrapper {
  width: 90%;
  margin: 0 auto 0 auto;
  display: -ms-grid;
  -ms-grid-columns: 1fr (4.25fr 1fr)[16];
  -ms-grid-rows: (auto 20px)[24];
}
```

Like the 960 grid system I was using as an example, my grid starts with a gutter, followed by the first actual column, plus another gutter repeated sixteen times. What this means is that if I want to span two columns, as far as the grid layout module is concerned that is actually three columns: two wide columns, plus one gutter. So this needs to be accounted for when positioning items.

I created a CSS class for each positioning option: column position; rows position; and column span. For example:

```
.grid1 {-ms-grid-column: 2;} /* applying this class
positions an item in the first column (the gutter is
column 1) */
.grid2 {-ms-grid-column: 4;} /* 2nd column -
```

```
gutter|column 1|gutter */
.grid3 {-ms-grid-column: 6;} /* 3rd column -
gutter|column 1|gutter|column2|gutter */

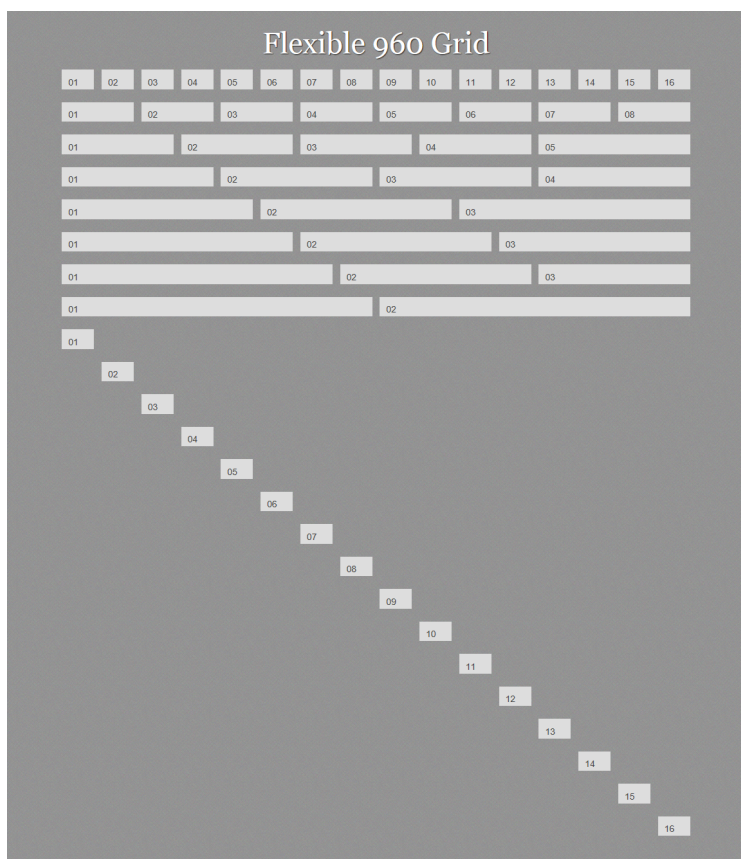
.row1 {-ms-grid-row:1;}
.row2 {-ms-grid-row:3;}
.row3 {-ms-grid-row:5;}

.colspan1 {-ms-grid-column-span:1;}
.colspan2 {-ms-grid-column-span:3;}
.colspan3 {-ms-grid-column-span:5;}
```

I could then add multiple classes to each element to set the position on on the grid.



## Giving Content Priority with CSS3 Grid Layout



This then gives me a replica of the fluid grid using CSS3 grid layout. To see this working fire up IE10 and view [Example 1](#).

## THIS WORKS, BUT...

This worked, but isn't ideal. I considered not showing this stage of my experiment – however, I think it clearly shows how the grid layout module works and is a useful starting point. That said, it's not an approach I would take in production. First, we have to add classes to our markup that tie an element to a position on the grid. This might not be too much of a problem if we are always going to maintain the sixteen-column grid, though, as I will show you that the real power of the grid layout module appears once you start to redefine the grid, using different grids based on media queries. If you drop to a six-column layout for small screens, positioning items into column 16 makes no sense any more.

## CALCULATING GRID POSITION USING LESS

As we've seen, if you want to use a grid with main columns and gutters, you have to take into account the spacing between columns as well as the actual columns. This means we have to do some calculating every time we place an item on the grid. In my example above I got around this by creating a CSS class for each position, allowing me to think in sixteen rather than thirty-two columns. But by using a CSS preprocessor, I can avoid using all the classes yet still think in main columns.

I'm using LESS for my example. My simple grid framework consists of one simple mixin.

```
.position(@column,@row,@colspan,@rowspan) {  
  -ms-grid-column: @column*2;  
  -ms-grid-row: @row*2-1;  
  -ms-grid-column-span: @colspan*2-1;  
  -ms-grid-row-span: @rowspan*2-1;  
}
```

My mixin takes four parameters: column; row; colspan; and rowspan. So if I wanted to place an item on column four, row three, spanning two columns and one row, I would write the following CSS:

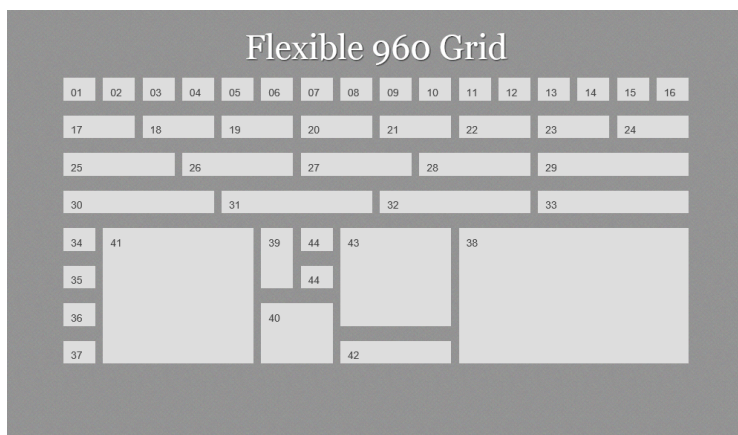
```
.box {  
  .position(4,3,2,1);  
}
```

The mixin would return:

```
.box {  
  -ms-grid-column: 8;  
  -ms-grid-row: 5;  
  -ms-grid-column-span: 3;  
  -ms-grid-row-span: 1;  
}
```

This saves me some typing and some maths. I could also add other prefixed values into my mixin as other browsers started to add support.

We can see this in action creating a new grid. Instead of adding multiple classes to each element, I can add one class; that class uses the mixin to create the position. I have also played around with row spans using my mixin and you can see we end up with a quite complicated arrangement of boxes. **Have a look at example two in IE10.** I've used the JavaScript LESS parser so that you can view the actual LESS that I use. Note that I have needed to escape the `-ms` prefixed properties with `~""` to get LESS to accept them.



This is looking better. I don't have direct positioning information on each element in the markup, just a class name – I've used `grid(x)`, but it could be something far more semantic. We can now take the example a step further and redefine the grid based on screen width.

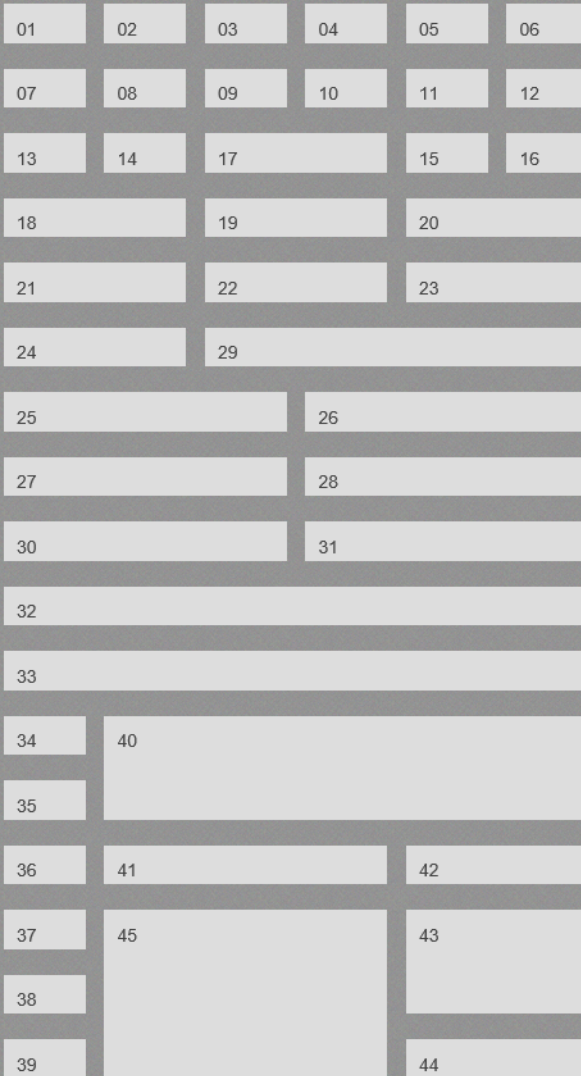
## MEDIA QUERIES AND THE GRID

This example uses exactly the same markup as the previous example. However, we are now using media queries to detect screen width and redefine the grid using a different number of columns depending on that width.

I start out with a six-column grid, defining that on `.wrapper`, then setting where the different items sit on this grid:

```
.wrapper {
  width: 90%;
  margin: 0 auto 0 auto;
  display: ~"-ms-grid"; /* escaped for the LESS parser
*/
  -ms-grid-columns: ~"1fr (4.25fr 1fr)[6]"; /* escaped
for the LESS parser */
  -ms-grid-rows: ~"(auto 20px)[40]"; /* escaped for
the LESS parser */
}
.grid1 { .position(1,1,1,1); }
.grid2 { .position(2,1,1,1); }
/* ... see example for all declarations ... */
```

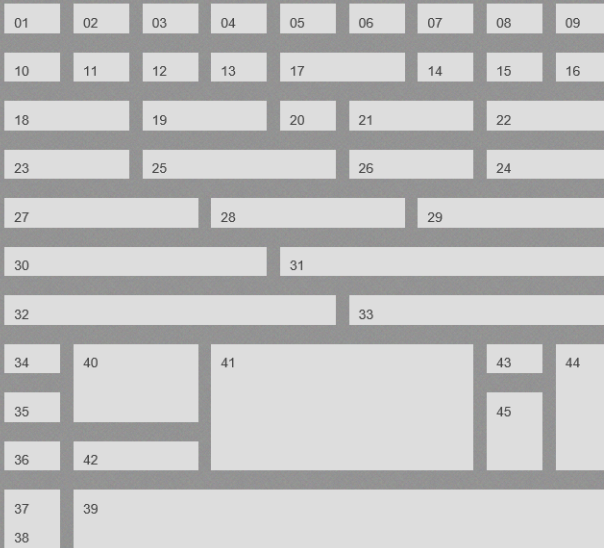
# Responsive Grid



Using media queries, I redefine the grid to nine columns when we hit a minimum width of 700 pixels.

