24

2008

24 WAYS

Credits

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

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

2008

This year saw Apple's App Store open, and the release of Android 1.0 and Google Chrome 1.0. Taking all that in its stride, 24 ways brought its seasonal perspective to bear on business, with articles on project management, the path from design to development, how to charm clients, and killer contracts. Also, a first look at modular layout systems. Pulse, meet finger.

1. Easing The Path from Design to Development

Drew McLellan 24ways.org/200801

As a web developer, I have the pleasure of working with a lot of different designers. There has been a lot of industry discussion of late about designers and developers, focusing on how different we sometimes are and how the interface between our respective phases of a project (that is to say moving from a design phase into production) can sometimes become a battleground.

I don’t believe it has to be a battleground. It’s actually more like being a dance partner – our steps are different, but as long as we know our own part and have a little knowledge of our partner’s steps, it all goes together to form a cohesive dance. Albeit with less spandex and fewer sequins (although that may depend on the project in question).

As the process usually flows from design towards development, it's most important that designers have a little knowledge of how the site is going to be built. At the specialist web development agency I'm part of, we find that designs that have been well considered from a technical perspective help to keep the project on track and on budget.

Based on that experience, I've put together my checklist of things that designers should consider before handing their work over to a developer to build.

LAYOUT

One rookie mistake made by traditionally trained designers transferring to the web is to forget a web browser is not a fixed medium. Unlike designing a magazine layout or a piece of packaging, there are lots of available options to consider. Should the layout be fluid and resize with the window, or should it be set to a fixed width? If it's fluid, which parts expand and which not? If it's fixed, should it sit in the middle of the window or to one side?

If any part of the layout is going to be flexible (get wider and narrower as required), consider how any graphics are affected. Images don't usually look good if displayed at anything other that their original size, so should they behave? If a column is going to get wider than it's shown in the Photoshop comp, it may be necessary to provide separate wider versions of any background images.

TEXT SIZE AND CONTENT VOLUME

A related issue is considering how the layout behaves with both different sizes of text and different volumes of content. Whilst text zooming rather than text resizing is becoming more commonplace as the default behaviour in browsers, it's still a fundamentally important principal of web design that we are suggesting and not dictating how something should look. Designs must allow for a little give and take in the text size, and how this affects the design needs to be taken into consideration.

Keep in mind that the same font can display differently in different places and platforms. Something as simple as Times will display wider on a Mac than on Windows. However, the main impact of text resizing is the change in how much vertical space copy takes up. This is particularly important where space is limited by the design (making text bigger causes many more problems than making text smaller). Each element from headings to box-outs to navigation items and buttons needs to be able to expand at least vertically, if not horizontally as well. This may require some thought about how elements on the page may wrap onto new lines, as well as again making sure to provide extended versions of any graphical elements.

Similarly, it’s rare theses days to know exactly what content you’re working with when a site is designed. Many, if not most sites are designed as a series of templates for some kind of content management system, and so designs cannot be tweaked around any specific item of content. Designs must be able to cope with both much greater and much lesser volumes of content that might be thrown in at the lorem ipsum phase.

Particular things to watch out for are things like headings (how do they wrap onto multiple lines) and any user-generated items like usernames. It can be very easy to forget that whilst you might expect something like a username to be 8-12 characters, if the systems powering your site allow for 255 characters they’ll always be someone who’ll go there. Expect them to do so.

Again, if your site is content managed or not, consider the possibility that the structure might be expanded in the future. Consider how additional items might be added to each level of navigation. Whilst it’s rarely desirable to make significant changes without revisiting the site’s information architecture more thoroughly, it’s an inevitable fact of life that the structure needs a little bit of flexibility to change over time.

INTERACTIONS WITH AND WITHOUT JAVASCRIPT

A great number of sites now make good use of JavaScript to streamline the user interface and make everything just that touch more usable. Remember, though, that any developer worth their salt will start by building the interface without JavaScript, get it all working, and then layer that JavaScript on top. This is to allow for users viewing the site without JavaScript available or enabled in their browser.

Designers need to consider both states of any feature they're designing – how it looks and functions with and without JavaScript. If the feature does something fancy with Ajax, consider how the same can be achieved with basic HTML forms, links and intermediary pages. These all need to be designed, because this is how some of your users will interact with the site.

LOGGED IN AND LOGGED OUT STATES

When designing any type of web application or site that has a membership system – that is to say users can create an account and log into the site – the design will need to consider how any element is presented in both logged in and logged out states. For some items there'll be no difference, whereas for others there may be considerable differences.

Should an item be hidden completely not logged out users? Should it look different in some way? Perhaps it should look the same, but prompt the user to log in when they interact with it. If so, what form should that prompt take on and how does the user progress through the authentication process to arrive back at the task they were originally trying to complete?

Couple logged in and logged out states with the possible absence of JavaScript, and every feature needs to be designed in four different states:

- Logged out with JavaScript available
- Logged in with JavaScript available
- Logged out without JavaScript available
- Logged in without JavaScript available

FONTS

There are three main causes of war in this world; religions, politics and fonts. I've said publicly before that I believe the responsibility for this falls squarely at the feet of Adobe Photoshop. Photoshop, like a mistress at a brothel, parades a vast array of ropey, yet strangely enticing typefaces past the eyes of weak, lily-livered designers, who can't help but crumble to their curvy charms.

Yet, on the web, we have to be a little more restrained in our choice of typefaces. The purest solution is always to make the best use of the available fonts, but this isn't always the most desirable solution from a design point of view. There are several technical solutions such as techniques that utilise Flash (like sIFR), dynamically generated images and even canvas in newer browsers. Discuss the best approach with your developer, as every different technique has different trade-offs, and this may impact the design in other ways.

MESSAGING

Any site that has interactive elements, from a simple contact form through to fully featured online software application, involves some kind of user messaging. By this I mean the error messages when something goes wrong and the success and thank-you messages when something goes right. These typically appear as the result of an interaction, so are easy to forget and miss off a Photoshop comp.

For every form, consider what gets displayed to the user if they make a mistake or miss something out, and also what gets displayed back when the interaction is successful. What do they see and where do the go next?

With Ajax interactions, the user doesn't get any visual feedback of the site waiting for a response from the server unless you design it that way. Consider using a 'waiting' or 'in progress' spinner to give the user some visual feedback of any background processes. How should these look? How do they animate?

Similarly, also consider the big error pages like a 404. With luck, these won't often be seen, but it's at the point that they are when careful design matters the most.

FORM FIELDS

Depending on the visual style of your site, the look of a browser's default form fields and buttons can sometimes jar. It's understandable that many a designer wants to change the way they look. Depending on the browser in question, various things can be done to style form fields and their buttons (although it's not as flexible as most would like).

A common request is to replace the default buttons with a graphical button. This is usually achievable in most cases, although it's not easy to get a consistent result across all browsers – particularly when it comes to vertical positioning and the space surrounding the button. If the layout is very precise, or if space is at a premium, it's always best to try and live with the browser's default form controls.

Whichever way you go, it's important to remember that in general, each form field should have a label, and each form should have a submit button. If you find that your form breaks either of those rules, you should double check.

PRACTICAL TIPS FOR HANDING FILES OVER

There are a couple of basic steps that a design can carry out to make sure that the developer has the best chance of implementing the design exactly as envisioned.

If working with Photoshop of Fireworks or similar comping tool, it helps to **group and label layers** to make it easy for a developer to see which need to be turned on and off to get to isolate parts of the page and different states of the design. Also, if you don't work in the same office as your developer (and so they can't quickly check with you), provide a PDF of each page and state so that your developer can see how each page should look aside from any confusion with quick layers are switched on or off. These also act as a handy quick reference that can be used without firing up Photoshop (which can kill both productivity and your machine).

Finally, **provide a colour reference** showing the RGB values of all the key colours used throughout the design. Without this, the developer will end up colour-picking from the comps, and could potentially end up with different colours to those you intended. Remember, for a lot of developers, working in a tool like Photoshop is like presenting a designer with an SSH terminal into a web server. It's unfamiliar ground and easy to get things wrong. Be the expert of your own domain and help your colleagues out when they're out of their comfort zone. That goes both ways.

IN CONCLUSION

When asked the question of how to smooth hand-over between design and development, almost everyone who has experienced this situation could come up with their own list. This one is mine, based on some of the more common experiences we have at edgeofmyseat.com. So in bullet point form, here's my checklist for handing a design over.

- Is the layout fixed, or fluid?
- Does each element cope with expanding for larger text and more content?
- Are all the graphics large enough to cope with an area expanding?
- Does each interactive element have a state for with and without JavaScript?
- Does each element have a state for logged in and logged out users?
- How are any custom fonts being displayed? (and does the developer have the font to use?)

- Does each interactive element have error and success messages designed?
- Do all form fields have a label and each form a submit button?
- Is your Photoshop comp document well organised?
- Have you provided flat PDFs of each state?
- Have you provided a colour reference?
- Are we having fun yet?

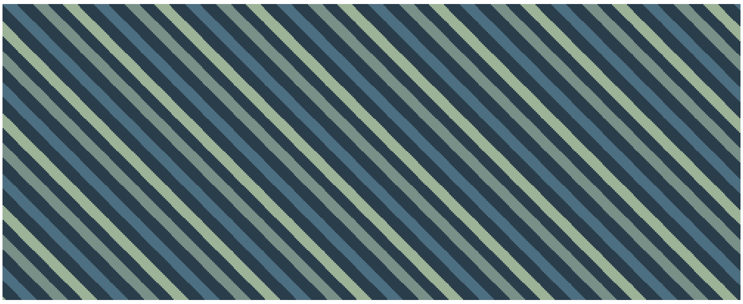
ABOUT THE AUTHOR

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

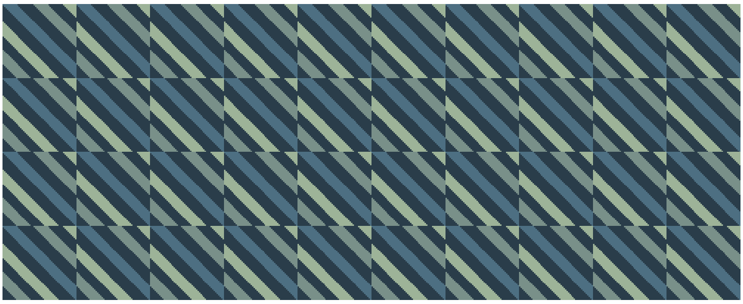
2. Geometric Background Patterns

Veerle Pieters 24ways.org/200802

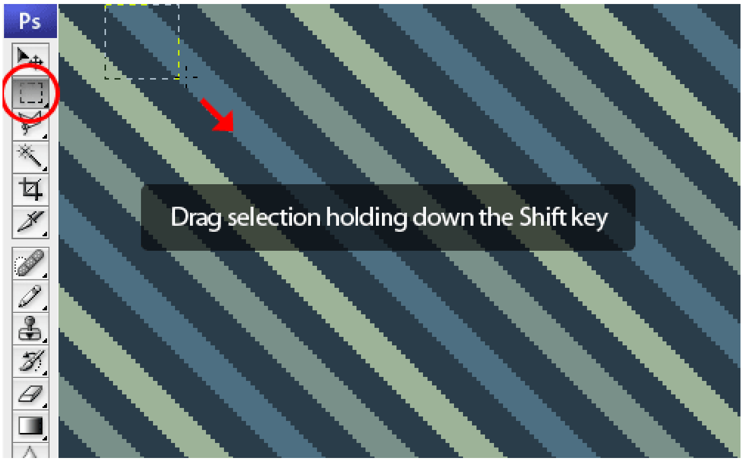
When the design is finished and you’re about to start the coding process, you have to prepare your graphics. If you’re working with a pattern background you need to export only the repeating fragment. It can be a bit tricky to isolate a fragment to achieve a seamless pattern background. For geometric patterns there is a method I always follow and that I want to share with you. Take for example a perfect 45° diagonal line pattern.



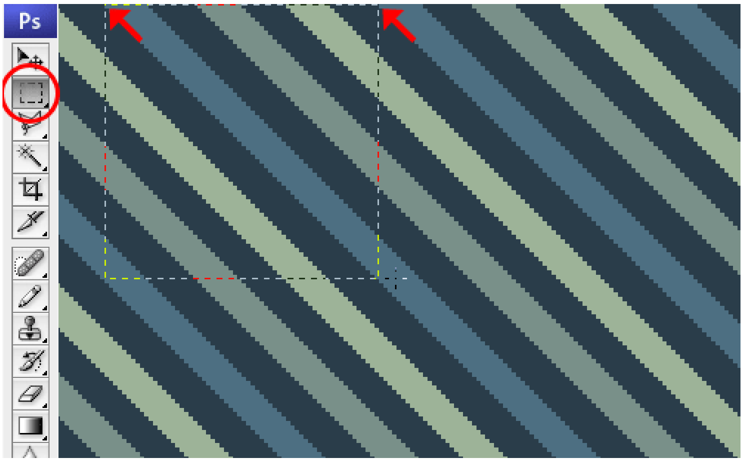
How do you define this pattern fragment so it will be rendered seamlessly?



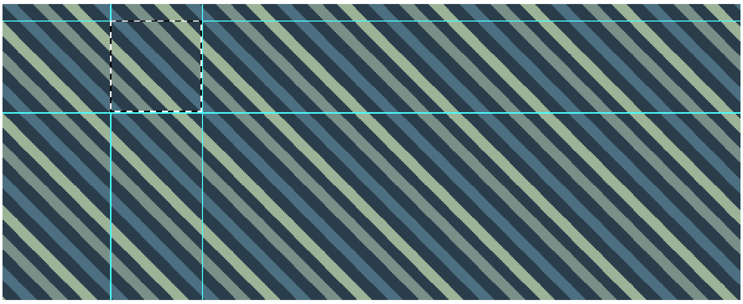
Here is the method I usually follow to avoid a mismatch. First, zoom in so you see enough detail and you can distinguish the pixels. Select the Rectangular Marquee Selection tool and start your selection at the intersection of 2 different colors of a diagonal line. Hold down the Shift key while dragging so you drag a perfect square.



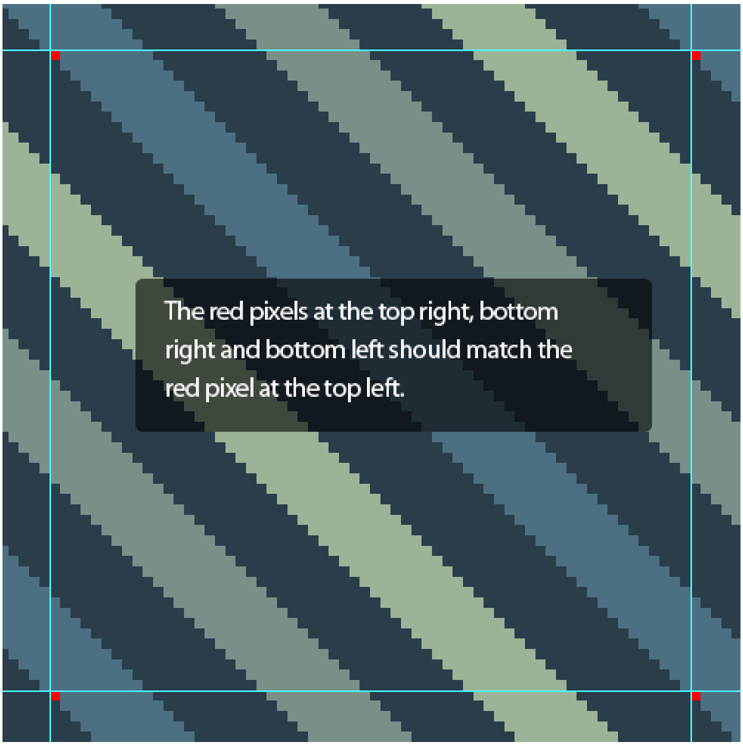
Release the mouse when you reach the exact same intesection (as your starting) point at the top right.