```
@media only screen and (min-width: 700px) {  
  .wrapper {  
    -ms-grid-columns: ~"1fr (4.25fr 1fr)[9]";  
    -ms-grid-rows: ~"(auto 20px)[50]";  
  }  
  .grid1 { .position(1,1,1,1); }  
  .grid2 { .position(2,1,1,1); }  
  /* ... */  
}
```

## Responsive Grid



Finally, we redefine the grid for 960 pixels, back to the sixteen-column grid we started out with.

```
@media only screen and (min-width: 940px) {  
  .wrapper {  
    -ms-grid-columns:~" 1fr (4.25fr 1fr)[16]";  
    -ms-grid-rows:~" (auto 20px)[24]";  
  }  
  .grid1 { .position(1,1,1,1); }
```



```
.grid2 { .position(2,1,1,1); }  
/* ... */  
}
```

If you view example three in Internet Explorer 10 you can see how the items reflow to fit the window size. You can also see, looking at the final set of blocks, that source order doesn't matter. You can pick up a block from anywhere and place it in any position on the grid.

### LAYING OUT A SIMPLE WEBSITE

So far, like a toddler on Christmas Day, we've been playing with boxes rather than thinking about what might be in them. So let's take a quick look at a more realistic layout, in order to see why the CSS3 grid layout module can be really useful. At this time of year, I am very excited to get out of storage my collection of odd nativity sets, prompting my family to suggest I might want to open a museum. Should I ever do so, I'll need a website, and here is an example layout.

## Welcome!

If you have a curious obsession with Nativity sets, then you are in the right place.

Our collection contains more crazy looking Nativity sets than you could shake a tinsel covered stick at.

## About Nativity Sets

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed euismod lacinia sodales. Etiam lobortis neque id nunc semper auctor. Nam sagittis, enim sed dignissim faucibus, neque magna fermentum neque, venenatis ullamcorper risus dolor at metus. Curabitur in neque est. Integer fringilla gravida dolor, a consequat erat blandit sed. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc eleifend leo ut lectus malesuada elementum. Donec sed neque nisi.

## History of the Museum

Morbi tincidunt odio in mi mattis faucibus vel ac turpis. Quisque feugiat nibh quis sapien bibendum eleifend. Phasellus at molestie arcu. Nunc at urna condimentum nisi ultricies rhoncus. Aliquam feugiat arcu vel risus bibendum at venenatis nunc mollis. Mauris aliquet dictum leo, sed tristique sapien luctus eu. Praesent ultricies nulla eget turpis tincidunt suscipit.

## The Museum Today

Nullam iaculis, magna a dignissim dictum, quam diam porta eros, sed luctus purus velit in eros. Morbi ullamcorper nunc eu lorem placerat lobortis. Sed condimentum sagittis nunc id imperdiet. Etiam pharetra sodales lorem, et tempus lectus vestibulum elementum. In viverra tempor magna non laoreet. Proin a augue eros, sit amet malesuada odio. Cras tellus tortor, laoreet id laoreet sit amet, volutpat accumsan metus. Nulla dapibus ante ut nisi vulputate ultrices. In hac habitasse platea dictumst. Aenean lectus massa, congue in malesuada eget, mollis vitae urna. Aenean neque nunc, laoreet id convallis a, interdum et arcu. Etiam aliquam euismod sem, et feugiat nunc lobortis id.

## Visit Us

Opening times: 9am - 5pm throughout Advent.

**The Nativity Museum**  
1 Christmas Street, Tinsel Town, United Kingdom



As I am using CSS3 grid layout, I can order my source in a logical manner. In this example my document is as follows, though these elements could be in any order I please:

```
<div class="wrapper">
  <div class="welcome">
    ...
  </div>
  <article class="main">
    ...
  </article>
  <div class="info">
```

## Giving Content Priority with CSS3 Grid Layout

```
...
</div>
<div class="ads">
...
</div>
</div>
```

For wide viewports I can use grid layout to create a sidebar, with the important information about opening times on the top right, with the ads displayed below it. This creates the layout shown in the screenshot above.

```
@media only screen and (min-width: 940px) {
  .wrapper {
    -ms-grid-columns:~" 1fr (4.25fr 1fr)[16]";
    -ms-grid-rows:~" (auto 20px)[24]";
  }
  .welcome {
    .position(1,1,12,1);
    padding: 0 5% 0 0;
  }
  .info {
    .position(13,1,4,1);
    border: 0;
    padding:0;
  }
  .main {
    .position(1,2,12,1);
    padding: 0 5% 0 0;
  }
  .ads {
    .position(13,2,4,1);
    display: block;
  }
}
```

```
    margin-left: 0;
  }
}
```

In a floated layout, a sidebar like this often ends up being placed under the main content at smaller screen widths. For my situation this is less than ideal. I want the important information about opening times to end up above the main article, and to push the ads below it. With grid layout I can easily achieve this at the smallest width .info ends up in row two and .ads in row five with the article between.

```
.wrapper {
  display: ~"-ms-grid";
  -ms-grid-columns: ~"1fr (4.25fr 1fr)[4]";
  -ms-grid-rows: ~"(auto 20px)[40]";
}
.welcome {
  .position(1,1,4,1);
}
.info {
  .position(1,2,4,1);
  border: 4px solid #fff;
  padding: 10px;
}
.content {
  .position(1,3,4,5);
}
.main {
  .position(1,3,4,1);
}
```

## Giving Content Priority with CSS3 Grid Layout

```
.ads {  
    .position(1,4,4,1);  
}
```

# Welcome!

If you have a curious obsession with Nativity sets, then you are in the right place.

Our collection contains more crazy looking Nativity sets than you could shake a tinsel covered stick at.

## Visit Us

Opening times: 9am - 5pm throughout Advent.

### **The Nativity Museum**

1 Christmas Street, Tinsel Town, United Kingdom

## About Nativity Sets

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed euismod lacinia sodales. Etiam lobortis neque id nunc semper auctor. Nam sagittis, enim sed dignissim faucibus, neque magna fermentum neque, venenatis ullamcorper risus dolor at metus. Curabitur in neque est. Integer fringilla gravida dolor, a consequat erat blandit sed. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc eleifend leo ut lectus malesuada elementum. Donec sed neque nisi.

## History of the Museum

Morbi tincidunt odio in mi mattis faucibus vel ac turpis. Quisque feugiat nibh quis sapien24ways 2012 edition eleifend. Phasellus at molestie arcu. Nunc at urna condimentum nisi ultricies rhoncus. Aliquam feugiat arcu vel risus bibendum at venenatis nunc mollis.

Finally, as an extra tweak I add in a breakpoint at 600 pixels and nest a second grid on the ads area, arranging those three images into a row when they sit below the article at a screen width wider than the very narrow mobile width but still too narrow to support a sidebar.