Copy this fragment (using *Copy Merged: Cmd/Ctrl + Shift + C*) and paste the fragment in a new layer. Give this layer the name 'pattern'. Now hold down the *Command Key* (*Control Key on Windows*) and click on the 'pattern' layer in the Layers Palette to select the fragment. Now go to *Edit > Define Pattern*, enter a name for your pattern and click OK. Test your pattern in a new document. Create a new document of 600 px by 400px, hit *Cmd/Ctrl + A* and go to *Edit > Fill…* and choose your pattern. If the result is OK, you have created a perfect pattern fragment.



Below you see this pattern enlarged. The guides show the boundaries of the pattern fragment and the red pixels are the reference points. The red pixels at the top right, bottom right and bottom left should match the red pixel at the top left.

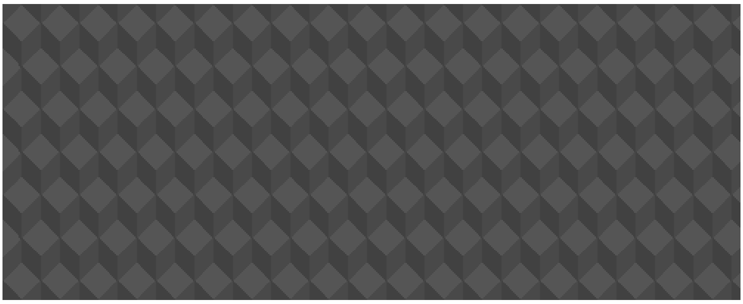

The red pixels at the top right, bottom right and bottom left should match the red pixel at the top left.

This technique should work for every geometric pattern. Some patterns are easier than others, but this, and the Photoshop pattern fill test, has always been my guideline.

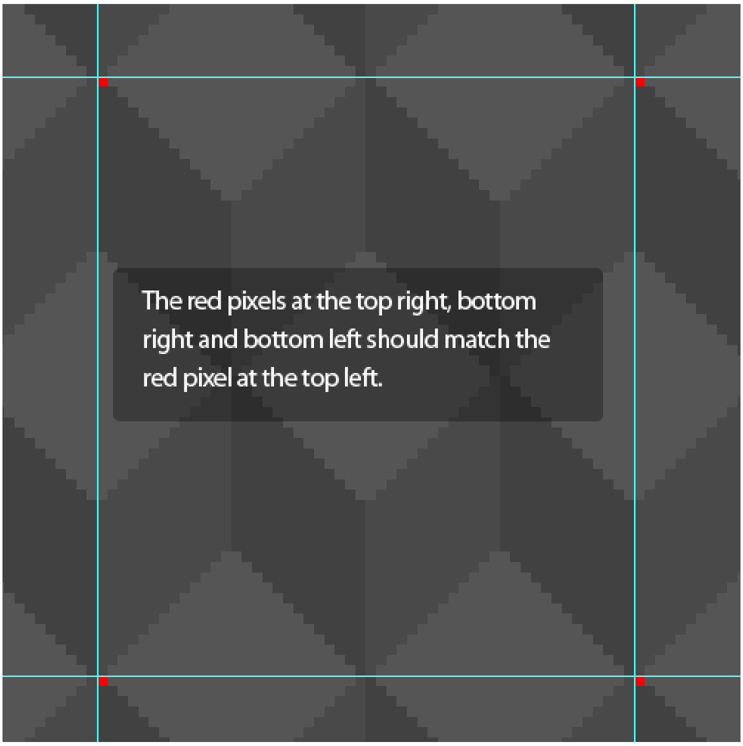
OTHER GEOMETRIC PATTERN EXAMPLES

Example 1

Not all geometric pattern fragments are squares. Some patterns look easy at first sight, because they look very repetitive, but they can be a bit tricky.

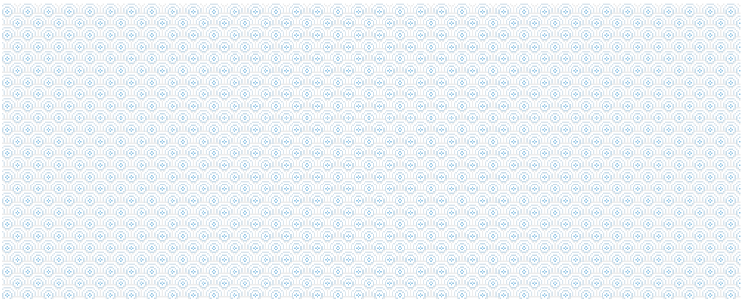


Zoomed in pattern fragment with point of reference shown:

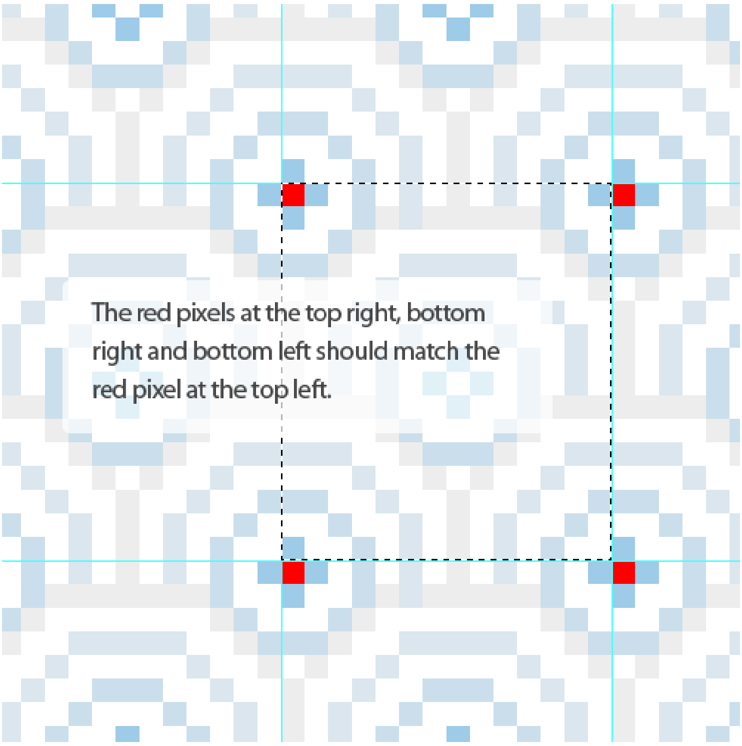

The red pixels at the top right, bottom right and bottom left should match the red pixel at the top left.

Example 2

Some patterns have a clear repeating point that can guide you, such as the blue small circle of this pattern as you can see from this zoomed in screenshot:

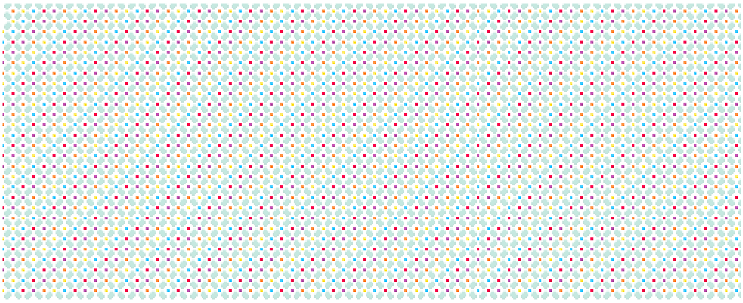


Zoomed in pattern fragment with point of reference shown:

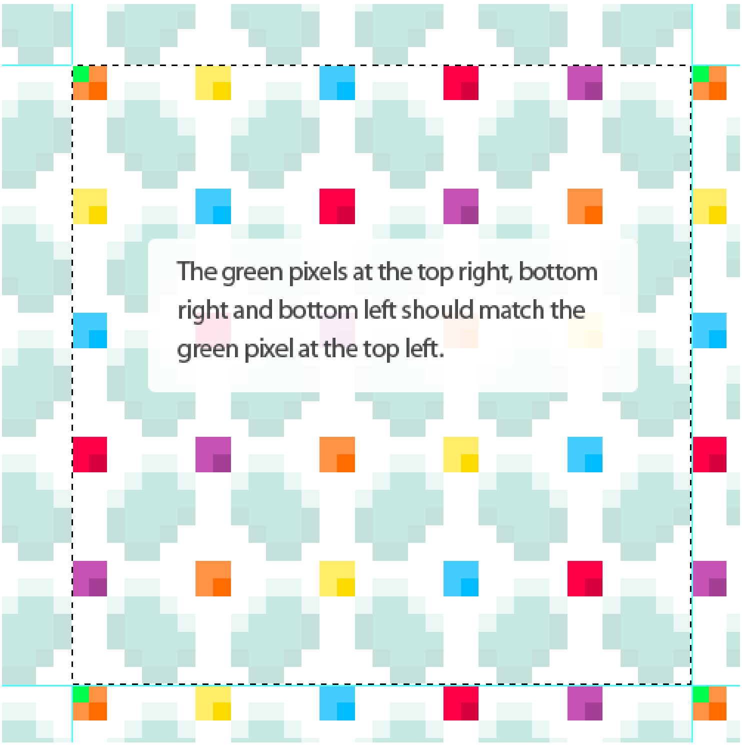

The red pixels at the top right, bottom right and bottom left should match the red pixel at the top left.

Example 3

The different diagonal colors makes a bit more tricky to extract the correct pattern fragment.



The orange dot, which is the starting point of the selection is captured a few times inside the fragment selection:


The green pixels at the top right, bottom right and bottom left should match the green pixel at the top left.

ABOUT THE AUTHOR

Veerle Pieters is a graphic/web designer based in Deinze, Belgium. Starting in '92 as a freelance graphic designer, Veerle worked on print design before focussing more on webdesign and GUI (since '96). She runs her own design studio Duoh! together with Geert Leyseele. Veerle has been blogging since 2003 and is considered number 39 on the list of "NxE's Fifty Most Influential 'Female' Bloggers".

3. User Styling

Jon Hicks 24ways.org/200803

During the recent US elections, Twitter decided to add an ‘election bar’ as part of their site design. You could close it if it annoyed you, but the action wasn’t persistent and the bar would always come back like a bad penny.

The solution to common browsing problems like this is CSS. ‘User styling’ (or the creepy ‘skinning’) is the creation of CSS rules to customise and personalise a particular domain. Aside from hiding adverts and other annoyances, there are many reasons for taking the time and effort to do it:

- Improving personal readability by changing text size and colour
- Personalising the look of a web app like GMail to look less insipid
- Revealing microformats

- Sport! My dreams of site skinning tennis are not yet fully realised, but it'll be all the rage by next Christmas, believe me.

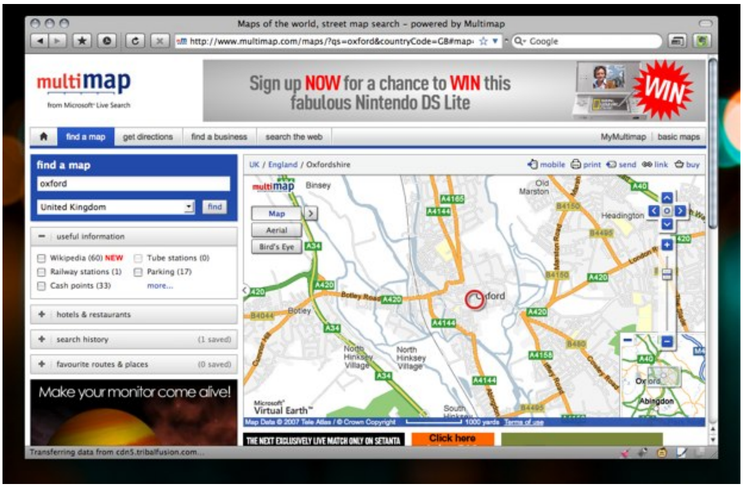
Hopefully you're now asking "But how? HOW?!". The process of creating a site skin is roughly as follows:

1. See something you want to change
2. Find out what it's called, and if any rules already apply to it
3. Write CSS rule(s) to override and/or enhance it.
4. Apply the rules

So let's get stuck in…

See something

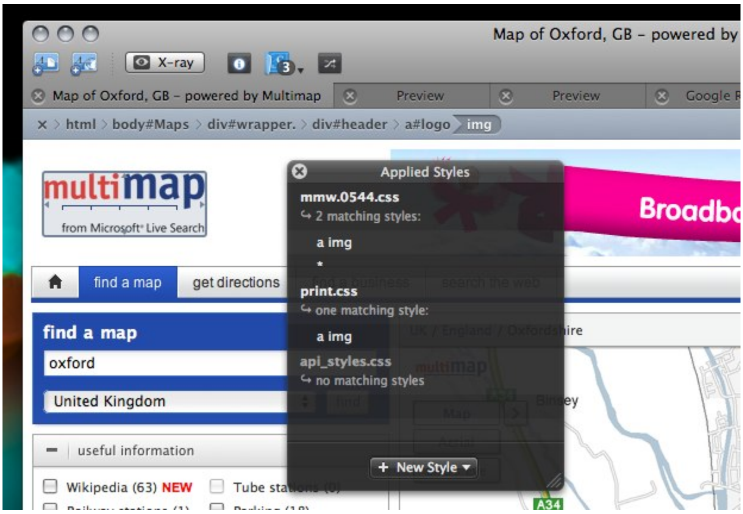
Let's start small with Multimap.com. Look at that big header – it takes up an awful lot of screen space doesn't it?


Maps of the world, street map search - powered by Multimap
multimap
Sign up NOW for a chance to WIN this fabulous Nintendo DS Lite
find a map
get directions
find a business
search the web
Make your monitor come alive!

No matter, we can fix it.

TOOLS

Now we need to find out where that big assed header is in the DOM, and make overriding CSS rules. The best tool I've found yet is the Mac OS X app, CSS Edit. It utilises a slick 'override stylesheets' function and DOM Inspector. Rather than give you all the usual DOM inspection tools, CSS Edit's is solely concerned with style. Go into 'X-Ray' mode, click an element, and look at the inspector window to see every style rule governing it. Click the selector to be taken to where it lives in the CSS. It really is a user styling dream app.



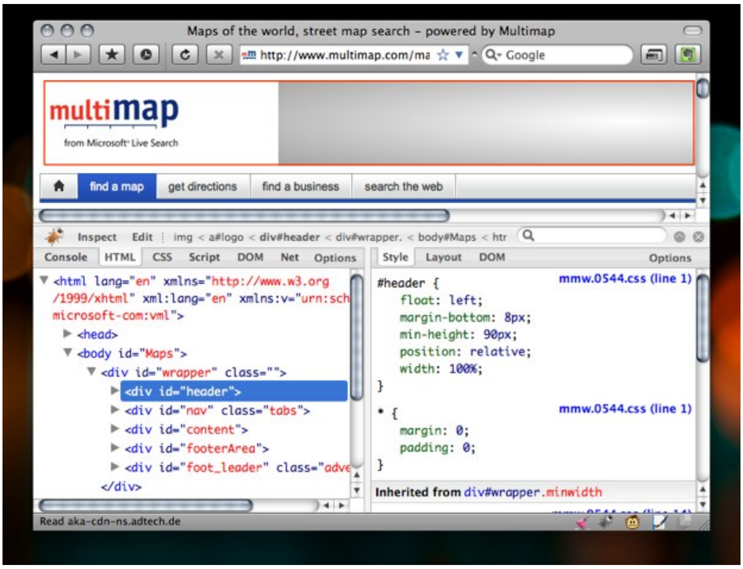
Applied Styles

mmw.0544.css
↪ 2 matching styles:
a img
*

print.css
↪ one matching style:
a img

api_styles.css
↪ no matching styles

Having said all that, you **can** achieve all this with free, cross platform tools – namely Firefox with the Firebug and Stylish extensions. We'll be using them for these examples, so make sure you have them installed if you want to follow along.