```
@media only screen and (min-width: 600px) {
  .ads {
    display: ~"-ms-grid";
    -ms-grid-columns: ~"20px 1fr 20px 1fr 20px 1fr";
    -ms-grid-rows: ~"1fr";
    margin-left: -20px;
  }
  .ad:nth-child(1) {
    .position(1,1,1,1);
  }
  .ad:nth-child(2) {
    .position(2,1,1,1);
  }
  .ad:nth-child(3) {
    .position(3,1,1,1);
  }
}
```

**View example four in Internet Explorer 10.**

per congue in nunc, per inceptos himenaeos. Nunc etiam leo ut lectus mauesuada diciturum. Donec sed neque nisi.

## History of the Museum

Morbi tincidunt odio in mi mattis faucibus vel ac turpis. Quisque feugiat nibh quis sapien bibendum eleifend. Phasellus at molestie arcu. Nunc at urna condimentum nisi ultricies rhoncus. Aliquam feugiat arcu vel risus bibendum at venenatis nunc mollis. Mauris aliquet dictum leo, sed tristique sapien luctus eu. Praesent ultricies nulla eget turpis tincidunt suscipit.

## The Museum Today

Nullam iaculis, magna a dignissim dictum, quam diam porta eros, sed luctus purus velit in eros. Morbi ullamcorper nunc eu lorem placerat lobortis. Sed condimentum sagittis nunc id imperdiet. Etiam pharetra sodales lorem, et tempus lectus vestibulum elementum. In viverra tempor magna non laoreet. Proin a augue eros, sit amet malesuada odio. Cras tellus tortor, laoreet id laoreet sit amet, volutpat accumsan metus. Nulla dapibus ante ut nisi vulputate ultrices. In hac habitasse platea dictumst. Aenean lectus massa, congue in malesuada eget, mollis vitae urna. Aenean neque nunc, laoreet id convallis a, interdum et arcu. Etiam aliquam euismod sem, et feugiat nunc lobortis id.



This is a very simple example to show how we can use CSS grid layout without needing to add a lot of classes to our document. It also demonstrates how we can manipulate the content depending on the context in which the user is viewing it.

## LAYOUT, SOURCE ORDER AND THE IDEA OF CONTENT PRIORITY

CSS3 grid layout isn't the only module that starts to move us away from the issue of visual layout being linked to source order. However, with good support in Internet Explorer 10, it is a nice way to start looking at how this might work. If you look at the grid layout module as



something to be used in conjunction with the **flexible box layout module** and the very interesting **CSS regions** and **exclusions** specifications, we have, tantalizingly on the horizon, a powerful set of tools for layout.

I am particularly keen on the potential separation of source order from layout as it dovetails rather neatly into something I spend a lot of time thinking about. As a CMS developer, working on larger scale projects as well as our CMS product **Perch**, I am interested in how we better enable content editors to create content for the web. In particular, I search for better ways to help them create adaptive content; content that will work in a variety of contexts rather than being tied to one representation of that content.

If the concept of adaptive content is new to you, then Karen McGrane's presentation **Adapting Ourselves to Adaptive Content** is the place to start. Karen talks about needing to think of content as chunks, that might be used in many different places, displayed differently depending on context.

I absolutely agree with Karen's approach to content. We have always attempted to move content editors away from thinking about creating a page and previewing it on the desktop. However at some point content does need to be published as a page, or a collection of content if you prefer, and bits of that content have priority. Particularly

in a small screen context, content gets linearized, we can only show so much at a time, and we need to make sure important content rises to the top. In the case of my example, I wanted to ensure that the address information was clearly visible without scrolling around too much. Dropping it with the entire sidebar to the bottom of the page would not have been so helpful, though neither would moving the whole sidebar to the top of the screen so a visitor had to scroll past advertising to get to the article.

If our layout is linked to our source order, then enabling the content editor to make decisions about priority is really hard. Only a system that can do some regeneration of the source order on the server-side – perhaps by way of multiple templates – can allow those kinds of decisions to be made. For larger systems this might be a possibility; for smaller ones, or when using an off-the-shelf CMS, it is less likely to be. Fortunately, any system that allows some form of custom field type can be used to pop a class on to an element, and with CSS grid layout that is all that is needed to be able to target that element and drop it into the right place when the content is viewed, be that on a desktop or a mobile device.

This approach can move us away from forcing editors to think visually. At the moment, I might have to explain to an editor that if a certain piece of content needs to come first when viewed on a mobile device, it needs to be placed in

the sidebar area, tying it to a particular layout and design. I have to do this because we have to enforce fairly strict rules around source order to make the mechanics of the responsive design work. If I can instead advise an editor to flag important content as high priority in the CMS, then I can make decisions elsewhere as to how that is displayed, and we can maintain the visual hierarchy across all the different ways content might be rendered.

## **WHY FRUSTRATE OURSELVES WITH SPECIFICATIONS WE CAN'T YET USE IN PRODUCTION?**

The CSS3 grid layout specification is listed under the Exploring section of the list of **current work of the CSS Working Group**. While discussing a module at this stage might seem a bit pointless if we can't use it in production work, there is a very real reason for doing so. If those of us who will ultimately be developing sites with these tools find out about them early enough, then we can start to give our feedback to the people responsible for the specification. There is information on the same page about **how to get involved with the discussions**.

So, if you have a bit of time this holiday season, why not have a play with the CSS3 grid layout module? I have outlined here some of my thoughts on how grid layout and other modules that separate layout from source order can be used in the work that I do. Likewise, wherever in the

stack you work, playing with and thinking about new specifications means you can think about how you would use them to enhance your work. Spot a problem? Think that a change to the specification would improve things for a specific use case? Then you have something you could post to [www-style](http://www-style) to add to the discussion around this module.

All the examples are on [CodePen](https://codepen.io) so feel free to play around and fork them.

## ABOUT THE AUTHOR



**Rachel Andrew** is a Director of [edgeofmyseat.com](http://edgeofmyseat.com), a UK web development consultancy and creators of the small content management system, **Perch**. She is the author of a number of

books, most recently *The Profitable Side Project Handbook* and *CSS3 Layout Modules*, and is a regular columnist for *A List Apart*.

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

# 19. Direction, Distance and Destinations

---

Brian Suda

24ways.org/201219

With all these new smartphones in the hands of lost and confused owners, we need a better way to represent distances and directions to destinations. The immediate examples that jump to mind are augmented reality apps which let you see another world through your phone's camera. While this is interesting, there is a simpler way: letting people know how far away they are and if they are getting warmer or colder.

In the app world, you can easily tap into the phone's array of sensors such as the GPS and compass, but what people rarely know is that you can do the same with HTML. The native versus web app debate will never subside, but at least we can show you how to replicate some of the functionality progressively in HTML and JavaScript.

In this tutorial, we'll walk through how to create a simple webpage listing distances and directions of a few popular locations around the world. We'll use JavaScript to access

the device's geolocation API and also attempt to access the compass to get a heading. Both of these APIs are documented, to be included in the W3C geolocation API specification, and can be used on both desktop and mobile devices today.

To get started, we need a list of a few locations around the world. I have chosen the highest mountain peak on each continent so you can see a diverse set of distances and directions.

Mountain	°Latitude	°Longitude
Kilimanjaro	-3.075833	37.353333
Vinson Massif	-78.525483	-85.617147
Puncak Jaya	-4.078889	137.158333
Everest	27.988056	86.925278
Elbrus	43.355	42.439167
Mount McKinley	63.0695	-151.0074
Aconcagua	-32.653431	-70.011083

Source: [Wikipedia](#)

We can put those into an HTML list to be styled and accessed by JavaScript to create some distance and directions calculations.

The next thing we need to do is check to see if the browser and operating system have geolocation support. To do this we test to see if the function is available or not using a single JavaScript `if` statement.

```
<script>
// If this is true, then the method is supported and we
can try to access the location
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(geo_success,
geo_error);
}
</script>
```

The `if` statement will be false if geolocation support is not present, and then it is up to you to do something else instead as a fallback. For this example, we'll do nothing since our page should work as is and only get progressively better if more functionality is available.

The `if` statement will be true if there is support and therefore will continue inside the curly brackets to try to get the location. This should prompt the reader to accept or deny the request to get their location. If they say no, the second function callback is processed, in this case a function called `geo_error`; whereas if the location is available, it fires the `geo_success` function callback.

The function `geo_error(){ }` isn't that exciting. You can handle this in any way you see fit. The success function is more interesting. We get a position object passed into the



function which contains a series of exciting attributes, namely the latitude and longitude of the device's current location.

```
function geo_success(position){
  gLat = position.coords.latitude;
  gLon = position.coords.longitude;
}
```

Now, in the variables `gLat` and `gLon` we have the user's approximate geographical position. We can use this information to start to calculate some distances between where they are and all the destinations.

At the time of writing, you can also get `position.coords.heading`, but on Windows and iOS devices this returned `NULL`. In the future, if and when this is supported, this is also where you can easily grab the compass information.

Inside the `geo_success` function, we want to loop through the HTML to get all of the mountain peaks' latitudes and longitudes and compute the distance.

```
...
$('.geo').each(function(){
  // Get the lat/lon from the HTML
  tLat = $(this).find('.lat').html()
  tLon = $(this).find('.lon').html()

  // compute the distances between the current location
  and this points location
```

```

dist = distance(tLat,tLon,gLat,gLon);

// set the return values into something useful
d = parseInt(dist[0]*10)/10;
a = parseFloat(dist[1]);

// display the value in the HTML and style the arrow
$(this).find('.distance').html(d+' km away');

$(this).find('.direction').css('-webkit-transform','rotate(-'+
+ a + 'deg)');

// store the arc for later use if compass is available
$(this).attr('data-arc',a);
}

```

In the variable `d` we have the distance between the current location and the location of the mountain peak based on the **Haversine Formula**. The variable `a` is the arc, which has a value from 0 to 359.99. This will be useful later if we have compass support. Given these two values we have a distance and a heading to style the HTML.

The next thing we want to do is check to see if the device has a compass and then get access to the the current heading. As we'll see, there are several ways to do this, some of which work on certain devices but not others. The W3C geolocation spec says that, along with the coordinates, there are several other attributes: accuracy; altitude; and heading. Heading is the direction to true north, which is different than magnetic north! WebKit

and Windows return NULL for the heading value, but WebKit has an experimental method to fetch the heading. If you get into accessing these sensors, you'll have to try to catch a few of these methods to finally get a value. Assuming you do, we can move on to the more interesting display opportunities.

In an ideal world, this would succeed and set a variable we'll call `compassHeading` to get a value between 0 and 359.99 degrees. Now we know which direction north is, we also know the direction relative to north of the path to our destination, so we can subtract the two values to get an arrow to display on the screen. But we're not finished yet: we also need to get the device's orientation (landscape or portrait) and subtract the correct amount from the angle for the arrow. Once we have a value, we can use CSS to rotate the arrow the correct number of degrees.

```
-webkit-transform: rotate(-180deg)
```

Not all devices support a standard way to access compass information, so in the meantime we need to use a work around. On iOS, you can use the experimental event method `e.webkitCompassHeading`. We want the compass to update in real time as the device is moved around, so we'll put this inside an event listener.

```

window.addEventListener('deviceorientation', function(e)
{
    // Loop through all the locations on the page
    $(' .geo').each(function(){
        // get the arc value from north we computed and
        stored earlier
        destination_arc = parseInt($(this).attr('data-arc'))
        compassHeading = e.webkitCompassHeading +
window.orientation + destination_arc;
        // find the arrow element and rotate it accordingly