Maps of the world, street map search - powered by Multimap
http://www.multimap.com/ma
Google
multimap
from Microsoft Live Search
find a map
get directions
find a business
search the web
Inspect
Edit
img < a#logo < div#header < div#wrapper. < body#Maps < htr
Console
HTML
CSS
Script
DOM
Net
Options
Style
Layout
DOM
Options
Read aka-cdn-ns.adtech.de

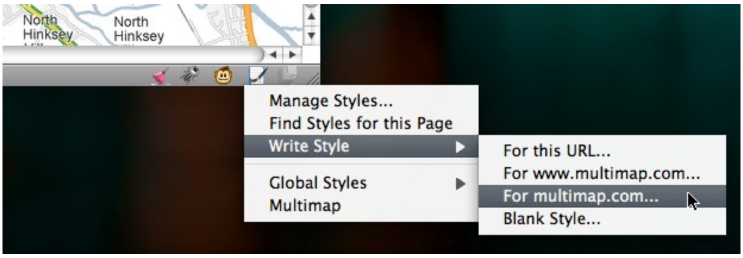
```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" xmlns:v="urn:sch
microsoft-com:vml">
  <head>
  <body id="Maps">
    <div id="wrapper" class="">
      <div id="header">
      <div id="nav" class="tabs">
      <div id="content">
      <div id="footerArea">
      <div id="foot_leader" class="adve
    </div>
```

```
#header {                                mmw.0544.css (line 1)
    float: left;
    margin-bottom: 8px;
    min-height: 90px;
    position: relative;
    width: 100%;
}

* {                                      mmw.0544.css (line 1)
    margin: 0;
    padding: 0;
}

Inherited from div#wrapper.minwidth
```

Using Firebug, we can see that the page is very helpfully marked up, and that whole top area is simply a div with an ID of **header**.

Change Something

When you installed Stylish, it added a page and brush icon to your status bar. Click on that, and choose Write Style > for Multimap.com. The other options allow you to only create a style for a particular part of a website or URL, but we want this to apply to the whole of Multimap:

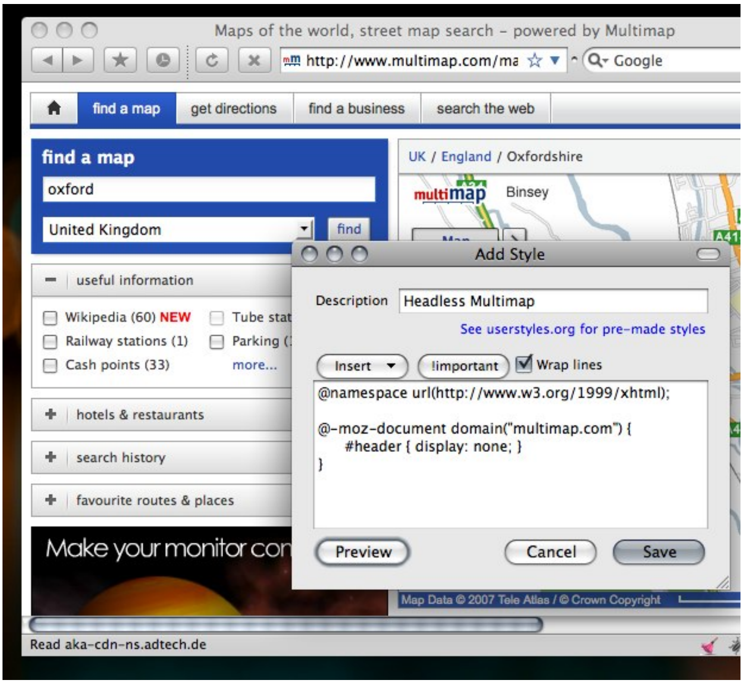


The 'Add Style' window then pops up, with the @-moz-document query at the top:

```
@namespace url(http://www.w3.org/1999/xhtml);
@-moz-document domain("multimap.com") {
}
```

All you need to do is add the CSS to hide the header, in between the curly brackets.

```
@namespace url(http://www.w3.org/1999/xhtml);
@-moz-document domain("multimap.com") {
        #header {display: none;}
}
```



Description: Headless Multimap

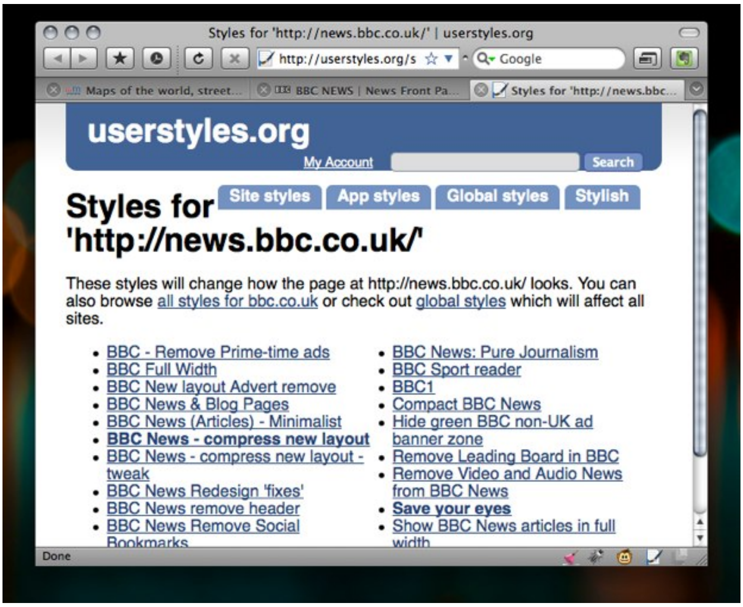
Wrap lines [✓]

```
@namespace url(http://www.w3.org/1999/xhtml);

@-moz-document domain("multimap.com") {
    #header { display: none; }
}
```

A click of the preview button shows us that it's worked! Now the map appears further up the page. The ethics of hiding adverts is a discussion for another time, but let's face it, when did you last whoop at the sight of a banner?

Make Something Better

If we're happy with our modifications, all we need to do is give it a name and save. Whenever you visit Multimap.com, the style will be available. Stylish also allows you to toggle a style on/off via the status bar menu. If you feel you want to share this style with the world, then userstyles.org is the place to do it. It’s a grand repository of customisations that Stylish connects with. Whenever you visit a site, you can see if anyone else has written a style for it, again, via the status bar menu “Find Styles for this Page”. Selecting this with “BBC News” shows that there are plenty of options, ranging from small layout tweaks to redesigns:


Styles for 'http://news.bbc.co.uk/' | userstyles.org
http://userstyles.org/s
Google
Maps of the world, street...
BBC NEWS | News Front Pa...
Styles for 'http://news.bbc...
userstyles.org
My Account
Search
Site styles
App styles
Global styles
Stylish
Done

Styles for 'http://news.bbc.co.uk/'

These styles will change how the page at http://news.bbc.co.uk/ looks. You can also browse all styles for bbc.co.uk or check out global styles which will affect all sites.

- BBC - Remove Prime-time ads
- BBC Full Width
- BBC New layout Advert remove
- BBC News & Blog Pages
- BBC News (Articles) - Minimalist
- **BBC News - compress new layout**
- BBC News - compress new layout - tweak
- BBC News Redesign 'fixes'
- BBC News remove header
- BBC News Remove Social Bookmarks
- BBC News: Pure Journalism
- BBC Sport reader
- BBC1
- Compact BBC News
- Hide green BBC non-UK ad banner zone
- Remove Leading Board in BBC
- Remove Video and Audio News from BBC News
- **Save your eyes**
- Show BBC News articles in full width

What’s more, whenever a style is updated, Stylish will notify you, and offer a one-click process to update it. This does only work in Firefox and Flock, so I’ll cover ways of applying site styles to other browsers later.

Specific Techniques

IMPORTANT!

In the Multimap example there wasn't a `display` specified on that element, but it isn't always going to be that easy. You may have spent most of your CSS life being a good designer and not resorting to adding `!important` to give your rule priority. There's no way to avoid this in user styling – if you're overriding an existing rule it's a necessity! Be prepared to be typing `!important` a lot.

STAR SELECTOR

The Universal Selector is a particularly useful way to start a style. For example, if we want to make Flickr use Helvetica before Arial (as they should've done!), we can cover all occurrences with just one rule:

```
* {font-family: "Helvetica Neue", Helvetica, sans-serif
!important;}
```

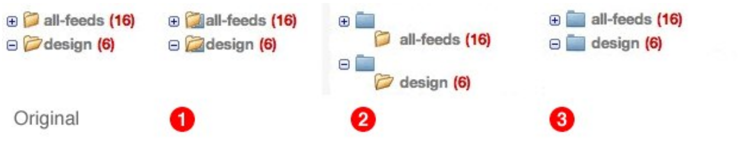
You can also use it to select 'everything within an element', by placing it after the element name:

```
#content * {font-family: "Helvetica Neue", Helvetica,
sans-serif !important;}
```

SWAPPING IMAGES

If you're changing something a little more complex, such as Google Reader, then at some point you'll probably want to change an `<img>`. The technique for replacing an image involves:

1. making your replacement image the background of the `<img>` tag
2. adding padding top and left to the size of you image to push the 'top' image away
3. making the height and width zero.



The old image is then pushed out of the way and hidden from view, allowing the replacement in the background to be revealed. Targeting the image may require using an attribute selector:

```
img[src="/reader/ui/
3544433079-tree-view-folder-open.gif"] {
  padding: 16px 0 0 16px;
  width: 0 !important;
  height: 0 !important;
  background-image: url(data:image/
png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYA
AAAf8/
9hAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxIB0t1+/AAAA
Bx0RVh0U29mdHdhcmUAQWRvYmUgRmlyZXdvcmtzIENTM5jWRgMAAAAVdE
VYdENyZWF0aW9uIFRpbWUAMjkvNi8wOJJ/
BVgAAAG3SURBVDiNpZIhb5RBEIaf
2W+vpIagIITSBIHBgsGjEYQaFLYShcITDL+ABIPnh4BFN0GQNFA4Cnf3fbszL2L3
jiuEVLDJbCazu8+8Mzsmif9ZBvDy7bvXlni0HRe8eXL/
zuPzABng62J5kFKaAQS
QgJAOgHMB9vDZq+d71689Hcyw9LfAZAYdioE10VSJo6OPL/
KNvSuHD+7dhU
0vHEsDUUWJChIlYJIjFx5BuMB2mJY/DnMoOJl/
R147oBUR0QAm8LAGCOEh3IO
ULiAl8jSOy/
nPetGsbGRKjktEiBCEHMlQj4loCuu4zCXCi4lUHTNDtGqEiACTqAFSI
OgAUAKv4bkWVy2g6tAbJtGy0TNugM3HADmlurKH27dVZSecxjboXggiAsMItR
h99wTILdewYRpXVJWtY85k7fPW8e1GpJFJacgesXs6VYYomz9G2yDhwPB7NEB
BDAMK7WYJlisYVBCpfaJBeB+eocFyVyAgCaoMCTJSTOOCWSyILrAnaXpSexRsx
GGAZ0AR+XT+5fjzyfwSpnUB/1w64xizVI/
t6q3b+58+vJ96mWtLf9haxNoc8M
v7N3d+AT4XPcFIxghoAAAAAElFTkSuQmCC) no-repeat !important;
}
```

Woah boy! What was all that gubbins in the background-image? It was a Data URI, and you can create these easily with Hixie's online tool. It's simply the image translated into text so that it can be embedded in the CSS, cutting down on the number of http requests. It's also a necessity with Mozilla browsers, as they don't allow user CSS to reference images stored locally. Converting images to URI's avoids this, as well as making a style easily portable – no images folder to pass around.

Don't forget all your other CSS techniques at your disposal: inserting your own content with `:before` and `:after` pseudo classes, make elements semi-transparent with `opacity` and round box corners without hacking . You can have fun, and for once, enjoy the freedom of not worrying about IE!

User styling without Stylish

Instead of using the Stylish extension, you can add rules to the userContent.css file, or use @import in that file to load a separate stylesheet. You can find this is in /Library/ Application Support/Camino/chrome/ on OS X, or C/ Program Files/Mozilla Firefox/Chrome on Windows. This is only way to apply user styles in Camino, but what about other browsers?

OPERA & OMNIWEB:

Both allow you to specify a custom CSS file as part of the site's preferences. Opera also allows custom javascript, using the same syntax as Greasemonkey scripts (more on that below)

SAFARI

There are a few options here: the PithHelmet and SafariStand haxies both allow custom stylesheets, or alternatively, a Greasemonkey style user script can employed via GreaseKit. The latter is my favoured solution on my Helvetireader theme, as it can allow for more prescriptive domain rules, just like the Mozilla @-moz-document method. User scripts are also the solution supported by the widest range of browsers.

WHAT NOW?

Hopefully I've given you enough information for you to be able start making your own styles. If you want to go straight in and tackle the 'Holy Grail', then off with you to GMail – I get more requests to theme that than anything else!

If you're a site author and want to encourage this sort of tom foolery, a good way is to provide a unique class or ID name with the body tag:

```
<body id="journal" class="hicksdesign-co-uk">
```

This makes it very easy to write rules that only apply to that particular site. If you wanted to use Safari without any of the haxies mentioned above, this method means you can include rules in a general CSS file (chosen via Preferences > Advanced > Stylesheet) without affecting other sites.

One final revelation on user styling – it's not just for web sites. You can tweak the UI of Firefox itself with the userChrome.css. You'll need to use the in-built DOM

Inspector instead of Firebug to inspect the window chrome, instead of a page. Great if you want to make small tweaks (changing the size of tab text for example) without creating a full blown theme.

ABOUT THE AUTHOR

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

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

4. Sitewide Search On A Shoe String

Christian Heilmann 24ways.org/200804

One of the questions I got a lot when I was building web sites for smaller businesses was if I could create a search engine for their site. Visitors should be able to search only this site and find things without the maintainer having to put "related articles" or "featured content" links on every page by hand.

Back when this was all fields this wasn't easy as you either had to write your own scraping tool, use ht://dig or a paid service from providers like Yahoo, Altavista or later on Google. In the former case you had to swallow the bitter pill of computing and indexing all your content and storing it in a database for quick access and in the latter it hurt your wallet.

Times have moved on and nowadays you can have the same functionality for free using Yahoo's "Build your own search service" – BOSS. The cool thing about BOSS is that it allows for a massive amount of hits a day and you can mash up the returned data in any format you want. Another good feature of it is that it comes with JSON-P as an output format which makes it possible to use it without any server-side component!

Starting with a working HTML form

In order to add a search to your site, you start with a simple HTML form which you can use without JavaScript. Most search engines will allow you to filter results by domain. In this case we will search “bbc.co.uk”. If you use Yahoo as your standard search, this could be:

```
<form id="customsearch" action="http://search.yahoo.com/
search">
  <div>
    <label for="p">Search this site:</label>
    <input type="text" name="p" id="term">
    <input type="hidden" name="vs" id="site"
value="bbc.co.uk">
    <input type="submit" value="go">
  </div>
</form>
```

The Google equivalent is:

```
<form id="customsearch" action="http://www.google.co.uk/
search">
  <div>
    <label for="p">Search this site:</label>
    <input type="text" name="as_q" id="term">
    <input type="hidden" name="as_sitesearch" id="site"
```

```
value="bbc.co.uk">
    <input type="submit" value="go">
  </div>
</form>
```

In any case make sure to use the ID `term` for the search term and `site` for the site, as this is what we are going to use for the script. To make things easier, also have an ID called `customsearch` on the form.

To use BOSS, you should get your own developer API for BOSS and replace the one in the demo code. There is click tracking on the search results to see how successful your app is, so you should make it your own.