$(this).find('.direction').css('-webkit-transform', 'rotate(-'
+ compassHeading + 'deg)');
    }
}

```

As the device is rotated, the compass arrow will constantly be updated. If you want to see an example, you can **have a look at this page** which shows the distances to all the peaks on each continent.

With progressive enhancement, we slowly layer on additional functionality as we go. The reader will first see the list of locations with a latitude and longitude. If the device is capable and permissions allow, it will then compute the distance. If a compass is available, with the correct permissions it will then add the final layer which is direction.

You should consider this code a stub for your projects. If you are making a hyperlocal webpage with restaurant locations, for example, then consider adding these

features. Knowing not only how far away a place is, but also the direction can be hugely important, and since the compass is always active, it acts as a guide to the location.

## **FUTURE DEVELOPMENTS**

Improvements to this could include setting a timer and recalling the `navigator.geolocation.getCurrentPosition()` function and updating the distances. I chose very distant mountains so kilometres made sense, but you can divide again by 1,000 to convert to metres if you are dealing with much nearer places. Walking or driving would change the distances so the ability to refresh would be important.

It is outside the scope of this article, but if you manage to get this HTML to work offline, then you can make a nice web app which sits on your devices' homescreens and works even without an internet connection. This could be ideal for travellers in an unknown city looking for your destination. Just with offline storage, base64 encoding and data URIs, it is possible to embed plenty of design and functionality into a small offline webpage.

Now you know how to use JavaScript to look up a destination's location and figure out the distance and direction – never get lost again.

## ABOUT THE AUTHOR



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

Photo: Jeremy Keith

## 20. Content Planning Demystified

---

Erin Kissane

[24ways.org/201220](http://24ways.org/201220)

The first thing you learn as a junior editor is that you can't do everything yourself. You must rely on someone else to do at least part of what must be done: the long-range planning, the initial drafting or shooting or recording, the editing, the production, the final polish. All of those pieces of work that belong to someone else take quite a lot of time — days, weeks, sometimes months. If you're the sort of person who wrote college term papers the night before they were due, this can come as a bit of a shock. To my twenty-two-year-old self, it certainly did.

It turns out that the only real way to avoid a trainwreck with editorial work is to get ahead of the trouble, line everything up carefully, and leave oodles of room for all the pieces to connect on time. The same is true of content strategy, content planning, and just about everything to

do with content on the web, except for the writing itself — and that, too, usually takes far longer than anyone expects. If you're not a professional editor and you suddenly find yourself dealing with content creation, you're almost certainly going to underestimate the time and effort involved, or to skip something important in the planning process that pops up to bite you later.

Without good content, it doesn't matter how well designed or coded your web project is, because it won't be doing the thing it's meant to do. And even if content is far from your specialty, you may well end up being the only one willing to coordinate it far enough in advance to avoid a chaotic ending. Whether you're hiring writers and editors for a big project, working with a small client, or coaxing some editorial help out of a co-worker, getting the planning work done correctly — and ahead of time — will allow you to orchestrate a glorious ballet of togetherness, instead of feverishly scraping together something to put on your site when the deadline looms. So get out the graph paper and the pocket protector, because we're going to go Full Nerd on this problem.

## **KNOW YOUR POISON**

Anyone who's seen a project delayed for six months by content trouble, or derailed by content that's bland and unhelpful, knows this stuff can make you feel like a dead sock. To get ahead of the problem, you're going to have to



learn to spot common problems and plan your way around them. On web projects without a dedicated editorial lead, you're likely to encounter content that is:

- **Useless** – Content that doesn't serve your readers' needs in some way is pointless. And because it takes up your time and crowds out genuinely helpful things, it's actually damaging. The logic is simple: you can make content that's all about you, and that serves your stated messaging goals, but if no one is motivated to read it, it's a waste of everyone's time.
- **Badly written** – When you publish articles or instructions or other content that is too stiffly formal, overly wordy, hard to understand, offensive, unintentionally cheesy, or otherwise off in tone or style, you're doing two things. First, you're weakening the information you're trying to convey by making it obscure or annoying. Second – and this one is even more damaging – you're demonstrating bad taste. When you get the cultural elements of publishing wrong, you encourage your readers to believe that you either don't understand them or don't care about getting it wrong.
- **Goopy** – Content strategists have been talking about structured content (that's **chunks versus blobs**) for years. If you're publishing more than a few dozen pages without thinking through the structure of your content, you're probably missing a chance to improve your long-term efficiency. If you're publishing more than a couple of

thousand pages without taking care of your content structure, you're probably doing a lot more manual wrangling (or cumbersome CMS work) than you need to be, especially when it comes to cross-platform publishing.

- **Unregulated** – If you're not tracking what works and what doesn't — and especially if you don't know what “works” means for your project or organization — you're almost certainly getting worse results than you should be, for more work.
- **Overabundant** – As demonstrated by the cinnamon challenge, too much of a delicious thing can be a giant and publicly embarrassing disaster. For most projects and organizations, if you're making more stuff than your readers can handle, or if you're spreading your creative and editorial resources too thinly, that's bad. Spammers, content farms, and barrel-bottom tabloids have their own special math, the side effects of which include insomnia, irritability, and crying in traffic while silently mouthing Wilson Phillips lyrics.

## **PREVENT ALL PREVENTABLE DAMAGE**

Once you know what kind of trouble to look for, you can prevent a lot of it by doing some smart planning well before someone starts writing (or recording or shooting video).

- **To prevent uselessness:** Know your readers and decide what you're trying to accomplish — with your website as a whole, and with each piece of content, always. Once you know what you're trying to achieve, you can evaluate your work as you go to make sure that it's actually doing the right thing. (I've written a lot more about this for *A List Apart* and in *The Elements of Content Strategy*.)
- **To prevent bad writing:** Establish a consistent and appropriate style using examples (and a style guide if you need one), designate an editor, hire good writers, and make time for quality control. Kate Kiefer's style guide for MailChimp is a superb example of style-wrangling that everyone can use.
- **To prevent repulsive goo:** Give your content as much structure as possible, and know how structure relates to your entire publishing ecosystem, including all those mobile devices. Sara Wachter-Boettcher's *Content Everywhere* and Karen McGrane's *Content Strategy for Mobile* offer brilliant yet friendly introductions to the wide world of structured content.
- **To prevent unregulated chaos:** Measure everything that matters to your project, your client, your organization, and especially your readers — not generic measures of someone else's success. Measure it all regularly. Be disciplined. Adjust at regular intervals. Rick Allen's series on content strategy analytics is an excellent place to begin ([part one](#); [part two](#)).

- **To prevent overabundance:** Stop trying to do everything and focus on giving your readers just a few things they want and genuinely need. Don't establish a schedule your writers might not be able to keep, and focus on differentiating yourself with quality, not quantity. (And while you're at it, scratch the auto-posting to social networks and the cross-posting between them. It's about as engaging as an automated phone system.)

At a slightly higher level, pick the right content person (or team) for the work. If you really only need a few pages of copy, find a smart writer who does good work for multi-platform readers. If you're slinging tens of thousands of pages of content, get someone with field experience in high-level editorial planning and the ability to turn blobs into chunks and melted goo into Legos. If you're starting a project that involves making a lot of content over time, bring in someone with journalism experience (or get your client to do so).

"But wait!" you may say. "I'm not hiring anyone. I have to do this all myself." That's not uncommon at all. The bad news is, you have to learn a bunch of stuff. The good news is, you get to learn a bunch of awesome stuff. Figure out what the project needs, just as though you were going to hire someone, and then give yourself time to get up to speed. If it's a really complicated project, you're probably going to have trouble unless you eventually get

professional help. But if it's small and you can do it in steps, you can certainly do much better by giving yourself a plan and working on the things that matter most.

## **PLAN FOR THE MARATHON, NOT THE SPRINT**

Launching with awesome content is a tiny fraction of a victory, which is why it's so important that your content not be gooeey or unregulated. It also means that if you don't plan for a realistic publication schedule, you are going to slam into reality in a really unpleasant way not too long after you've begun. If you're asking people to make words (or videos or whatever) for you, they're going to have to do less of something else, so plan for that beforehand.

And while you're at it, unless publishing is your core business, ditch the feed-the-beast plan that leads to fluffy blog posts and spiritless, unhelpful social media content. It's antisocial for your reading community, offers short-term gains at best, and will burn you out or lower your standards until you don't even know you're doing lousy work. Good content is expensive, no matter how you do it, but spreading yourself too thin is a much worse investment than doing a smaller thing well and gradually building up a body of superb content that people want to share and keep and return to.

## ABOUT THE AUTHOR



**Erin Kissane** edits magazines, websites, does content strategy for institutions and companies, and reads a lot. She currently edits *Contents* magazine and Knight-Mozilla OpenNews's *Source* community site for journalists who code. She was formerly editorial director at Happy Cog Studios and a lead content strategist at Brain Traffic, and she edited *A List Apart* magazine for a long time. She lives in Brooklyn and tweets at @Kissane.

# 21. Infinite Canvas: Moving Beyond the Page

---

Nathan Peretic

[24ways.org/201221](http://24ways.org/201221)

Remember Web 2.0? I do. In fact, that phrase neatly bifurcates my life on the internet. Pre-2.0, I was occupied by chatting on AOL and eventually by learning HTML so I could build sites on Geocities. Around 2002, however, I saw a WYSIWYG demo in Dreamweaver. The instructor was dragging boxes and images around a canvas. With a few clicks he was able to build a dynamic, single-page interface. Coming from the world of tables and inline HTML styles, I was stunned.

As I entered college the next year, the web was blossoming: broadband, Wi-Fi, mobile (proud PDA owner, right here), CSS, Ajax, Bloglines, Gmail and, soon, Google Maps. I was a technology fanatic and a hobbyist web

developer. For me, the web had long been informational. It was now rapidly becoming something else, something more: sophisticated, presentational, actionable.

In 2003 we watched as the internet changed. The predominant theme of those early Web 2.0 years was the withering of Internet Explorer 6 and the triumph of web standards. Upon cresting that mountain, we looked around and collectively breathed the rarefied air of pristine HTML and CSS, uncontaminated by toxic hacks and forks – only to immediately begin hurtling down the other side at what is, frankly, terrifying speed.

Ten years later, we are still riding that rocket. Our days (and nights) are spent cramming for exams on CSS3 and RWD and Sass and RESS. We are the proud, frazzled owners of tiny pocket computers that annihilate the best laptops we could have imagined, and the architects of websites that are no longer restricted to big screens nor even segregated by device. We dragoon our sites into working any time, anywhere. At this point, we can hardly ask the spec developers to slow down to allow us to catch our breath, nor should we. It is, without a doubt, a most wonderful time to be a web developer.

But despite the newfound luxury of rounded corners, gradients, embeddable fonts, low-level graphics APIs, and, glory be, shadows, the canyon between HTML and native appears to be as wide as ever. The improvements in HTML



and CSS have, for the most part, been conveniences rather than fundamental shifts. What I'd like to do now, if you'll allow me, is outline just a few of the remaining gaps that continue to separate web sites and applications from their native companions.

## WHAT I'D LIKE FOR CHRISTMAS

There is one irritant which is the grandfather of them all, the one from which all others flow and have their being, and it is, simply, the page refresh. That's right, the foundational principle of the web is our single greatest foe. To paraphrase a patron saint of designers everywhere, if you see a page refresh, we blew it.

The page refresh brings with it, of course, many noble and lovely benefits: addressability, for one; and pagination, for another. (See also caching, resource loading, and probably half a dozen others.) Still, those concerns can be answered (and arguably answered more compellingly) by replacing the weary page with the young and hearty document.

**Flash may be dead**, but it has many lessons yet to bequeath.

Preparing a single document when the site loads allows us to engage the visitor in a smooth and engrossing experience. We have long known this, of course. Twitter was not the first to attempt, via JavaScript, to envelop the user in a single-page application, nor the first to abandon

it. Our shared task is to move those technologies down the stack, to make them more primitive, so that the next Twitter can be built with the most basic combination of HTML and CSS rather than relying on complicated, slow, and unreliable scripted solutions.

So, let's take a look at what we can do, right now, that we might have a better idea of where our current tools fall short.

## **A PRINT MAGAZINE IN HTML CLOTHING**

Like many others, I suspect, one of my earliest experiences with publishing was laying out newsletters and newspapers on a computer for print. If you've ever used InDesign or Quark or even Microsoft Publisher, you'll remember reflowing content from page to page. The advent of the internet signaled, in many ways, the abandonment of that model. Articles were no longer constrained by the physical limitations of paper. In shedding our chains, however, it is arguable that we've lost something useful. We had a self-contained and complete package, a closed loop. It was a thing that could be handled and finished, and doing so provided a sense of accomplishment that our modern, infinitely scrolling, ever-fractal web of content has stolen.

For our purposes today, we will treat 24 ways as the online equivalent of that newspaper or magazine. A single year's worth of articles could easily be considered an issue. Right now, navigating between articles means clicking on the article you'd like to view and being taken to that specific address via a page reload. If Drew wanted to, it wouldn't be difficult to update the page in place (via JavaScript) and change the address (again via JavaScript with the **History API**) to reflect the new content found at the new location. But what if Drew wanted to do that *without* JavaScript? And what if he wanted the site to not merely load the content but actually whisk you along the page in a compelling and delightful way, à la **the Mag+ demo** we all saw a few years ago when the iPad was first introduced? Uh, no.

We're all familiar with websites that have attempted to go beyond the page by weaving many chunks of content together into a large document and for good reason. There is tremendous appeal in opening and exploring the canvas beyond the edges of our screens.

In one rather straightforward example from last year, Mozilla contacted Full Stop to build a website promoting **Aza Raskin's proposal** for a set of Creative Commons-style privacy icons. Like a lot of the sites we build (**including our own**), the amount of information we were presenting was minimal. In these instances, we encourage our clients to consider including everything on a single

page. **The result** was a horizontally driven site that was, if not whimsical, at least clever and attractive to the intended audience. An experience that is taken for granted when using device-native technology is utterly, maddeningly impossible to replicate on the web without jumping through JavaScript hoops.

In another, more complex example, we again had the pleasure of working with Aza earlier this year, this time on a redesign of the Massive Health website. Our assignment was to design and build a site that communicated Massive's commitment to modern personal health. The site had to be visually and interactively stunning while maintaining a usable and clear interface for the casual visitor. **Our solution** was to extend the infinite company logo into a ribbon that carried the visitor through the site narrative. It also meant we'd be asking the browser to accommodate something it was never designed to handle: a non-linear design. (Be sure to play around. There's a lot going on under the hood. We were also *this close* to a ZUI, if WebKit didn't freak out when pages were scaled beyond 10x.) Despite the apparent and deliberate design simplicity, the techniques necessary to implement it are anything but. From updating the URL to moving the visitor from section to section, we're firmly in JavaScript territory. And that's a shame.

## WHAT CAN WE DO?

We might not be able to specify these layouts in HTML and CSS just yet, but that doesn't mean we can't learn a few new tricks while we wait. Let's see how close we can come to recreating the privacy icons design, the Massive design, or the Mag+ design without resorting to JavaScript.

## A HORIZONTALLY PAGINATED SITE

The first thing we're going to need is the concept of a page within our HTML document. Using plain old HTML and CSS, we can stack a series of `<div>`s sideways (with a little assist from our new friend, the viewport-width unit, not that he was strictly necessary). All we need to know is how many pages we have. (And, boy, wouldn't it be nice to be able to know that without having to predetermine it or use JavaScript?)