Adding the BOSS magic

BOSS is a REST API, meaning you can use it in any HTTP request or in a browser by simply adding the right parameters to a URL. Say for example you want to search "bbc.co.uk" for "christmas" all you need to do is open the following URL:

http://boss.yahooapis.com/ysearch/web/v1/
christmas?sites=bbc.co.uk&format=xml&appid=YOUR-APPLICATION
ID

Try it out and click it to see the results in XML. We don't want XML though, which is why we get rid of the `format=xml` parameter which gives us the same information in JSON:

http://boss.yahooapis.com/ysearch/web/v1/
christmas?sites=bbc.co.uk&appid=YOUR-APPLICATION-ID

JSON makes most sense when you can send the output to a function and immediately use it. For this to happen all you need is to add a `callback` parameter and the JSON will be wrapped in a function call. Say for example we want to call `SITESEARCH.found()` when the data was retrieved we can do it this way:

http://boss.yahooapis.com/ysearch/web/v1/
christmas?sites=bbc.co.uk&callback=SITESEARCH.found&appid=YOU
ID

You can use this immediately in a script node if you want to. The following code would display the total amount of search results for the term `christmas` on `bbc.co.uk` as an alert:

```
<script type="text/javascript">
  var SITESEARCH = {};
  SITESEARCH.found = function(o){
    alert(o.ysearchresponse.totalhits);
  }
</script>
<script type="text/javascript"
src="http://boss.yahooapis.com/ysearch/web/v1/
christmas?sites=bbc.co.uk&callback=SITESEARCH.found&appid=Kzv_lcHV34HI
</script>
```

However, for our example, we need to be a bit more clever with this.

Enhancing the search form

Here's the script that enhances a search form to show results below it.

```
SITESEARCH = function(){
  var config = {
    IDs:{
      searchForm:'customsearch',
      term:'term',
      site:'site'
    },
    loading:'Loading results...',
    noresults:'No results found.',
    appID:'YOUR-APP-ID',
    results:20
  };
  var form;
  var out;
  function init(){
    if(config.appID === 'YOUR-APP-ID'){
      alert('Please get a real application ID!');
    } else {
      form =
document.getElementById(config.IDs.searchForm);
      if(form){
        form.onsubmit = function(){
          var site =
document.getElementById(config.IDs.site).value;
          var term =
document.getElementById(config.IDs.term).value;
          if(typeof site === 'string' && typeof term ===
'string'){
            if(typeof out !== 'undefined'){
              out.parentNode.removeChild(out);
            }
            out = document.createElement('p');

out.appendChild(document.createTextNode(config.loading));
            form.appendChild(out);
            var APIurl = 'http://boss.yahooapis.com/
ysearch/web/v1/' +
                          term +
'?callback=SITESEARCH.found&sites=' +
                          site + '&count=' +
config.results +
                          '&appid=' + config.appID;
            var s = document.createElement('script');
            s.setAttribute('src',APIurl);
            s.setAttribute('type','text/javascript');

document.getElementsByTagName('head')[0].appendChild(s);
            return false;
          }
        };
      }
    }
  };
  function found(o){
    var list = document.createElement('ul');
    var results = o.ysearchresponse.resultset_web;
    if(results){
      var item,link,description;
      for(var i=0,j=results.length;i<j;i++){
        item = document.createElement('li');
        link = document.createElement('a');
        link.setAttribute('href',results[i].clickurl);
        link.innerHTML = results[i].title;
        item.appendChild(link);
        description = document.createElement('p');
        description.innerHTML = results[i]['abstract'];
        item.appendChild(description);
        list.appendChild(item);
      }
    } else {
      list = document.createElement('p');

list.appendChild(document.createTextNode(config.noresults));
    }
    form.replaceChild(list,out);
    out = list;
  };
  return{
    config:config,
    init:init,
    found:found
  };
}();
```

Oooohhhh scary code! Let's go through this one bit at a time:

We start by creating a module called `SITESEARCH` and give it an configuration object:

```
SITESEARCH = function(){
  var config = {
    IDs:{
      searchForm:'customsearch',
      term:'term',
      site:'site'
    },
    loading:'Loading results...',
    appID:'YOUR-APP-ID',
    results:20
  }
```

Configuration objects are a great idea to make your code easy to change and also to override. In this case you can define different IDs than the one agreed upon earlier, define a message to show when the results are loading, when there aren't any results, the application ID and the number of results that should be displayed.

Note: you need to replace "YOUR-APP-ID" with the real ID you retrieved from BOSS, otherwise the script will complain!

```
var form;
var out;
function init(){
  if(config.appID === 'YOUR-APP-ID'){
    alert('Please get a real application ID!');
  } else {
```

We define `form` and `out` as variables to make sure that all the methods in the module have access to them. We then check if there was a real application ID defined. If there wasn't, the script complains and that's that.

```
form = document.getElementById(config.IDs.searchForm);
if(form){
  form.onsubmit = function(){
    var site =
document.getElementById(config.IDs.site).value;
    var term =
document.getElementById(config.IDs.term).value;
    if(typeof site === 'string' && typeof term ===
'string'){
```

If the application ID was a winner, we check if the form with the provided ID exists and apply an onsubmit event handler. The first thing we get is the values of the site we want to search in and the term that was entered and check that those are strings.

```
if(typeof out !== 'undefined'){
  out.parentNode.removeChild(out);
}
out = document.createElement('p');
out.appendChild(document.createTextNode(config.loading));
form.appendChild(out);
```

If both are strings we check of `out` is undefined. We will create a loading message and subsequently the list of search results later on and store them in this variable. So if `out` is defined, it'll be an old version of a search (as users will re-submit the form over and over again) and we need to remove that old version.

We then create a paragraph with the loading message and append it to the form.

```
var APIurl = 'http://boss.yahooapis.com/ysearch/web/v1/'
+
                        term +
'?callback=SITESEARCH.found&sites=' +
                        site + '&count=' +
config.results +
                        '&appid=' + config.appID;
          var s = document.createElement('script');
          s.setAttribute('src',APIurl);
          s.setAttribute('type','text/javascript');

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

Now it is time to call the BOSS API by assembling a correct REST URL, create a script node and apply it to the head of the document. We return `false` to ensure the form does not get submitted as we want to stay on the page.

Notice that we are using `SITESEARCH.found` as the callback method, which means that we need to define this one to deal with the data returned by the API.

```
function found(o){
  var list = document.createElement('ul');
  var results = o.ysearchresponse.resultset_web;
  if(results){
    var item,link,description;
```

We create a new list and then get the `resultset_web` array from the data returned from the API. If there aren't any results returned, this array will not exist which is why we need to check for it. Once we done that we can define three variables to repeatedly store the item title we want to display, the link to point to and the description of the link.

```
for(var i=0,j=results.length;i<j;i++){
  item = document.createElement('li');
  link = document.createElement('a');
  link.setAttribute('href',results[i].clickurl);
  link.innerHTML = results[i].title;
  item.appendChild(link);
  description = document.createElement('p');
  description.innerHTML = results[i]['abstract'];
  item.appendChild(description);
  list.appendChild(item);
}
```

We then loop over the results array and assemble a list of results with the titles in links and paragraphs with the abstract of the site. Notice the bracket notation for abstract as `abstract` is a reserved word in JavaScript2 :).

```
} else {
    list = document.createElement('p');

list.appendChild(document.createTextNode(config.noresults));
  }
  form.replaceChild(list,out);
  out = list;
};
```

If there aren't any results, we define a paragraph with the no results message as `list`. In any case we replace the old `out` (the loading message) with the `list` and re-define `out` as the `list`.

```
return{
    config:config,
    init:init,
    found:found
  };
}();
```

All that is left to do is return the properties and methods we want to make public. In this case `found` needs to be public as it is accessed by the API return. We return `init` to make it accessible and `config` to allow implementers to override any of the properties.

Using the script

In order to use this script, all you need to do is to add it **after** the form in the document, override the API key with your own and call `init()`:

```
<form id="customsearch" action="http://search.yahoo.com/
search">
  <div>
    <label for="p">Search this site:</label>
    <input type="text" name="p" id="term">
    <input type="hidden" name="vs" id="site"
value="bbc.co.uk">
    <input type="submit" value="go">
  </div>
</form>
<script type="text/javascript"
src="boss-site-search.js"></script>
<script type="text/javascript">
  SITESEARCH.config.appID =
'copy-the-id-you-know-to-get-where';
  SITESEARCH.init();
</script>
```

Where to go from here

This is just a very simple example of what you can do with BOSS. You can define languages and regions, retrieve and display images and news and mix the results with other data sources before displaying them. One very cool feature is that by adding a `view=keyterms` parameter to the URL you can get the keywords of each of the results to drill deeper into the search. An example for this written in PHP is available on the YDN blog. For JavaScript solutions there is a handy wrapper called yboss available to help you go nuts.

AVAILABLE IN GERMAN
webkrauts.de

ABOUT THE AUTHOR

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

5. Art Directing with Looking Room

Mark Boulton 24ways.org/200805

Using photographic composition techniques to start to art direct on the template-driven web.

Think back to last night. There you are, settled down in front of the TV, watching your favourite soap opera, with nice hot cup of tea in hand. Did you notice – whilst engrossed in the latest love-triangle – that the cameraman has worked very hard to support your eye's natural movement on-screen? He's carefully framed individual shots to create balance.

Think back to last week. There you were, sat with your mates watching the big match. Did you notice that the cameraman frames the shot to go with the direction of play? A player moving right will always be framed so that he is on the far left, with plenty of 'room' to run into.

Both of these cameramen use a technique called Looking Room, sometimes called Lead Room. Looking Room is the space between the subject (be it a football, or a face), and the edge of the screen. Specifically, Looking Room is the negative space on the side the subject is looking or moving. The great thing is, it's not just limited to photography, film or television; we can use it in web design too.

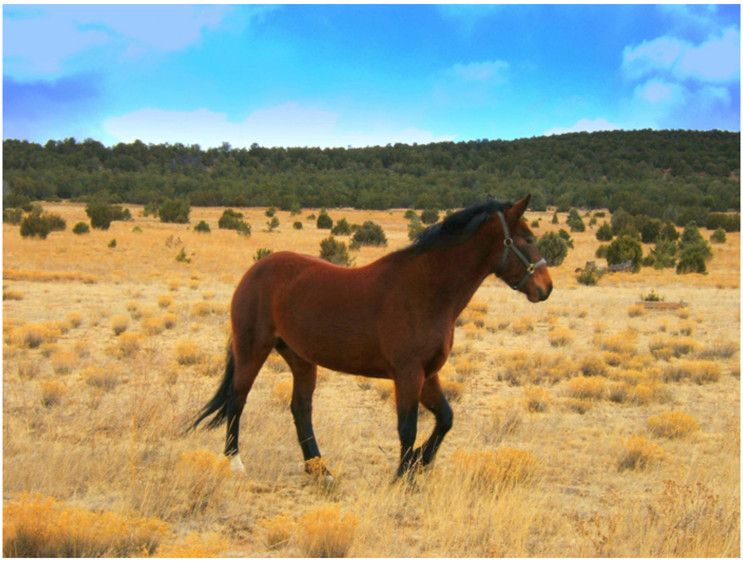
BASIC FRAMING

Before we get into Looking Room, and how it applies to web, we need to have a look at some basics of photographic composition.

Many web sites use imagery, or photographs, to enhance the content. But even with professionally shot photographs, without a basic understanding of framing or composition, you can damage how the image is perceived.

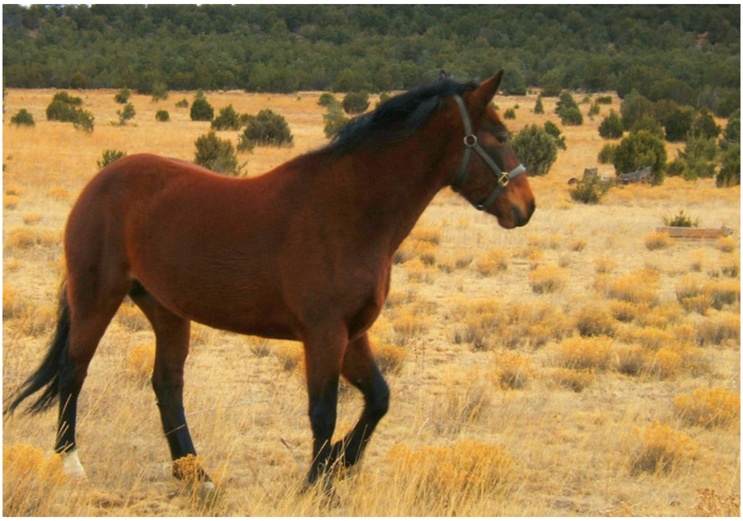
A simple, easy way to make photographs more interesting is to fill the frame.

Take this rather mundane photograph of a horse:



A typical point and click affair. But, we can work with this.

By closely cropping, and filling the frame, we can instantly change the mood of the shot.

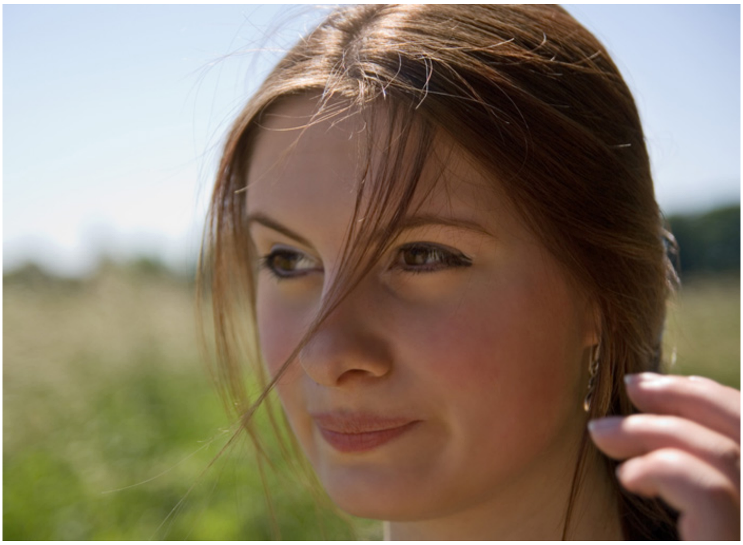


I’ve also added Looking Room on the right of the horse. This is space that the horse would be walking into. It gives the photograph movement.

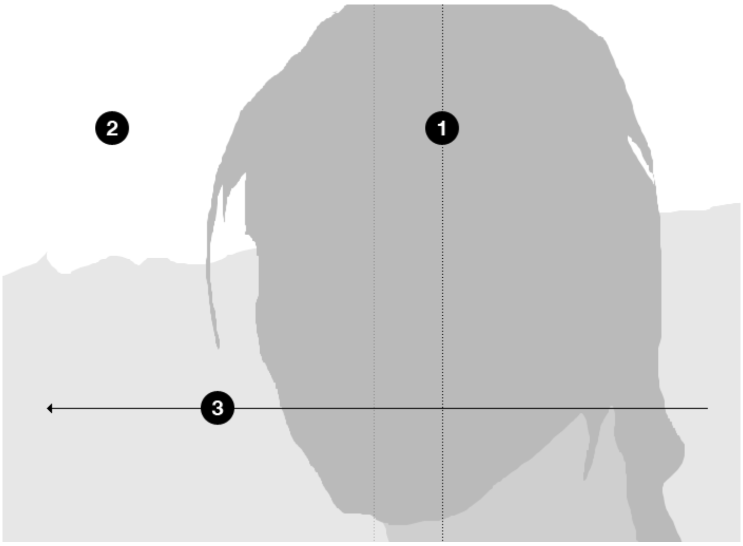
SUBJECT, SPACE, AND MOVEMENT

Generally speaking, a portrait photograph will have a subject and space around them. Visual interest in portrait photography can come from movement; how the eye moves around the shot. To get the eye moving, the photographer modifies the space around the subject.

Look at this portrait:



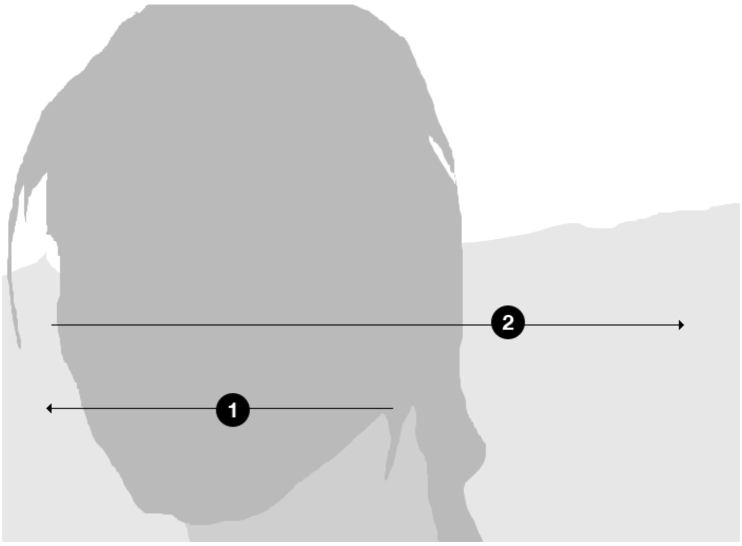
The photography has framed the subject on the right, allowing for whitespace, or Looking Room, in the direction the subject is looking. The framing of the subject (1), with the space to the left (2) – the Looking Room – creates movement, shown by the arrow (3).



Note the subject is not framed centrally (shown by the lighter dotted line).

If the photographer had framed the subject with equal space either side, the resulting composition is static, like our horse.

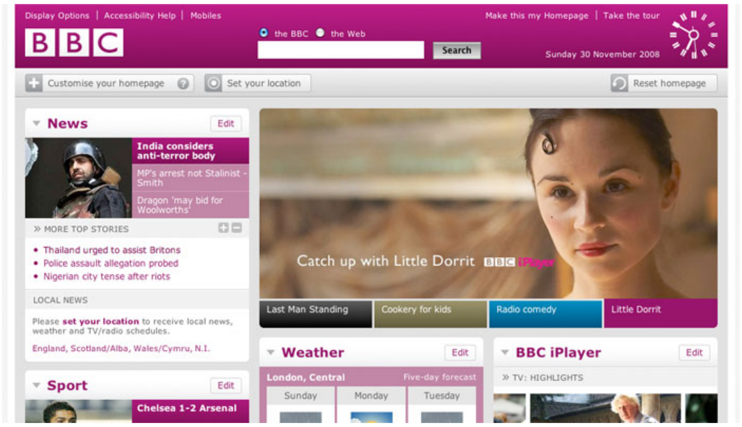
If the photographer framed the subject way over on the left, as she is looking that way, the resulting whitespace on the right leads a very uncomfortable composition.



The root of this discomfort is what the framing is telling our eye to do. The subject, looking to the left, suggests to us that we should do the same. However, the Looking Room on the right is telling our eye to occupy this space. The result is a confusing back and forth.

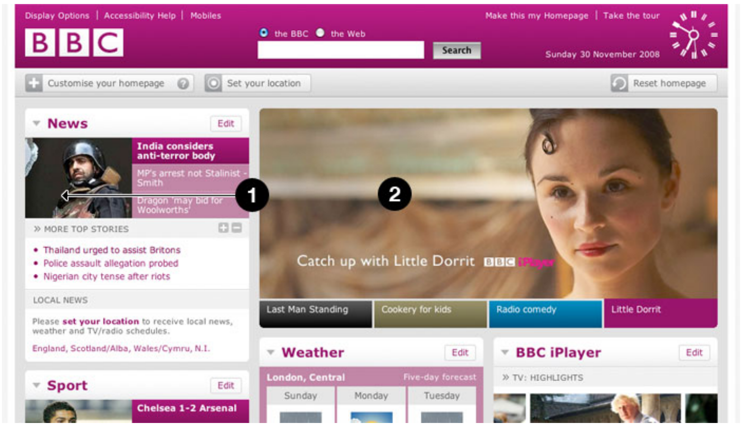
HOW LOOKING ROOM APPLIES TO THE WEB

We can apply the same theory to laying out a web page or application. Taking the three same elements – Subject, Space, and resulting Movement – we can guide a user's eye to the elements we need to. As designers, or content editors, framing photographs correctly can have a subtle but important effect on how a page is visually scanned. Take this example:


Display Options | Accessibility Help | Mobiles
Make this my Homepage | Take the tour
the BBC
the Web
Search
Sunday 30 November 2008
Customise your homepage
Set your location
Reset homepage
News
Edit
India considers anti-terror body
MP's arrest not Stalinist - Smith
Dragon 'may bid for Woolworths'
» MORE TOP STORIES
Thailand urged to assist Britons
Police assault allegation probed
Nigerian city tense after riots
LOCAL NEWS
Please set your location to receive local news, weather and TV/radio schedules.
England, Scotland/Alba, Wales/Cymru, N.I.
Sport
Edit
Chelsea 1-2 Arsenal
Catch up with Little Dorrit
Last Man Standing
Cookery for kids
Radio comedy
Little Dorrit
Weather
Edit
London, Central
Five-day forecast
Sunday
Monday
Tuesday
BBC iPlayer
Edit
» TV: HIGHLIGHTS

The BBC homepage uses great photography as a way of promoting content. Here, they have cropped the main photograph to guide the user's eye into the content.

By applying the same theory, the designer or content editor has applied considerable Looking Room (2) to the photograph to create balance with the overall page design, but also to create movement of the user's eye toward the content (1)


Display Options | Accessibility Help | Mobiles

the BBC the Web Search

Make this my Homepage | Take the tour

Sunday 30 November 2008

Customise your homepage Set your location Reset homepage

News Edit

India considers anti-terror body

MP's arrest not Stalinist - Smith

Dragon 'may bid for Woolworths'

» MORE TOP STORIES

- Thailand urged to assist Britons
- Police assault allegation probed
- Nigerian city tense after riots

LOCAL NEWS

Please **set your location** to receive local news, weather and TV/radio schedules.

England, Scotland/Alba, Wales/Cymru, N.I.

Sport Edit

Chelsea 1-2 Arsenal

Catch up with Little Dorrit BBC iPlayer

Last Man Standing | Cookery for kids | Radio comedy | Little Dorrit

Weather Edit

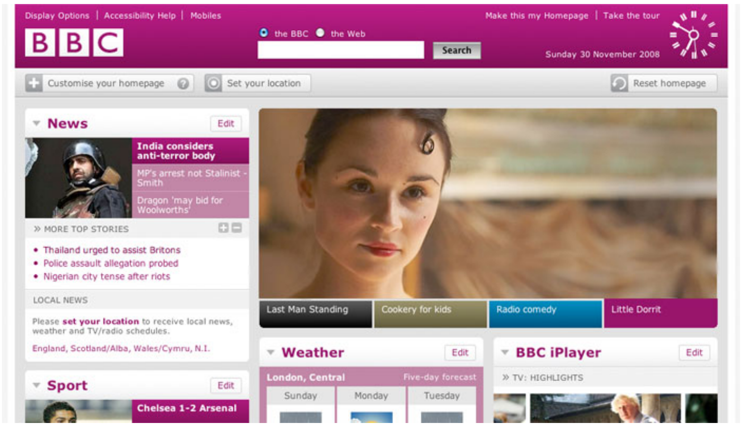
London, Central Five-day forecast

Sunday Monday Tuesday

BBC iPlayer Edit

» TV: HIGHLIGHTS

If the image was flipped horizontally. The Looking Room is now on the right. The subject of the photograph is looking off the page, drawing the user's eye away from the content. Once again, this results in a confusing back and forth as your eye fights its way over to the left of the page.


Display Options | Accessibility Help | Mobiles

the BBC the Web Search

Make this my Homepage | Take the tour

Sunday 30 November 2008

Customise your homepage Set your location Reset homepage

News Edit

India considers anti-terror body

MP's arrest not Stalinist - Smith

Dragon 'may bid for Woolworths'

» MORE TOP STORIES

- Thailand urged to assist Britons
- Police assault allegation probed
- Nigerian city tense after riots

LOCAL NEWS

Please **set your location** to receive local news, weather and TV/radio schedules.

England, Scotland/Alba, Wales/Cymru, N.I.

Sport Edit

Chelsea 1-2 Arsenal

Last Man Standing | Cookery for kids | Radio comedy | Little Dorrit

Weather Edit

London, Central Five-day forecast

Sunday Monday Tuesday

BBC iPlayer Edit

» TV: HIGHLIGHTS

A LITTLE BIT OF ART DIRECTION

Art Direction can be described as the act or process of managing the visual presentation of content. Art Direction is difficult to do on the web, because content and presentation are, more often than not, separated. But where there are images, and when we know the templates that those images will populate, we can go a little way to bridging the gap between content and presentation.

By understanding the value of framing and Looking Room, and the fact that it extends beyond just a good looking photograph, we can start to see photography playing more of an integral role in the communication of content.

We won't just be populating templates. We'll be art directing.

Photo credits:

- Portrait by Carsten Tolkmit
- Horse by Mike Pedroncelli

ABOUT THE AUTHOR

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

6. Using Google App Engine as Your Own Content Delivery Network

Matt Riggott 24ways.org/200806

Do you remember, years ago, when hosting was expensive, domain names were the province of the rich, and you hosted your web pages on Geocities? It seems odd to me now that there was a time when each and every geek didn't have his own top-level domain and super hosting setup. But as the parts became more and more affordable a man could become an outcast if he didn't have his own slightly surreal-sounding TLD.

And so it will be in the future when people realise with surprise there was a time before affordable content delivery networks.

A content delivery network, or CDN, is a system of servers spread around the world, serving files from the nearest physical location. Instead of waiting for a file to find its way from a server farm in Silicon Valley 8,000 kilometres away, I can receive it from London, Dublin, or Paris, cutting down the time I wait. The big names — Google, Yahoo, Amazon, et al — use CDNs for their sites, but they've always been far too expensive for us mere mortals. Until now.

There's a service out there ready for you to use as your very own CDN. You have the company's blessing, you won't need to write a line of code, and — best of all — it's free. The name? Google App Engine.

In this article you'll find out how to set up a CDN on Google App Engine. You'll get the development software running on your own computer, tell App Engine what files to serve, upload them to a web site, and give everyone round the world access to them.

CREATING YOUR FIRST GOOGLE APP ENGINE PROJECT

Before we do anything else, you'll need to download the Google App Engine software development kit (SDK). You'll need Python 2.5 too — you won't be writing any Python code but the App Engine SDK will need it to run on your computer. If you don't have Python, App Engine will install it for you (if you use Mac OS X 10.5 or a Linux-based OS you'll have Python; if you use Windows you won't).

Done that? Excellent, because that's the hardest step. The rest is plain sailing.

You'll need to choose a unique 'application id' — nothing more than a name — for your project. Make sure it consists only of lowercase letters and numbers. For this article I'll use `24ways2008`, but you can choose anything you like.

On your computer, create a folder named after your application id. This folder can be anywhere you want: your desktop, your documents folder, or wherever you usually keep your web files. Within your new folder, create a folder called `assets`, and within that folder create three folders called `images`, `css`, and `javascript`. These three folders are the ones you'll fill with files and serve from your content delivery network. You can have other folders too, if you like.

That will leave you with a folder structure like this:

```
24ways2008/
    assets/
      css/
      images/
      javascript/
```

Now you need to put a few files in these folders, so we can later see our CDN in action. You can put anything you want in these folders, but for this example we'll include an HTML file, a style sheet, an image, and a Javascript library.

In the top-level folder (the one I've called `24ways2008`), create a file called `index.html`. Fill this with any content you want. In the `assets/css` folder, create a file named `core.css` and throw in a couple of CSS rules for good measure. In the `assets/images` directory save any image that takes your fancy — I've used the silver badge from the App Engine download page. Finally, to fill the JavaScript folder, add in this jQuery library file. If you've got the time and the inclination, you can build a page that uses all these elements.

So now we should have a set of files and folders that look something like this:

```
24ways2008/
    assets/
        index.html
        css/
            core.css
        images/
            appengine-silver-120x30.gif
        javascript/
            jquery-1.2.6.min.js
```

Which leaves us with one last file to create. This is the important one: it tells App Engine what to do with your files. It's named `app.yaml`, it sits at the top-level (inside the folder I've named `24ways2008`), and it needs to include these lines:

```
application: 24ways2008
version: 1
runtime: python
api_version: 1

handlers:
- url: /
  static_files: assets/index.html
  upload: assets/index.html

- url: /
  static_dir: assets
```

You need to make sure you change `24ways2008` on the first line to whatever you chose as your application id, but otherwise the content of your `app.yaml` file should be identical. And with that, you've created your first App Engine project. If you want it, you can download a zip file containing my project.

TESTING YOUR PROJECT

As it stands, your project is ready to be uploaded to App Engine. But we couldn't call ourselves professionals if we didn't test it, could we? So, let's put that downloaded SDK to good use and run the project from your own computer.

One of the files you'll find App Engine installed is named `dev_appserver.py`, a Python script used to simulate App Engine on your computer. You'll find lots of information on how to do this in the documentation on the development web server, but it boils down to running the script like so (the space and the dot at the end are important):

```
dev_appserver.py .
```

You'll need to run this from the command-line: Mac users can run the Terminal application, Linux users can run their favourite shell, and Windows users will need to run it via the Command Prompt (open the Start menu, choose 'Run...', type '`cmd`', and click 'OK'). Before you run the script you'll need to make sure you're in the project folder — in my case, as I saved it to my desktop I can go there by typing

```
cd ~/Desktop/24ways2008
```

in my Mac's Terminal app; if you're using Windows you can type

```
cd "C:\Documents and Settings\username\Desktop\
24ways2008"
```

If that's successful, you'll see a few lines of output, the last looking something like this:

```
INFO     2008-11-22 14:35:00,830 dev_appserver_main.py]
Running application 24ways2008 on port 8080:
http://localhost:8080
```

Now you can power up your favourite browser, point it to `http://localhost:8080/`, and you'll see the page you saved as `index.html`. You'll also find your CSS file at `http://localhost:8080/css/core.css`. In fact, anything you put inside the assets folder in the project will be accessible from this domain. You're running our own App Engine web server!

Note that no-one else will be able to see your files: `localhost` is a special domain that you can only see from your computer — and once you stop the development server (by pressing Control–C) you'll not be able to see the files in your browser until you start it again.

You might notice a new file has turned up in your project: `index.yaml`. App Engine creates this file when you run the development server, and it's for internal App Engine use only. If you delete it there are no ill effects, but it will reappear when you next run the development server. If you're using version control (e.g. Subversion) there's no need to keep a copy in your repository.

So you've tested your project and you've seen it working on your own machine; now all you need to do is upload your project and the world will be able to see your files too.

UPLOADING YOUR PROJECT

If you don't have a Google account, create one and then sign in to App Engine. Tell Google about your new project by clicking on the 'Create an Application' button. Enter your application id, give the application a name, and agree to the terms and conditions. That's it. All we need do now is upload the files.

Open your Mac OS X Terminal, Windows Command Prompt, or Linux shell window again, move to the project folder, and type (again, the space and the dot at the end are important):