```
.window {
overflow: hidden;
  width: 100%;
}
.pages {
  width: 200vw;
}
.page {
  float: left;
```

```
overflow: hidden;  
width: 100vw;  
}
```

If you look carefully, you'll see that the conceit we'll use in the rest of the demos is in place. Despite the document containing multiple pages, only one is visible at any given time. This allows us to keep the user focused on the task (or content) at hand.

By the way, you'll need to use a modern, WebKit-based browser for these demos. I recommend downloading **the WebKit nightly builds**, **Chrome Canary**, or being comfortable with **setting flags in Chrome**.

## **A HORIZONTALLY PAGINATED SITE, WITH TRANSITIONS**

Ah, here's the rub. We have functional navigation, but precious few cues for the user. It's not much good shoving the visitor around various parts of the document if they don't get the pleasant whooshing experience of the journey. You might be thinking, what about that new CSS selector, `target-something...`? Well, my friend, you're on the right track. **Let's test it.** We're going to need to use a bit of sleight of hand. While we'd like to simply offset the containing element by the number of pages we're moving (like we did on Massive), CSS alone can't give us that

information, and that means we're going to need to fake it by expanding and collapsing pages as you navigate. Here are the bits we're going to need:

```
.page {
  -webkit-transition: width 1s; // Naturally you're
going to want to include all the relevant prefixes here
  float: left;
  left: 0;
  overflow: hidden;
  position: relative;
  width: 100vw;
}
.page:not(:target) {
  width: 0;
}
```

Ah, but we're not fooling anyone with that trick. As soon as you move beyond a single page, the visitor's disbelief comes tumbling down when the linear page transitions are unaffected by the distance the pages are allegedly traveling. And you may have already noticed an even more fatal flaw: I secretly linked you to the first page rather than the unadorned URL. If you visit the same page with no URL fragment, you get a blank screen. Sure, we could force a redirect with some server-side trickery, but that feels like cheating. Perhaps if we had **the CSS4 subject selector** we could apply styles to the parent based on the child being targeted by the URL. We might also need a few

more abilities, like determining the total number of pages and having relative sibling selectors (e.g. `nth-sibling`), but we'd sure be a lot closer.

## A HORIZONTALLY PAGINATED SITE, WITH TRANSITIONS – NO CHEATING

Well, what other cards can we play? How about the **checkbox hack**? Sure, it's a garish trick, but it might be the best we can do today. Check it out.

```
label {
  cursor: pointer;
}
input {
  display: none;
}
input:not(:checked) + .page {
  max-height: 100vh;
  width: 0;
}
```

Finally, we can see the first page thanks to the state we are able to set on the appropriate radio button. Of course, now we don't have URLs, so maybe this isn't a winning plan after all. While our HTML and CSS toolkit may feel primitive at the moment, we certainly don't want to sacrifice the addressability of the web. If there's one bedrock principle, that's it.



## A HORIZONTALLY PAGINATED SITE, WITH TRANSITIONS – NO CHEATING AND A GORGEOUS HOMEPAGE

Gorgeous may not be the right word, but our little magazine is finally shaping up. Thanks to the CSS regions spec, we've got an exciting new power, the ability to begin an article in one place and bend it to our will. (Remember, your everyday browser isn't going to work for these demos. Try the **WebKit nightly build** to see what we're talking about.) As with the rest of the examples, we're clearly abusing these features. Off-canvas layouts (you can thank **Luke Wroblewski** for the name) are simply not considered to be normal patterns... yet.

Here's a quick look at what's going on:

```
.excerpt-container {
  float: left;
  padding: 2em;
  position: relative;
  width: 100%;
}
.excerpt {
  height: 16em;
}
.excerpt_name_article-1,
.page-1 .article-flow-region {
  -webkit-flow-from: article-1;
}
```