```
appcfg.py update .
```

Enter your email address and password when prompted, and let App Engine do it's thing. It'll take no more than a few seconds, but in that time App Engine will have done the equivalent of logging in to an FTP server and copying files across. It's fairly understated, but you now have your own project up and running. You can see mine at http://24ways2008.appspot.com/, and everyone can see yours at `http://your-application-id.appspot.com/`. Your files are being served up over Google's content delivery network, at no cost to you!

BENEFITS OF USING GOOGLE APP ENGINE

The benefits of App Engine as a CDN are obvious: your own server doesn't suck up the bandwidth, while your visitors will appreciate a faster site. But there are also less obvious benefits.

First, once you've set up your site, updating it is an absolute breeze. Each time you update a file (or a batch of files) you need only run `appcfg.py` to see the changes appear on your site. To paraphrase Joel Spolsky, a good web site must be able to be updated in a single step. Many designers and developers can't make that claim, but with App Engine, *you* can.

App Engine also allows multiple people to work on one application. If you want a friend to be able to upload files to your site you can let him do so without giving him usernames and passwords — all he needs is his own Google account. App Engine also gives you a log of all actions taken by collaborators, so you can see who's made updates, and when.

Another bonus is the simple version control App Engine offers. Do you remember the file named `app.yaml` you created a while back? The second line looked like this:

```
version: 1
```

If you change the version number to 2 (or 3, or 4, etc), App Engine will keep a copy of the last version you uploaded. If anything goes wrong with your latest version, you can tell App Engine to revert back to that last saved version. It's no proper version control system, but it could get you out of a sticky situation.

One last thing to note: if you're not happy using `your-application-id.appspot.com` as your domain, App Engine will quite happily use any domain you own.

THE WEAK POINTS OF GOOGLE APP ENGINE

In the right circumstances, App Engine can be a real boon. I run my own site using the method I've discussed above, and I'm very happy with it. But App Engine does have its disadvantages, most notably those discussed by Aral Balkan in his post 'Why Google App Engine is broken and what Google must do to fix it'.

Aral found the biggest problems while using App Engine as a web application platform; I wouldn't recommend using it as such either (at least for now) but for our purposes as a CDN for static files, it's much more worthy. Still, App Engine has two shortcomings you should be aware of.

The first is that you can't host a file larger than one megabyte. If you want to use App Engine to host that 4.3MB download for your latest-and-greatest desktop software, you're out of luck. The only solution is to stick to smaller files.

The second problem is the quota system. Google's own documentation says you're allowed 650,000 requests a day and 10,000 megabytes of bandwidth in and out (20,000 megabytes in total), which should be plenty for most sites. But people have seen sites shut down temporarily for breaching quotas — in some cases after inexplicable jumps in Google's server CPU usage. Aral, who's seen it happen to his own sites, seemed genuinely frustrated by this, and if you measure your hits in the hundreds of thousands and don't want to worry about uptime, App Engine isn't for you.

That said, for most of us, App Engine offers a fantastic resource: the ability to host files on Google's own content delivery network, at no charge.

CONCLUSION

If you've come this far, you've seen how to create a Google App Engine project and host your own files on Google's CDN. You've seen the great advantages App Engine offers — an excellent content delivery network, the ability to update your site with a single command, multiple authors, simple version control, and the use of your own domain — and you've come across some of its weaknesses — most importantly the limit on file sizes and the quota system. All that's left to do is upload those applications — but not before you've finished your Christmas shopping.

ABOUT THE AUTHOR

Matt Riggott is a web programmer and informatician living in Edinburgh, Scotland. When in geek mode he enjoys using Python and Django to help people do interesting things on the web. He volunteers as a technical advisor for the Sandbag Campaign and is available for freelance work.

When trying to fit in with normal people he enjoys taking photos, travelling, and following politics avidly.

7. How To Create Rockband'ism

Henriette Weber 24ways.org/200807

There are mysteries happening in the world of business these days. We want something else by now. The business of business has to become more than business. We want to be able to identify ourselves with the brands we purchase and we want them to do good things. We want to feel cool because we buy stuff, and we don't just want a shopping experience – we want an engagement with a company we can relate to.

Let me get back to "feeling cool" – if we want to feel cool, we might get the companies we buy from to support that. That's why I am on a mission to make companies into rockbands.

Now when I say rockbands – I don't mean the puke-y, drunky, nasty stuff that some people would highlight is also a part of rockbands. Therefore I have created my own word "rockband'ism". This word is the definition of a childhood dream version of being in a rockband – the feeling of being more respected and loved and cool, than a cockroach or a suit on the floor of a company.

ROCKBAND’ISM

Rockband’ism is what we aspire to, to feel cool and happy.

So basically what I am arguing is that companies should look upon themselves as rockbands. Because the world has changed, so business needs to change as well.

I have listed a couple of things you could do today to become a rockband, as a person or as a company.

1 – Give your support to companies that make a difference to their surroundings – if you are buying electronics look up what the electronic producers are doing of good in the world (check out the Greenpeace Guide to Greener Electronics).

2 – Implement good karma in your everyday life (and do well by doing good). What you give out you get back at some point in some shape – this can also be implemented for business.

3 – WWRD? – “what would a rockband do”? or if you are into Kenny Rogers – what would he do in any given situation? This will also show yourself where your business or personal integrity lies because you actually act as a person or a rockband you admire.

4 – Start leading instead of managing – If we can measure stuff why should we manage it? Leadership is key here instead of management. When you lead you tell people how to reach the stars, when you manage you keep them on the ground.

5 – Respect and confide in, that people are the best at what they do. If they aren't, they won't be around for long. If they are and you keep on buggin' them, they won't be around for long either.

6 – Don't be arrogant – Because audiences can't stand it – talk to people as a person not as a company.

7 – Focus on your return on involvement – know that you get a return on, what you involve yourself in. No matter if it's bingo, communities, talks, ornithology or un-conferences.

8 – Find out where you can make a difference and do it. Don't leave it up to everybody else to save the world.

9 – Find out what you can do to become an authentic, trustworthy and remarkable company. Maybe you could even think about this a lot and make these thoughts into an actionplan.

10 – Last but not least – if you're not happy – do something else, become another type of rockband, maybe a soloist of a sort, or an orchestra.

NO MORE BUSINESS AS USUAL

This really isn't time for more business as usual, our environment (digital, natural, work or any other kind of environment) is changing. You are going to have to change too.

This article actually sprang from a talk I did at the Shift08 conference in Lisbon in October. In addition to this article for 24 ways I have turned the talk into an eBook that you can get on Toothless Tiger Press for free.

May you all have a sustainable and great Christmas full of great moments with your loved ones. December is a month for gratitude, enjoyment and love.

ABOUT THE AUTHOR

Henriette Weber is the founder of Toothless Tiger where she thrives as a social business expert, specializing in marketing, branding, online presence, social media and communities.

In addition, she works with web trend spotting, e-commerce and online reputation management.

Her “formal” pitch is that “she helps companies use their online presence and strategies as a marketing tool” – but basically her hidden talent (and her real pitch) is “to stop companies from looking like (complete) asses online”.

8. The IE6 Equation

Jeremy Keith 24ways.org/200808

It is the destiny of one browser to serve as the nemesis of web developers everywhere. At the birth of the Web Standards movement, that role was played by Netscape Navigator 4; an outdated browser that refused to die. Its tenacious existence hampered the adoption of modern standards. Today that role is played by *Internet Explorer 6*.

There’s a sensation that I’m sure you’re familiar with. It’s a horrible mixture of dread and nervousness. It’s the feeling you get when—after working on a design for a while in a standards-compliant browser like Firefox, Safari or Opera—you decide that you can no longer put off the inevitable moment when you must check the site in IE6. Fingers are crossed, prayers are muttered, but alas, to no avail. The nemesis browser invariably screws something up.

What do you do next? If the differences in IE6 are minor, you could just leave it be. After all, websites don't need to look exactly the same in all browsers. But if there are major layout issues and a significant portion of your audience is still using IE6, you'll probably need to roll up your sleeves and start fixing the problems.

A common approach is to quarantine IE6-specific CSS in a separate stylesheet. This stylesheet can then be referenced from the HTML document using conditional comments like this:

```
<!--[if lt IE 7]>
<link rel="stylesheet" href="ie6.css" type="text/css"
media="screen" />
<![endif]-->
```

That stylesheet will only be served up to Internet Explorer where the version number is less than 7.

You can put anything inside a conditional comment. You could put a `script` element in there. So as well as serving up browser-specific CSS, it's possible to serve up browser-specific JavaScript.

A few years back, before Microsoft released Internet Explorer 7, JavaScript genius Dean Edwards wrote a script called IE7. This amazing piece of code uses JavaScript to make Internet Explorer 5 and 6 behave like a standards-compliant browser. Dean used JavaScript to bootstrap IE's CSS support.

Because the script is specifically targeted at Internet Explorer, there's no point in serving it up to other browsers. Conditional comments to the rescue:

```
<!--[if lt IE 7]>
<script src="http://ie7-js.googlecode.com/svn/version/
2.0(beta3)/IE7.js" type="text/javascript"></script>
<![endif]-->
```

Standards-compliant browsers won't fetch the script. Users of IE6, on the hand, will pay a kind of bad browser tax by having to download the JavaScript file.

So when should you develop an IE6-specific stylesheet and when should you just use Dean's JavaScript code? This is the question that myself and my co-worker Natalie Downe set out to answer one morning at Clearleft. We realised that in order to answer that question you need to first answer two other questions, "how much time does it take to develop for IE6?" and "how much of your audience is using IE6?"

Let's say that t represents the total development time. Let t_6 represent the portion of that time you spend developing for IE6. If your total audience is a, then a_6 is the portion of your audience using IE6. With some algebraic help from our mathematically minded co-worker Cennydd Bowles, Natalie and I came up with the following equation to calculate the percentage likelihood that you should be using Dean's IE7 script:

$$p = 50\left[\log\left(\frac{at_6}{ta_6}\right)+1\right]$$