```
.article-content_for_article-1 {  
  -webkit-flow-into: article-1;  
}
```

The regions pattern is comprised of at least three components: a beginning; an ending; and a source. Using CSS, we're able to define specific elements that should be available for the content to flow through. If magazine-style layouts are something you're interested in learning more about (and you should be), be sure to check out [the great work Adobe has been doing](#).

## LOOKING FORWARD, AND BACKWARD

As designers, builders, and consumers of the web, we share a desire to see the usability and enjoyability of websites continue to rise. We are incredibly lucky to be working in a time when a three-month-old website can be laughably outdated. Our goal ought to be to improve upon both the weaknesses *and* the strengths of the web platform. We seek not only smoother transitions and larger canvases, but fine-grained addressability. Our URLs should point directly and unambiguously to specific content elements, be they pages, sections, paragraphs or words. Moreover, off-screen design patterns are essential to accommodating and empowering the multitude of devices we use to access the web. We should express the desire that interpage links take advantage of the CSS transitions which have been put to such good effect in

every other aspect of our designs. Transitions aren't just nice to have, they're table stakes in the highly competitive world of native applications.

The tools and technologies we have right now allow us to create smart, beautiful, useful webpages. With a little help, we can begin removing the seams and sutures that bind the web to an earlier, less sophisticated generation.

## ABOUT THE AUTHOR



Co-founder of Full Stop Interactive, a web shop in Pittsburgh, PA, and United Pixelworkers, a fake union for people who make the things we see on our screens every day. Developer, writer, reluctant businessman. **Nathan Peretic** would rather be reading.

# 22. Unwrapping the Wii U Browser

---

Anna Debenham

24ways.org/201222

The Wii U was released on 18 November 2012 in the US, and 30 November in the UK. It's the first eighth generation home console, the first mainstream second-screen device, and it has some really impressive browser specs.

Consoles are not just for games now: they're marketed as complete entertainment solutions. Internet connectivity and browser functionality have gone from a nice-to-have feature in game consoles to a **selling point**. In Nintendo's case, they see it as a challenge to design an experience that's better than browsing on a desktop.

Let's make a browser that users can use on a daily basis, something that can really handle everything we've come to expect from a browser and do it more naturally.

*Sasaki – Iwata Asks on Nintendo.com*

With 11% of people using console browsers to visit websites, it's important to consider these devices right from the start of projects. Browsing the web on a TV or handheld console is a very different experience to browsing on a desktop or a mobile phone, and has many usability implications.

### CONSOLE BROWSER TESTING

When I'm testing a console browser, one of the first things I do is run Niels Leenheer's HTML5 test and Lea Verou's CSS3 test. I use these benchmarks as a rough comparison of the standards each browser supports.

In October, IE9 came out for the Xbox 360, scoring 120/500 in the HTML5 test and 32% in the CSS3 test. The PS Vita also had an update to its browser in recent weeks, jumping from 58/500 to 243/500 in the HTML5 test, and 32% to 55% in the CSS3 test. Manufacturers have been stepping up their game, trying to make their browsing experiences better.

To give you an idea of how the Wii U currently compares to other devices, here are the test results of the other TV consoles I've tested. I've written more in-depth notes on TV and portable console browsers separately.

	Year of release	HTML5 score	CSS3 score	Notes
Wii U	2012	258/500	48%	Runs a Netfront browser (WebKit).
Wii	2006	89/500	Wouldn't run	Runs an Opera browser.
PS3	2006	68/500	38%	Runs a Netfront browser (WebKit).
Xbox 360	2005	120/500	32%	A browser for the Xbox (IE9) was only recently released in October 2012. The Kinect provides voice and gesture support. There's also SmartGlass, a second-screen app for platforms including Android and iOS.

The **Wii U browser** is Nintendo's fifth attempt at a console browser. Based on these tests, it's already looking promising.

## WHY CONSOLE BROWSERS USED TO SUCK

It takes a lot of system memory to run a good browser, and the problem of older consoles is that they don't have much memory available. The original Nintendo DS needs a memory expansion pack just to run the browser,

because the 4MB it has on board isn't enough. I noticed that even on newer devices, some sites fail to load because the system runs out of memory.

The Wii came out six years ago with an Opera browser. Still being used today and with such low resources available, the latest browser features can't reasonably be supported. There's also pressure to add features such as tabs, and enable gamers to use the browser while a game is paused. Nintendo's browser team have the advantage of higher specs to play with on their new console (1GB of memory dedicated to games, 1GB for the system), which makes it easier to support the latest standards. But it's still a challenge to fit everything in.

...even though we have more memory, the amount of memory we can use for the browser is limited compared to a PC, so we've worked in ways that efficiently allocates the available memory per tab. To work on this, the experience working on the browser for the Nintendo 3DS system under a limited memory constraint helped us greatly.

*Sasaki - Iwata Asks on Nintendo.com*

## IN THE BOX

The Wii U consists of a console unit which plugs into a TV (the first to support HD), and a wireless controller known as a gamepad. The gamepad is a lot bigger than typical TV console controllers, and it has a touchscreen on the front. The touchscreen is resistive, responding to pressure rather than electrical current. It's intended to be used with a stylus (provided) but fingers can be used.

It might look a bit like one, but the gamepad *isn't* a portable console designed to be taken out like the PS Vita. The gamepad can be used as a standalone screen with the TV switched off, as long as it's within range of the console unit – it basically piggybacks off it.





It's surprisingly lightweight for its size. It has a wealth of detectors including 9-axis control. Sensors wake the device from sleep when it's picked up. There's also a camera on the front, and a headphone port and speakers, with audio coming through both the TV and the gamepad giving a surround sound feel.

Up to six tabs can be opened at once, and the browser can be used while games are paused. There's a really nice little feature here – the current game's name is saved as a search option, so it's really quick to look up contextual content such as walk-throughs.

### CONTROLS

Only one gamepad can be used to control the browser, but if there are Wiimotes connected, they can be used as pointers. This doesn't let the user do anything except point (they each get a little hand icon with a number on it displayed on the screen), but it's interesting that multiple people can be interacting with a site at once.



See a bigger version

The gamepad can also be used as a simple TV remote control, with basic functions such as bringing up the programme guide, adjusting volume and changing channel. I found the simplified interface much more usable than a full-featured remote control.



I'm used to scrolling being sluggish on consoles, but the Wii U feels almost as snappy as a desktop browser. Sites load considerably faster compared with others I've tested.

## Tilt-scroll

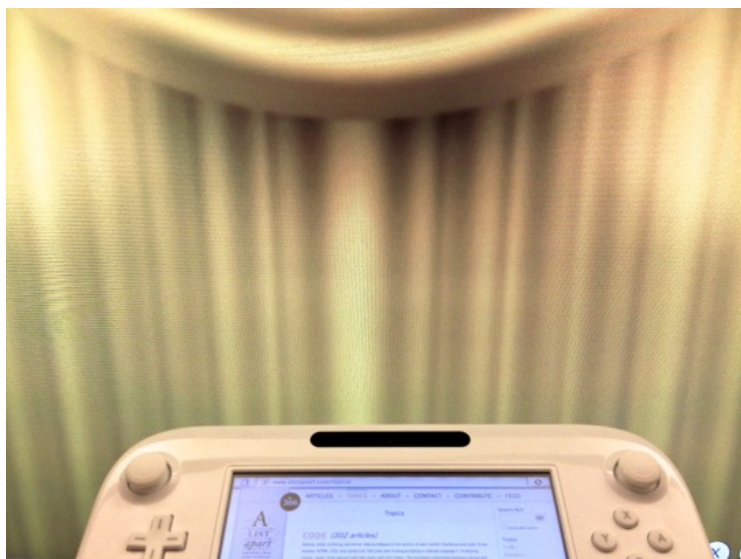
Holding down **ZL** and **ZR** while tilting the screen activates an Instapaper-style tilt to scroll for going up and down the page quickly, useful for navigating very long pages.

## SECOND SCREEN

The TV mirrors most of what's on the gamepad, although the TV screen just displays the contents of the browser window, while the gamepad displays the site along with the browser toolbar.

When the user with the gamepad is typing, the keyboard is hidden from the TV screen – there's just a bit of text at the top indicating what's happening on the gamepad.

Pressing **X** draws an on-screen curtain over the TV, hiding the content that's on the gamepad from the TV. Pressing **X** again opens the curtains, revealing what's on the gamepad. Holding the button down plays a drumroll before it's released and the curtains are opened. I can imagine this being used in meetings as a fun presentation tool.



In a sense, browsing is a personal activity, but you get the idea that people will be coming and going through the room. When I first saw the curtain function, it made a huge impression on me. I walked around with it all over the company saying, “They’ve really come up with something amazing!”

*Iwata - Iwata Asks on Nintendo.com*



## TEXT

### Writing text

Unlike the capacitive screens on smartphones, the Wii U's resistive screen needs to be pressed harder than you're probably used to for registering a touch event. The gamepad screen is big, which makes it much easier to type on this device than other handheld consoles, even without the stylus. It's still more fiddly than a full-sized keyboard though. When you're designing forms, consider the extra difficulty console users experience.



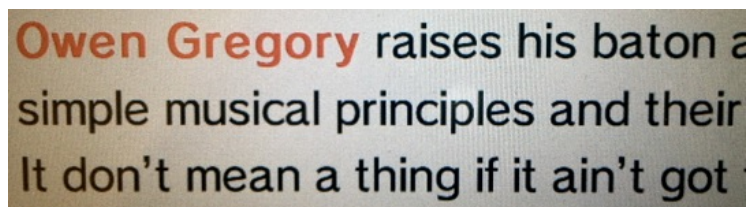
Although TV screens are physically big, they are typically viewed from further away than desktop screens. This makes readability an issue, so Nintendo have provided not one, but four ways to zoom in and out:

- Double-tapping on the screen.
- Tapping the on-screen zoom icons in the browser toolbar.
- Pressing the  and  buttons on the device.
- Moving the right analogue stick up and down.

As well as making it easy to zoom in and out, Nintendo have done a few other things to improve the reading experience on the TV.

### **System font**

One thing you'll notice pretty quickly is that the browser lacks all the fonts we're used to falling back to. Serif fonts are replaced with the system's sans-serif font. I couldn't get Typekit's font loading method to work but Fontdeck, which works slightly differently, does display custom fonts.



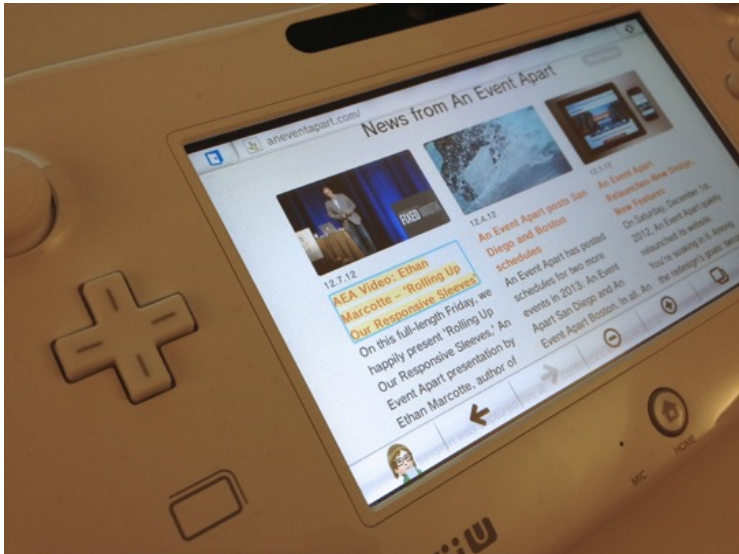
The system font has been optimised for reading at a distance and is easy to distinguish because the lowercase e has a quirky little tilt.

---

### Don't lose :focus

Using the D-pad to navigate is similar to using a keyboard. Individual links are focused on, with a blue outline drawn around them.

The recently redesigned **An Event Apart** site is an example that improves the experience for keyboard and D-pad users. They've added a yellow background colour to links on focus. It feels nicer than the default blue outline on its own.



## MEDIA

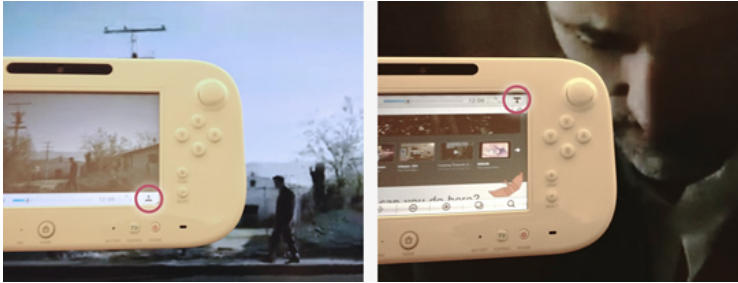
This year, television overtook PCs as the primary way to watch online video content. TV is the natural environment for video, and 42% of online TVs in the US are connected to the internet via a console. Unfortunately, the `<video>` tag isn't supported in most console browsers, and those that have Flash installed often have such an old version that the video won't play.

I suspect this has been a big driver in getting console browsers to support web standards. The Wii U is designed with video content in mind. It doesn't support Flash but it *does* support the HTML5 `<video>` tag.

Some video formats can't be played, but those that are supported bring up an optimised interface with a custom scrub bar. This is where the device switches from mirroring the TV to being a second screen. The full-screen video is displayed on the TV, and the interface on the gamepad.

The really clever bit is that while a video is playing, the gamepad user can keep the video playing on the TV screen while searching for another video or browsing the web. This is the same for images too.





On the left, the video is being shown full-screen on the TV and gamepad. Only the gamepad gets the scrub bar. Clicking the slide up/down button (circled) lets the gamepad user browse the web while the video on the TV continues to play full-screen, as shown in the image on the right.

---

There's support for SVG images, and they look great on a high-definition TV screen. However, there's currently no way to save or download files.

### **PREPARING FOR CONSOLE USERS**

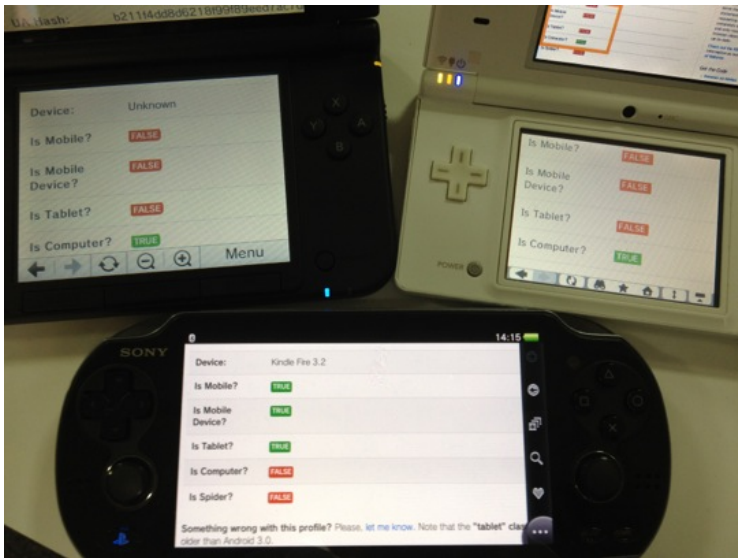
I wasn't expecting to be quite as impressed as I am by this browser. It's encouraging to see console makers investing time into improving the experience as well as the standards support. In the same way there was an explosion in mobile browser use as the experience got better, I'm sure we'll see the same with console browsers as the experience improves.

## The value of detection

Consoles offer a range of inputs including gesture, voice and controller buttons. That means we have to think about more diverse methods of input than just touch and click.

This is where I could tell you to add some detection methods such as user agent string sniffing to target a different experience for console users. But the majority of the time, that really isn't necessary. TV console browsers are getting a lot better at compensating for the living room environment, and they're designed to work with websites that haven't been optimised for this context.

Rather than tighten our grip on optimising experiences for every device out there, we've got to be pragmatic. There are so many new devices coming out every week, our designs need to be future-proof rather than fixed to a particular device in time.



Even fuzzy device detection isn't reliable – the PS Vita declares itself to be mobile, a mobile device and a Kindle Fire tablet, while the two DS devices state they're neither mobile nor mobile phones nor tablets, but computers. They're weird outliers, but they're still important devices to consider.

---

Thinking broadly about how our designs will be interacted with and viewed on a TV screen can help improve that experience for everyone. This is about accessibility. Considering how someone uses a site with a D-pad, we can improve the experience for keyboard users. When we think about colour contrast and text legibility on TV screens, we can apply that for anyone who reads content on the web. So why just offer this to the TV users?

---

## Playing with the browser

...we want to prove to them through this Wii U Internet Browser that browsing itself can be an entertainment.

*Iwata - Iwata Asks on Nintendo.com*

Although I'm cautious about designing experiences for specific devices, it's fun to experiment with the technology available. Nintendo have promised web developers access to the Wii U's buttons and sensors. There's already some JavaScript documentation, and a demo for you to try.

If you're interested in making your own games, thanks to web standards, a lot of HTML5 games work already on the device. Matt Hackett wrote up his experience of testing the game he built, and he talks about some of features the browser lacks. There's certainly an incentive there for console manufacturers to improve their HTML5 support so more games can be played in their browser.

What excites me about consoles is that it's like looking at what might be available to us in future browsers. As well as thinking about how our sites work on small screens, we should also consider big screens. We're already figuring out how images should work at different screen widths and connection speeds, but we've also got some interesting challenges ahead of us catering for second screen experiences and 3D-enabled devices.

So, this Christmas, if you're huddled round the game console or a smart TV, give the browser in it a try.

## 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.

# 23. Monkey Business

---

Andrew Clarke

24ways.org/201223

“Too expensive.” “Over-priced.” “A bit rich.”

They all mean the same thing.

When you say that something's too expensive, you're doing much more than commenting on a price. You're questioning the explicit or implicit value of a product or a service. You're asking, “Will I get out of it what you want me to pay for it?” You're questioning the competency, judgement and possibly even integrity of the individual or company that gave you that price, even though you don't realise it. You might not be saying it explicitly, but what you're implying is, “Have you made a mistake?”, “Am I getting the best deal?”, “Are you being honest with me?”, “Could I get this cheaper?”

Finally, you're being dishonest, because deep down you know all too well that there's no such thing as too expensive.

Why?

It doesn't matter what you're questioning the price of. It could be a product, a service or the cost of an hour, day or week of someone's time. Whatever you're buying, too expensive is always an excuse. Saying it shifts acceptability of a price back to the person who gave it. What you should say, but are too afraid to admit, is:

- "It's more money than I *wanted* to pay."
- "It's more than I *estimated* it would cost."
- "It's more than I can *afford*."

Everyone who's given a price for a product or service will have been told at some point that it's too expensive. It's never comfortable to hear that. Thoughts come thick and fast: "What do I do?" "How do I react?" "Do I really want the business?" "Am I prepared to negotiate?" "How much am I willing to compromise?"

It's easy to be defensive when someone questions a price, but before you react, stay calm and remember that if someone says what you're offering is too expensive, they're saying more about themselves and their situation than they are about your price. Learn to read that situation and how to follow up with the right questions.

Imagine you've quoted someone for a week of your time. "That's too expensive," they respond. How should you handle that? Think about what they might otherwise be saying.



“It’s more money than I want to pay” may mean that they don’t understand the value of your service. How could you respond?

Start by asking what similar projects they’ve worked on and the type of people they worked with. Find out what they paid and what they got for their money, because it’s possible what you offer is different from what they had before. Ask if they saw a return on that previous investment. Maybe their problem isn’t with your headline price, but the value they think they’ll receive. Put the emphasis on value and shift the conversation to what they’ll gain, rather than what they’ll spend.

It’s also possible they can’t distinguish your service from those of your competitors, so now would be a great time to explain the differences. Do you work faster? Explain how that could help them launch faster, get customers faster, make money faster. Do you include more? Emphasise that, and how unique the experience of working with you will be.



“It’s more than I estimated it would cost” could mean that your customer hasn’t done their research properly. You’d never suggest that to them, of course, but you should ask how they’ve arrived at their estimate. Did they base it on



work they've purchased previously? How long ago was that? Does it come from comparable work or from a different sector?

Help your customer by explaining how you arrived at your estimate. Break down each element and while you're doing that, emphasise the parts of your process that you know will appeal to them. If you know that they've had difficulty with something in the past, explain how your approach will benefit them. People almost always value a positive experience more than the money they'll save.