```
p = 50 [ log ( at6 / ta6 ) + 1 ]
```

Try plugging in your own numbers. If you spend a lot of time developing for IE6 and only a small portion of your audience is using that browser, you'll get a very high number out of the equation; you should probably use the IE7 script. But if you only spend a little time developing for IE6 and a significant portion of you audience are still using that browser, you'll get a very small value for `p`; you might as well write an IE6-specific stylesheet.

Of course this equation is somewhat disingenuous. While it's entirely possible to research the percentage of your audience still using IE6, it's not so easy to figure out how much of your development time will be spent developing for that one browser. You can't really know until you've already done the development, by which time the equation is irrelevant.

Instead of using the equation, you could try imposing a limit on how long you will spend developing for IE6. Get your site working in standards-compliant browsers first, then give yourself a time limit to get it working in IE6. If you can't solve all the issues in that time limit, switch over to using Dean's script. You could even make the time limit directly proportional to the percentage of your audience using IE6. If 20% of your audience is still using IE6 and you've just spent five days getting the site working in standards-compliant browsers, give yourself one day to get it working in IE6. But if 50% of your audience is still using IE6, be prepared to spend 2.5 days wrestling with your nemesis.

All of these different methods for dealing with IE6 demonstrate that there's no one single answer that works for everyone. They also highlight a problem with the current debate around dealing with IE6. There's no shortage of blog posts, articles and even entire websites discussing when to drop support for IE6. But very few of them take the time to define what they mean by "support." This isn't a binary issue. There is no Boolean answer. Instead, there's a sliding scale of support:

- Block IE6 users from your site.
- Develop with web standards and don't spend any development time testing in IE6.
- Use the Dean Edwards IE7 script to bootstrap CSS support in IE6.
- Write an IE6 stylesheet to address layout issues.
- Make your site look exactly the same in IE6 as in any other browser.

Each end of that scale is extreme. I don't think that anybody should be actively blocking any browser but neither do I think that users of an outdated browser should get exactly the same experience as users of a more modern browser. The real meanings of “supporting” or “not supporting” IE6 lie somewhere in-between those extremes.

Just as I think that semantics are important in markup, they are equally important in our discussion of web development. So let’s try to come up with some better terms than using the catch-all verb “support.” If you say in your client contract that you “support” IE6, define exactly what that means. If you find yourself in a discussion about “dropping support” for IE6, take the time to explain what you think that entails.

The web developers at Yahoo! are on the right track with their concept of graded browser support. I’m interested in hearing more ideas of how to frame this discussion. If we can all agree to use clear and precise language, we stand a better chance of defeating our nemesis.

ABOUT THE AUTHOR

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

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

9. Charm Clients, Win Pitches

Marcus Lillington 24ways.org/200809

Over the years I have picked up a number of sales techniques that have lead to us doing pretty well in the pitches we go for. Of course, up until now, these top secret practices have remained firmly locked in the company vault but now I am going to share them with you. They are cunningly hidden within the following paragraphs so I'm afraid you're going to have to read the whole thing.

Ok, so where to start? I guess a good place would be getting invited to pitch for work in the first place.

SHAMELESS SELF PROMOTION

What not to do

You're as keen as mustard to 'sell' what you do, but you have no idea as to the right approach. From personal experience (sometimes bitter!), the following methods are as useful as the proverbial chocolate teapot:

- Cold calling
- Advertising
- Bidding websites
- Sales people
- Networking events

Ok, I'm exaggerating; sometimes these things work. For example, cold calling can work if you have a story – a reason to call and introduce yourself other than "we do web design and you have a website". "We do web design and we've just moved in next door to you" would be fine.

Advertising can work if your offering is highly specialist. However, paying oodles of dollars a day to Google Ads to appear under the search term 'web design' is probably not the best use of your budget.

Specialising is, in fact, probably a good way to go. Though it can feel counter intuitive in that you are not spreading yourself as widely as you might, you will eventually become an expert and therefore gain a reputation in your field. Specialism doesn’t necessarily have to be in a particular skillset or technology, it could just as easily be in a particular supply chain or across a market.

Target audience

‘Who to target?’ is the next question. If you’re starting out then do tap-up your family and friends. Anything that comes your way from them will almost certainly come with a strong recommendation. Also, there’s nothing wrong with calling clients you had dealings with in previous employment (though beware of any contractual terms that may prevent this). You are informing your previous clients that your situation has changed; leave it up to them to make any move towards working with you. After all, you’re simply asking to be included on the list of agencies invited to tender for any new work.

Look to target clients similar to those you have worked with previously. Again, you have a story – hopefully a good one!

So how do you reach these people?

- Mailing lists
- Forums
- Writing articles
- Conferences / Meetups
- Speaking opportunities
- Sharing Expertise

In essence: blog, chat, talk, enthuse, show off (a little)… share.

There are many ways you can do this. There's the traditional portfolio, almost obligatory blog (regularly updated of course), podcast, 'giveaways' like Wordpress templates, CSS galleries and testimonials. Testimonials are your greatest friend. Always ask clients for quotes (write them and ask for their permission to use) and even better, film them talking about how great you are.

Finally, social networking sites can offer a way to reach your target audiences. You do have to be careful here though. You are looking to build a reputation by contributing value. Do not self promote or spam!

WRITING PROPOSALS

Is it worth it?

Ok, so you have been invited to respond to a tender or brief in the form of a proposal. Good proposals take time to put together so you need to be sure that you are not wasting your time. There are two fundamental questions that you need to ask prior to getting started on your proposal:

1. Can I deliver within the client's timescales?
2. Does the client's budget match my price?

The timescales that clients set are often plucked from the air and a little explanation about how long projects usually take can be enough to change expectations with regard to delivery. However, if a deadline is set in stone ask yourself if you can realistically meet it. Agreeing to a deadline that you know you cannot meet just to win a project is a recipe for an unhappy client, no chance of repeat business and no chance of any recommendations to other potential clients.

Price is another thing altogether. So why do we need to know?

The first reason, and most honest reason, is that we don't want to do a lot of unpaid pitch work when there is no chance that our price will be accepted. Who would? But this goes both ways – the client's time is also being wasted. It may only be the time to read the proposal and reject it, but what if all the bids are too expensive? Then the client needs to go through the whole process again.

The second reason why we need to know budgets relates to what we would like to include in a proposal over what we need to include. For example, take usability testing. We always highly recommend that a client pays for at least one round of usability testing because it will definitely improve their new site – no question. But, not doing it doesn't mean they'll end up with an unusable turkey. It's just more likely that any usability issues will crop up after launch.

I have found that the best way to discover a budget is to simply provide a ballpark total, usually accompanied by a list of 'likely tasks for this type of project', in an initial email or telephone response. Expect a lot of people to dismiss you out of hand. This is good. Don't be tempted to 'just go for it' anyway because you like the client or work is short – you will regret it.

Others will say that the ballpark is ok. This is not as good as getting into a proper discussion about what priorities they might have but it does mean that you are not wasting your time and you do have a chance of winning the work. The only real risk with this approach is that you misinterpret the requirements and produce an inaccurate ballpark.

Finally, there is a less confrontational approach that I sometimes use that involves modular pricing. We break down our pricing into quite detailed tasks for all proposals but when I really do not have a clue about a client's budget, I will often separate pricing into 'core' items and 'optional' items. This has proved to be a very effective method of presenting price.

What to include

So, what should go into a proposal? It does depend on the size of the piece of work. If it’s a quick update for an existing client then they don’t want to read through all your blurb about why they should choose to work with you – a simple email will suffice.

But, for a potential new client I would look to include the following:

- Your suitability
- Summary of tasks
- Timescales
- Project management methodology
- Pricing
- Testing methodology
- Hosting options
- Technologies
- Imagery
- References
- Financial information
- Biographies

However, probably the most important aspect of any proposal is that you respond fully to the brief. In other words, don’t ignore the bits that either don’t make sense to you or you think irrelevant. If something is questionable, cover it and explain why you don’t think it is something that warrants inclusion in the project.

Should you provide speculative designs? If the brief doesn’t ask for any, then certainly not. If it does, then speak to the client about why you don’t like to do speculative designs. Explain that any designs included as part of a proposal are created to impress the client and not the website’s target audience. Producing good web design is a partnership between client and agency. This can often impress and promote you as a professional. However, if they insist then you need to make a decision because not delivering any mock-ups will mean that all your other work will be a waste of time.

Walking away

As I have already mentioned, all of this takes a lot of work. So, when should you be prepared to walk away from a potential job? I have already covered unrealistic deadlines and insufficient budget but there are a couple of other reasons. Firstly, would this new client damage your reputation, particularly within current sectors you are working in? Secondly, can you work with this client? A difficult client will almost certainly lead to a loss-making project.

Requirements

If the original brief didn't spell out what is expected of you at a presentation then make sure you ask beforehand. The critical element is how much time you have. It seems that panels are providing less and less time these days.

The usual formula is that you get an hour; half of which should be a presentation of your ideas followed by 30 minutes of questions. This isn't that much time, particularly for a big project that covers all aspect of web design and production. Don't be afraid to ask for more time, though it is very rare that you will be granted any.

Ask if there any areas that a) they particularly want you to cover and b) if there are any areas of your proposal that were weak.

Ask who will be attending. The main reason for this is to see if the decision maker(s) will be present but it's also good to know if you're presenting to 3 or 30 people.

Who should be there

Generally speaking, I think two is the ideal number. Though I have done many presentations on my own, I always feel having two people to bounce ideas around with and have a bit of banter with, works well. You are not only trying to sell your ideas and expertise but also yourselves. One of the main things in the panels minds will be – “can I work with these people?”

Having more than two people at a presentation often looks like you’re wheeling people out just to demonstrate that they exist.

What makes a client want to hire you?

In a nutshell: Confidence, Personality, Enthusiasm.

You can impart confidence by being well prepared and professional, providing examples and demonstrations and talking about your processes. You may find project management boring but pretty much every potential client will want to feel reassured that you manage your projects effectively.

As well as demonstrating that you know what you’re talking about, it is important to encourage, and be part of, discussion about the project. Be prepared to suggest and challenge and be willing to say “I don’t know”.

Also, no-one likes a show-off so don’t over promote yourself; encourage them to contact your existing clients.

What makes a client like you?

Engaging with a potential client is tricky and it's probably the area where you need to be most on your toes and try to gauge the reaction of the client. We recommend the following:

- Encourage questions throughout
- Ask if you make sense – which encourages questions if you're not getting any
- Humour – though don't keep trying to be funny if you're not getting any laughs!
- Be willing to go off track
- Read your audience
- Empathise with the process – chances are, most of the people in front of you would rather be doing something else
- Think about what you wear – this sounds daft but do you want to be seen as either the 'stiff in the suit' or the 'scruffy art student'? Chances are neither character would get hired.

Differentiation

Sometimes, especially if you think you are an outsider, it's worth taking a few risks. I remember my colleague Paul starting off a presentation once with the line (backed up on screen) – "Headscape is not a usability consultancy".

This was in response to the clients request to engage a usability consultancy. The thrust of Paul's argument was that we are a lot more than that.

This really worked. We were the outside choice but they ended up hiring us. Basically, this differentiated us from the crowd. It showed that we are prepared to take risks and think, dare I say it, outside of the box.

Dealing with difficult characters

How you react to tricky questioning is likely to be what determines whether you have a good or bad presentation. Here are a few of those characters that so often turn up in panels:

The techie – this is likely to be the situation where you need to say "I don't know". Don't bluff as you are likely to dig yourself a great big embarrassment-filled hole. Promise to follow up with more information and make sure that you do so as quickly as possible after the pitch.

The 'hard man' MD – this the guy who thinks it is his duty to throw 'curve ball' questions to see how you react. Focus on your track record (big name clients will impress this guy) and emphasise your processes.

The ‘no clue’ client – you need to take control and be the expert though you do need to explain the reasoning behind any suggestions you make. This person will be judging you on how much you are prepared to help them deliver the project.

The price negotiator – be prepared to discuss price but do not reduce your rate or the effort associated with your proposal. Fall back on modular pricing and try to reduce scope to come within budget. You may wish to offer a one-off discount to win a new piece of work but don’t get into detail at the pitch.

Don’t panic…

If you go into a presentation thinking ‘we must win this’ then, chances are, you won’t. Relax and be yourself. If you’re not hitting it off with the panel then so be it. You have to remember that quite often you will be making up the numbers in a tendering process. This is massively frustrating but, unfortunately, part of it. If it’s not going well, concentrate on what you are offering and try to demonstrate your professionalism rather than your personality. Finally, be on your toes, watch people’s reactions and pay attention to what they say and try to react accordingly.

So where are the secret techniques I hear you ask? Well, using the words ‘secret’ and ‘technique’ was probably a bit naughty. Most of this stuff is about being keen, using your brain and believing in yourself and what you are selling rather than following a strict set of rules.

ABOUT THE AUTHOR

Marcus Lillington is Business Development Director at Headscape and Co-host of the Boagworld Podcast.

Marcus has over 10 years experience working in both new and traditional media, including experience in commercial, consulting and project management roles. With Paul Boag and Chris Scott, he is one of three founders of Headscape.

In his early career, Marcus was a professional musician, gaining success around the world. He still claims it’s the only thing he’s any good at.

10. A Christmas hCard From Me To You

Elliot Jay Stocks 24ways.org/200810

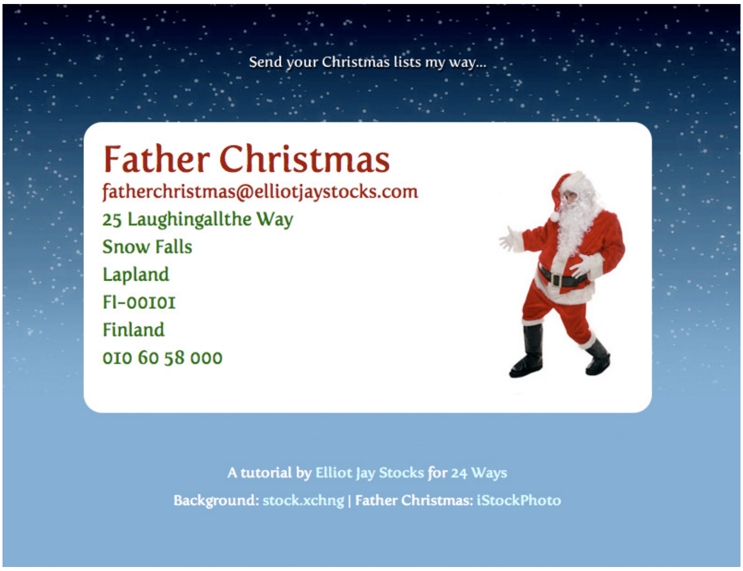
So apparently Christmas is coming. And what is Christmas all about? Well, cleaning out your address book, of course! What better time to go through your contacts, making sure everyone's details are up date and that you've deleted all those nasty clients who never paid on time?

It's also a good time to make sure your current clients and colleagues have your most up-to-date details, so instead of filling up their inboxes with e-cards, why not send them something useful? Something like a… vCard! (See what I did there?)

Just in case you've been working in a magical toy factory in the upper reaches of Scandinavia for the last few years, I'm going to tell you that now would also be the perfect time to get into microformats. Using the hCard format, we'll build a very simple web page and markup our contact details in such a way that they'll be understood by microformats plugins, like Operator or Tails for Firefox, or the cross-browser Microformats Bookmarklet.

Oh, and because Christmas is all about dressing up and being silly, we'll make the whole thing look nice and have a bit of fun with some CSS3 progressive enhancement.

If you can't wait to see what we end up with, you can preview it here.


Send your Christmas lists my way...

Father Christmas
fatherchristmas@elliotjaystocks.com
25 Laughingallthe Way
Snow Falls
Lapland
FI-00101
Finland
010 60 58 000

A tutorial by Elliot Jay Stocks for 24 Ways
Background: stock.xchng | Father Christmas: iStockPhoto

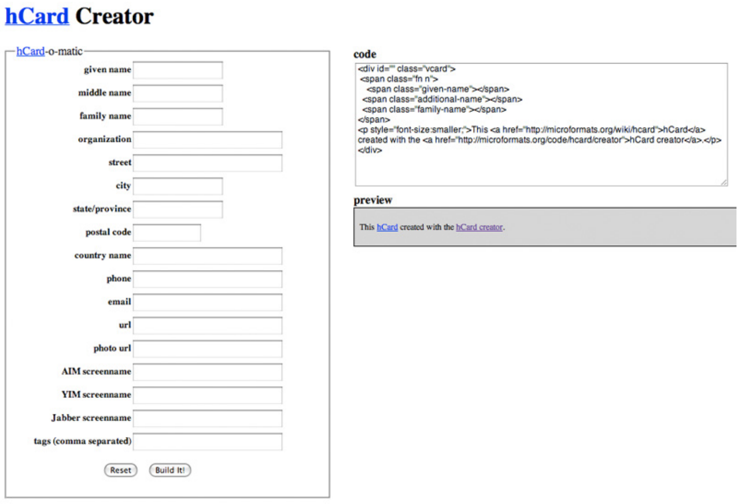
STEP 1: CONTACT DETAILS

First, let’s decide what details we want to put on the page. I’d put my full name, my email address, my phone number, and my postal address, but I’d rather not get surprise visits from strangers when I’m fannying about with my baubles, so I’m going to use Father Christmas instead (that’s Santa to you Yanks).

```
Father Christmas
fatherchristmas@elliotjaystocks.com
25 Laughingallthe Way
Snow Falls
Lapland
Finland
010 60 58 000
```

STEP 2: HCARD CREATOR

Now I’m not sure about you, but I rather like getting the magical robot pixies to do the work for me, so head on over to the hCard Creator and put those pixies to work! Pop in your details and they’ll give you some nice microformatted HTML in turn.



```
<div id="" class="vcard">
 <span class="fn n">
   <span class="given-name"></span>
  <span class="additional-name"></span>
  <span class="family-name"></span>
</span>
<p style="font-size:smaller;">This <a href="http://microformats.org/wiki/hcard">hCard</a>
created with the <a href="http://microformats.org/code/hcard/creator">hCard creator</a>.</p>
</div>
```

This hCard created with the hCard creator.

```
<div id="hcard-Father-Christmas" class="vcard">
  <a class="url fn" href="http://elliotjaystocks.com/
fatherchristmas">Father Christmas</a>
  <a class="email"
href="mailto:fatherchristmas@elliotjaystocks.com">
fatherchristmas@elliotjaystocks.com</a>
  <div class="adr">
  <div class="street-address">25 Laughingallthe Way</div>
  <span class="locality">Snow Falls</span>
  ,
  <span class="region">Lapland</span>
  ,
  <span class="postal-code">FI-00101</span>
  <span class="country-name">Finland</span>
</div>
<div class="tel">010 60 58 000</div>
  <p style="font-size:smaller;">This <a
```

```
href="http://microformats.org/wiki/hcard">hCard</a>
created with the <a href="http://microformats.org/code/
hcard/creator">hCard creator</a>.</p>
</div>
```

STEP 3: EDITING THE CODE

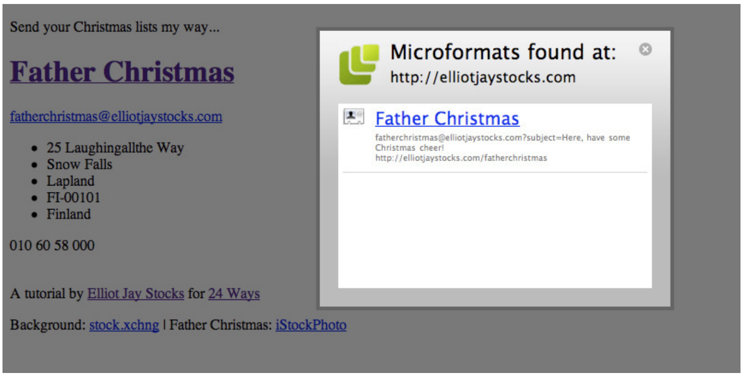
One of the great things about microformats is that you can use pretty much whichever HTML tags you want, so just because the hCard Creator Fairies say something should be wrapped in a `<span>` doesn't mean you can't change it to a `<blink>`. Actually, no, don't do that. That's not even excusable at Christmas.

I personally have a penchant for marking up each line of an address inside a `<li>` tag, where the parent url retains the class of `adr`. As long as you keep the class names the same, you'll be fine.