"It's more than I can afford" could mean they can't afford what you offer at all, but it could also mean they can't afford it right now or all at once. So ask if they could afford what you're asking if they spread payment over a longer period? Ask, "Would that mean you'll give me the business?"

It's possible they're asking for too much for what they can afford to pay. Will they compromise? Can you reach an agreement on something less? Ask, "If we can agree what's in and what's out, will you give me the business?"

What can they afford? When you know, you're in a good position to decide if the deal makes good business sense, for both of you. Ask, "If I can match that price, will you give me the business?"

There's no such thing as "a bit rich", only ways for you to get to know your customer better. There's no such thing as "over-priced", only opportunities for you to explain yourself better. You should relish those opportunities. There's really also no such thing as "too expensive", just ways to set the tone for your relationship and help you develop that relationship to a point where money will be less of a deciding factor.

## **UNFINISHED BUSINESS**

Join me and my co-host **Anna Debenham** next year for **Unfinished Business**, a new discussion show about the business end of working in web, design and creative industries.

## 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. Science!

---

Jon Tan

24ways.org/201224

Sometimes we want to capture people's attention at a glance to communicate something fast. At other times we want to have the interface fade away into the background, letting people paint pictures in their minds with our words (if you'll forgive a little flowery festive flourish).

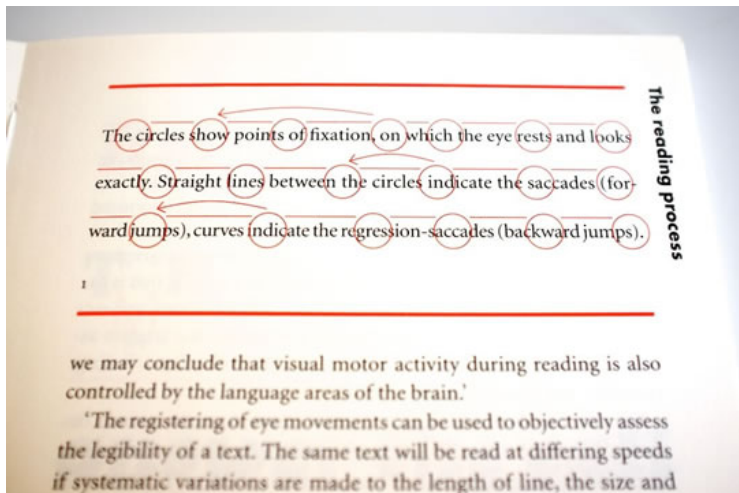
I tend to distinguish between these two broad objectives as designing for impact on the one hand, and designing for immersion on the other. What defines them is interruption. Impact needs an attention-grabbing interruption. Immersion requires us to remove interruption from the interface. Careful design *deliberately* interrupts but doesn't *accidentally* disrupt. If that seems to make sense to you, then you'll find the following snippets of science as useful as I did.

## **SACCADES AND FIXATIONS**

As you're reading this your eyes are skipping along the lines in tiny jumps. During each jump everything is blurred. Each jump ends in a small pause so your brain can

take a snapshot of the letters. It arranges them into words, and then parses out the meaning — fast — in around a quarter of a second.

The jumps are called saccades. The pauses are called fixations. Sometimes we take regressive saccades, skipping back to reread. There's a simple example in the excellent little book, *Detail in Typography*, by Jost Hochuli.

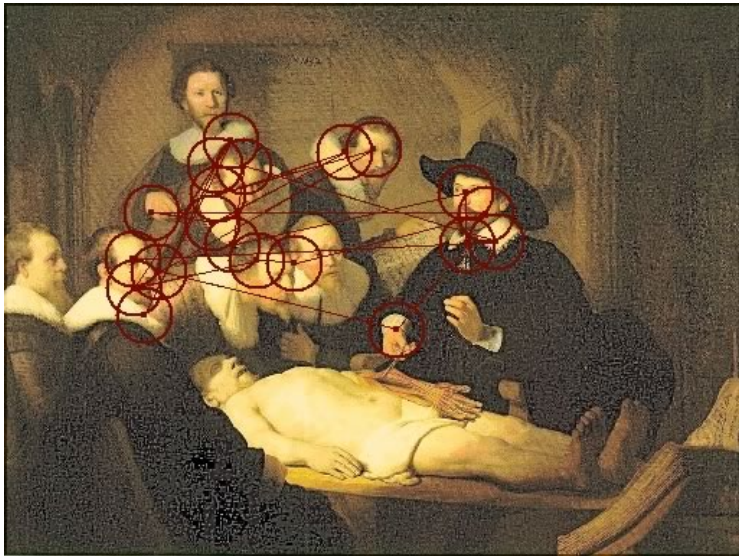


If you want to explore the science of reading in much more depth, I recommend the excellent paper, “The Science of Word Recognition”, by Dr Kevin Larson of Microsoft.

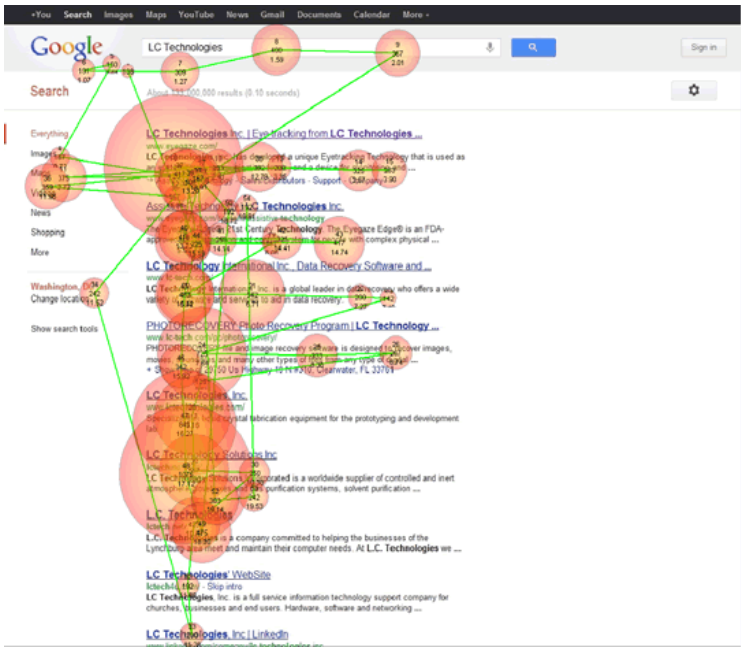
To design for legibility and readability is to design for saccades and fixations. It's the craft of making it easy for people's brains to extract meaning, using techniques like good contrast, font size, spacing and structure, and only interrupting the reading experience deliberately.

## SCAN PATHS

At some point when visiting 24 ways you probably scanned the screen to get orientated. The journey your eyes took is known as a scan path. Scan paths are made up of saccades and fixations. Right now you're following a scan path as you read, along one line, and down to the next. This is a map of the scan paths found by Olivier Le Meur from observing people looking at Rembrandt's *Leçon d'anatomie*:

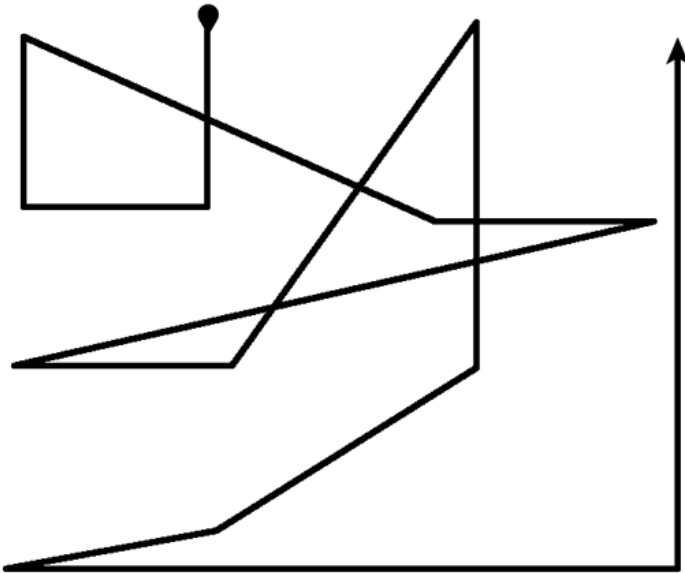


For websites, the scan path is a little different. This is an aggregate scan path of Google from LC Technologies:



The average shape of a website scan path becomes clearer in this average scan path taken by forty-six people during research by the Poynter Institute, the Estlow Center for Journalism & New Media, and Eyetools:





Just like when we read text arranged left to right in a vertical column, scan paths follow a roughly Z-shaped pattern from the top left to bottom right. Sometimes we skip back to reread a word or sentence, or glance again at a specific element, but the Z-shaped scan path persists.

Designing for scan paths is to organise content to help people move through an interface to get orientated, and to read.

The elements that are important enough to need impact must interrupt the scan path and clearly call attention to themselves. However, they don't always need to clip people round the ear from multiple directions at once to

get attention. It helps to list elements by importance. That gives us an interruption hierarchy to work with. Elements can then interrupt the design with degrees of contrast to the rest of the content using either positioning, treatment, or both. Ta-da! Impact achieved, but gently. No clips round the ear required.

## **SWINGING MOOD**

Human beings are resilient. Among the immersion and occasional interruptions, we even like a little disruption, especially if it's absurd and funny. The **Ling's Cars** website proves it. In fact, we're so resilient that we can work around all kinds of mayhem to get a seemingly simple task done.

In one study, "**The Aesthetics of Reading**" (PDF, 480Kb), Dr Kevin Larson of Microsoft and Dr Rosalind Picard of MIT explored the effect of good typography on mood. Two versions of the New Yorker ePeriodical were created. One was typeset well and the other poorly.

## AN APPENDECTOMY ON THE BAKERLOO LINE

BY GRAHAM CHAPMAN

Dear Sirs,  
I've had letter after letter after letter since you published one particular query that asked, "What should I do about my appendix on the Bakerloo Line?" Well, "Miss N.," I can

and ask for one. Remember, the stations marked with an "O" are interchange stations. Stations marked with a star are closed on Sundays and also remember to pick up a plastic bucket for the guts.

Then study your map and find the brown line clearly marked "Bakerloo" in the key. Select a station appropriate to the severity of the inflammation. For mild or grumbling appendicitis, you could start at Lambeth North—being careful not to change at Waterloo—and have comfortably incised your abdomen and exposed the inflamed organ by the time you are between Marblehouse and Kilburn Park. You will then have the time it takes you to reach Willesden Junction to complete the excision. And the six minutes between Chisle and Wembley



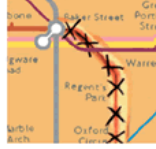
## AN APPENDECTOMY ON THE BAKERLOO LINE

BY GRAHAM CHAPMAN

Dear Sirs,  
I've had letter after letter after letter since you published one particular query that asked, "What should I do about my

later on. I have set out some details that may help you in this sometimes it's close. First, find yourself a Tube Map, issued free by London Transport, or go to your nearest Underground station and ask for one. Remember, the stations marked with an "O" are interchange stations. Stations marked with a star are closed on Sundays, and also remember to pick up a plastic bucket for the guts.

Then study your map and find the brown line clearly marked "Bakerloo" in the key. Select a station appropriate to the severity of the inflammation. For mild or grumbling



They engaged twenty volunteers — half male, half female — and showed the good version to half of the participants. The other half saw the poor version.

The good doctors found that, "there are important differences between good and poor typography that appear to have little effect on common performance measures such as reading speed and comprehension." In short, good typography didn't help people read faster or comprehend better.

Oh. On the face of it that seems to invalidate what we designers do. Hold your horses, though! They also found that "the participants who received the good typography performed better on relative subjective duration and on certain cognitive tasks", and that "good typography induces a good mood."

This means that even though there were no actual differences in reading speed and comprehension, the people who read the version with good typography

*thought* that it took less time to read, and were induced into a good mood by doing so. Not only that, but by being in a good mood, people were more capable of completing creative tasks faster.

That was a revelation to me. It means that the study showed there is a positive, measurable, emotional and perceptual benefit to good typography and design. To paraphrase: time and tasks fly when you're having fun!



Source: Nationaal Archief of the Netherlands: Cheering man after the first goal, Netherlands vs. Belgium, Amsterdam, 1931.

So, among all my talk of saccades, fixations, scan paths and typesetting, there is science, and the science helps us qualify our design decisions when we need to, and do our jobs better. The science helps us understand how people will interact with our work, and what the actual benefits

are for them, and the products or organisations we serve. Good design equals a subjectively quicker experience, a good mood, objectively faster completion of tasks, and happiness for everyone. Thank you, science!

## ABOUT THE AUTHOR



**Jon Tan** is a designer and typographer who co-founded the web fonts service, **Fontdeck**. He's a partner in **Fictive Kin**, where he works with friends making things like **Brooklyn Beta** and **Mapalong**.

His addiction to web typography has led him to share snippets of type news via **@t8y**. He also writes for publications like **Typographica** and **8 Faces**, speaks at international events like **An Event Apart**, and works with such organisations as the **BBC**.

Jon is based in **Mild Bunch HQ**, the co-working studio he started in Bristol, UK. He can often be found wrestling with his two sons, losing, then celebrating the fact as [@jontangerine](#) on Twitter.