```
<div id="hcard-Father-Christmas" class="vcard">
  <h1><a class="url fn" href="http://elliotjaystocks.com/
fatherchristmas">Father Christmas </a></h1>
  <a class="email"
href="mailto:fatherchristmas@elliotjaystocks.com?subject=Here,
have some Christmas
cheer!">fatherchristmas@elliotjaystocks.com</a>
  <ul class="adr">
    <li class="street-address">25 Laughingallthe Way</li>
    <li class="locality">Snow Falls</li>
    <li class="region">Lapland</li>
    <li class="postal-code">FI-00101</li>
    <li class="country-name">Finland</li>
  </ul>
  <span class="tel">010 60 58 000</span>
</div>
```

STEP 4: TESTING THE MICROFORMATS

With our microformats in place, now would be a good time to test that they're working before we start making things look pretty. If you're on Firefox, you can install the Operator or Tails extensions, but if you're on another browser, just add the Microformats Bookmarklet. Regardless of your choice, the results is the same: if you've code microformatted content on a web page, one of these bad boys should pick it up for you and allow you to export the contact info. Give it a try and you should see father Christmas appearing in your address book of choice. Now you'll never forget where to send those Christmas lists!



Send your Christmas lists my way...

Father Christmas

fatherchristmas@elliotjaystocks.com

- 25 Laughingallthe Way
- Snow Falls
- Lapland
- FI-00101
- Finland

010 60 58 000

A tutorial by Elliot Jay Stocks for 24 Ways

Background: stock.xchng | Father Christmas: iStockPhoto

Microformats found at:
http://elliotjaystocks.com

Father Christmas
fatherchristmas@elliotjaystocks.com?subject=Here, have some Christmas cheer!
http://elliotjaystocks.com/fatherchristmas

STEP 5: SOME EXTRA MARKUP

One of the first things we're going to do is put a photo of Father Christmas on the hCard. We'll be using CSS to apply a background image to a div, so we'll be needing an extra div with a class name of "photo". In turn, we'll wrap the text-based elements of our hCard inside a div cunningly called "text". Unfortunately, because of the float technique we'll be using, we'll have to use one of those nasty float-clearing techniques. I shall call this "christmas-cheer", since that is what its presence will inevitably bring, of course.

Oh, and let's add a bit of text to give the page context, too:

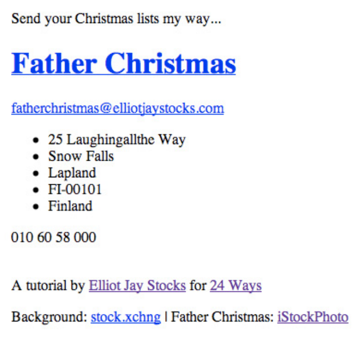
```
<p>Send your Christmas lists my way...</p>
<div id="hcard-Father-Christmas" class="vcard">
  <div class="text">
    <h1><a class="url fn"
href="http://elliotjaystocks.com/fatherchristmas">Father
Christmas </a></h1>
    <a class="email"
href="mailto:fatherchristmas@elliotjaystocks.com?subject=Here,
have some Christmas
cheer!">fatherchristmas@elliotjaystocks.com</a>
    <ul class="adr">
      <li class="street-address">25 Laughingallthe
Way</li>
      <li class="locality">Snow Falls</li>
      <li class="region">Lapland</li>
      <li class="postal-code">FI-00101</li>
      <li class="country-name">Finland</li>
    </ul>
    <span class="tel">010 60 58 000</span>
  </div>
  <div class="photo"></div>
  <br class="christmas-cheer" />
</div>
<div class="credits">
  <p>A tutorial by <a
href="http://elliotjaystocks.com">Elliot Jay Stocks</a>
for <a href="http://24ways.org/">24 Ways</a></p>
  <p>Background: <a href="http://sxc.hu/photo/
1108741">stock.xchng</a> | Father Christmas: <a
href="http://istockphoto.com/file_closeup/people/
4575943-active-santa.php?id=4575943">iStockPhoto</a></p>
</div>
```

STEP 6: SOME CHRISTMAS SPARKLE

So far, our hCard-housing web page is slightly less than inspiring, isn't it? It's time to add a bit of CSS. There's nothing particularly radical going on here; just a simple layout, some basic typographic treatment, and the placement of the Father Christmas photo. I'd usually use a more thorough CSS reset like the one found in the YUI or Eric Meyer's, but for this basic page, the simple `*` solution will do.

Check out the step 6 demo to see our basic styles in place.

From this…


Send your Christmas lists my way...

Father Christmas

fatherchristmas@elliotjaystocks.com

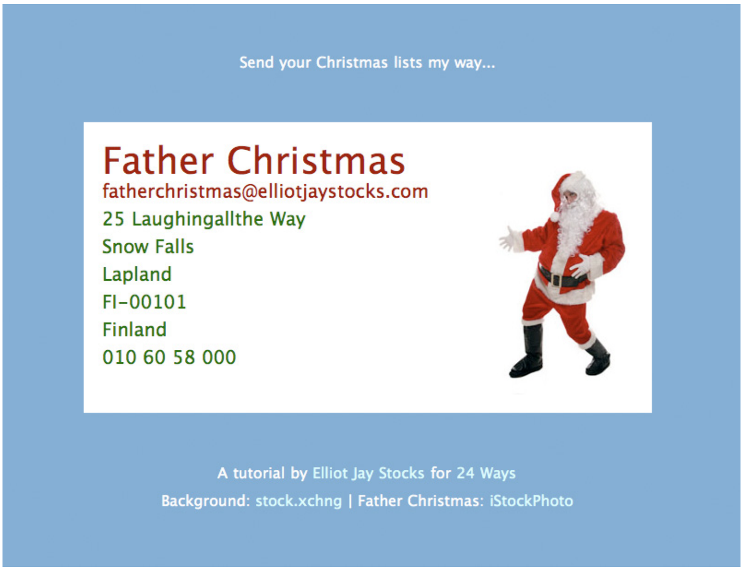
- 25 Laughingallthe Way
- Snow Falls
- Lapland
- FI-00101
- Finland

010 60 58 000

A tutorial by Elliot Jay Stocks for 24 Ways

Background: stock.xchng | Father Christmas: iStockPhoto

... to this:


Send your Christmas lists my way...

Father Christmas
fatherchristmas@elliotjaystocks.com
25 Laughingallthe Way
Snow Falls
Lapland
FI-00101
Finland
010 60 58 000

A tutorial by Elliot Jay Stocks for 24 Ways
Background: stock.xchng | Father Christmas: iStockPhoto

STEP 7: FUN WITH IMAGERY

Now it's time to introduce a repeating background image to the `<body>` element. This will seamlessly repeat for as wide as the browser window becomes.

But that's fairly straightforward. How about having some fun with the Father Christmas image? If you look at the image file itself, you'll see that it's twice as wide as the area we can see and contains a 'hidden' photo of our rather camp St. Nick.

As a light-hearted visual… er… ‘treat’ for users who move their mouse over the image, we move the position of the background image on the “photo” div. Check out the step 7 demo to see it working.

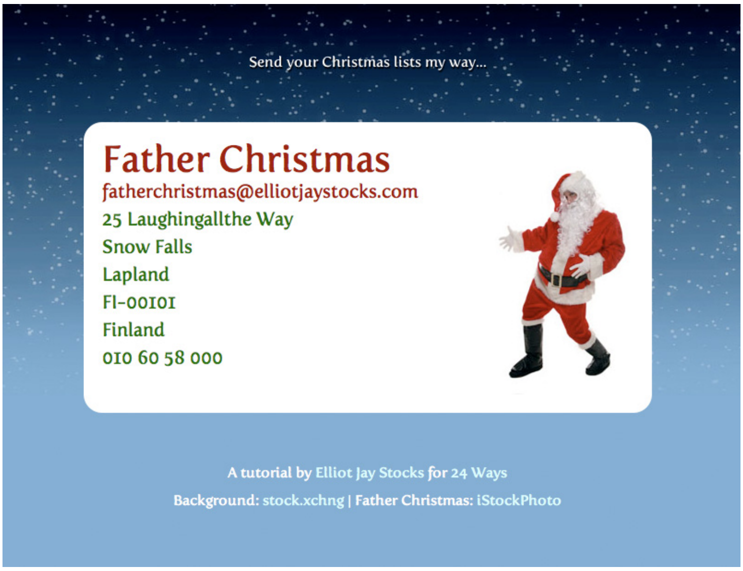
STEP 8: PROGRESSIVE ENHANCEMENT

Finally, this fun little project is a great opportunity for us to mess around with some advanced CSS features (some from the CSS3 spec) that we rarely get to use on client projects. (Don’t forget: no Christmas pressies for clients who want you to support IE6!)

Here are the rules we’re using to give some browsers a superior viewing experience:

- `@font-face` allows us to use Jos Buivenga’s free font ‘Fertigo Pro’ on all text;
- `text-shadow` adds a little emphasis on the opening paragraph;
- `body > p:first-child` causes only the first paragraph to receive this treatment;
- `border-radius` created rounded corners on our main `div` and the links within it;
- and `webkit-transition` allows us to gently fade in between the default and hover states of those links.

And with that, we're done! You can see the results here. It's time to customise the page to your liking, upload it to your site, and send out the URL. And do it quickly, because I'm sure you've got some last-minute Christmas shopping to finish off!



Send your Christmas lists my way...

Father Christmas
fatherchristmas@elliotjaystocks.com
25 Laughingallthe Way
Snow Falls
Lapland
FI-00101
Finland
010 60 58 000

A tutorial by Elliot Jay Stocks for 24 Ways
Background: stock.xchng | Father Christmas: iStockPhoto

ABOUT THE AUTHOR

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

Photo: Samantha Cliffe

11. Easier Page States for Wireframes

Richard Rutter 24ways.org/200811

When designing wireframes for web sites and web apps, it is often overlooked that the same ‘page’ can look wildly different depending on its context. A logged-in page will look different from a logged-out page; an administrator’s view may have different buttons than a regular user’s view; a power user’s profile will be more extensive than a new user’s.

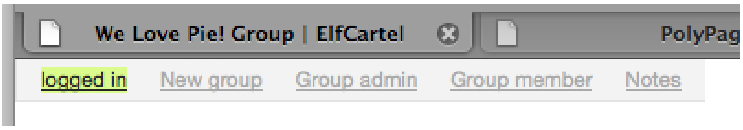
These different page states need designing at some point, especially if the wireframes are to form a useful communication medium between designer and developer. Documenting the different permutations can be a time consuming exercise involving either multiple pages in one’s preferred box-and-arrow software, or a fully fledged drawing containing all the possible combinations annotated accordingly.

ENTER INTERACTIVE WIREFRAMES AND POLYPAGE

Interactive wireframes built in HTML are a great design and communication tool. They provide a clickable prototype, running in the browser as would the final site. As such they give a great feel for how the site will be to use. Once you add in the possibilities of JavaScript and a library such as jQuery, they become even more flexible and powerful.

Polypage is a jQuery plugin which makes it really easy to design multiple page states in HTML wireframes. There's no JavaScript knowledge required (other than cutting and pasting in a few lines). The page views are created by simply writing all the alternatives into your HTML page and adding special class names to apply state and conditional view logic to the various options.

When the page is loaded Polypage automatically detects the page states defined by the class names and creates a control bar enabling the user to toggle page states with the click of a mouse or the clack of a keyboard.



Using cookies by way of the jQuery cookie plugin, Polypage retains the view state throughout your prototype. This means you could navigate through your wireframes as if you were logged out; as if you were logged in as an administrator; with notes on or off; or with any other view or state you might require. The possibilities are entirely up to you.

HOW DOES IT WORK?

Firstly you need to link to jQuery, the jQuery cookie plugin and to Polypage. Something like this:

```
<script src="javascripts/jquery-1.2.6.min.js" type="text/
javascript"></script>
<script src="javascripts/cookie.jquery.js" type="text/
javascript"></script>
<script src="javascripts/polypage.jquery.js" type="text/
javascript"></script>
```

Then you need to initialise Polypage on page load using something along these lines:

```
<script type="text/javascript">
  $(document).ready(function() {
    $.polypage.init();
  });
</script>
```

Next you need to define the areas of your wireframe which are particular to a given state or view. Do this by applying classes beginning with pp_. Polypage will ignore all other classes in the document.

The `pp_` prefix should be followed by a state name. This can be any text string you like, bearing in mind it will appear in the control bar. Typical page states might include 'logged_in', 'administrator' or 'group_owner'. A complete class name would therefore look something like `pp_logged_in`.

EXAMPLES

If a user is logged in, you might want to specify an option for him or her to sign out. Using Polypage, this could be put in the wireframe as follows:

```
<a href="logout" class="pp_logged_in"> Sign out </a>
```

Polypage will identify the `pp_logged_in` class on the link and hide it (as the 'Sign out' link should only be shown when the page is in the 'logged in' view). Polypage will then automatically write a 'logged in' toggle to the control bar, enabling you to show or hide the 'Sign out' link by toggling the 'logged in' view. The same will apply to all content marked with a `pp_logged_in` class.

States can also be negated by adding a `not` keyword to the class name. For example you might want to provide a log in link for users who are **not** signed in. Using Polypage, you would insert the `not` keyword after the `pp` prefix as follows:

```
<a href="login" class="pp_not_logged_in"> Login </a>
```

Again Polypage identifies the `pp` prefix but this time sees that the 'Login' link should not be shown when the 'logged in' state is selected.

States can also be joined together to add some basic logic to pages. The syntax follows natural language and uses the `or` and `and` keywords in addition to the aforementioned `not`. Some examples would be pp_logged_in_and_admin, pp_admin_or_group_owner and pp_logged_in_and_not_admin.

Finally, you can set default states for a page by passing an array to the polypage.init() function like this:

```
$.polypage.init(['logged_in', 'admin']);
```

You can see a fully fledged example in this fictional social network group page. The example page defaults to a logged in state. You can see the logged out state by toggling 'logged in' off in the Polypage control bar. There are also views specified for a *group member*, a *group admin*, a *new group* and *notes*.

WHERE CAN I GET HOLD OF IT?

You can download the current version from GitHub.

Polypage was originally developed by Clearleft and New Bamboo, with particular contributions from Andy Kent and Natalie Downe. It has been used in numerous real projects, but it is still an early release so there is bound to be room for improvement. We're pleased to say that Polypage is now an open source project so any feedback, particularly by way of actual improvements, is extremely welcome.

ABOUT THE AUTHOR

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

12. Checking Out: Progress Meters

Kimberly Blessing 24ways.org/200812

It’s the holiday season, so you know what that means: online shopping! When I started developing Web sites back in the 90s, many of my first clients were small local shops wanting to sell their goods online, so I developed many a checkout system. And because of slow dial-up speeds back then, informing the user about where they were in the checkout process was pretty important.

Even though we’re (mostly) beyond the dial-up days, informing users about where they are in a flow is still important. In usability tests at the companies I’ve worked at, I’ve seen time and time again how not adequately informing the user about their state can cause real frustration. This is especially true for two sets of users: mobile users and users of assistive devices, in particular, screen readers.

The progress meter is a very common design solution used to indicate to the user's state within a flow. On the design side, much effort may go in to crafting a solution that is as visually informative as possible. On the development side, however, solutions range widely. I've checked out the checkouts at a number of sites and here's what I've found when it comes to progress meters: they're sometimes inaccessible and often confusing or unhelpful — all because of the way in which they're coded. For those who use assistive devices or text-only browsers, there must be a better way to code the progress meter — and there is.

(Note: All code samples are from live sites but have been tweaked to hide the culprits' identities.)

HOW NOT TO MAKE PROGRESS

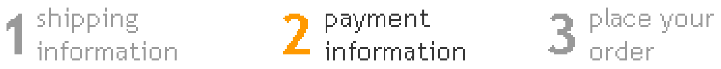
A number of sites assemble their progress meters using non- or semi-semantic markup and images with no alternate text. On text-only browsers (like my mobile phone) and to screen readers, this looks and reads like chunks of content with no context given.

```
<div id="progress">
  <img src="icon_progress_1a.gif" alt="">
  <em>Shipping information</em>
  <img src="icon_progress_arrow.gif" alt="">
  <img src="icon_progress_2a.gif" alt="">
  <em>Payment information</em>
  <img src="icon_progress_arrow.gif" alt=""
class="progarrow">
  <img src="icon_progress_3b.gif" alt="">
  <strong>Place your order</strong>
</div>
```

In the above example, the third state, "Place your order", is the current state. But a screen reader may not know that, and my cell phone only displays `"Shipping informationPayment informationPlace your order"`. Not good.

IS THIS PROGRESS?

Other sites present the entire progress meter as a graphic, like the following:



Now, I have no problem with using a graphic to render a very stylish progress meter (my sample above is probably not the most stylish example, of course, but you understand my point). What becomes important in this case is the use of appropriate alternate text to describe the image. Disappointingly, sites today have a wide range of solutions, including using no alternate text. Check out these code samples which call progress meter images.

```
<img src="checkout_step2.gif" alt="">
```

I think we can all agree that the above is bad, unless you really don't care whether or not users know where they are in a flow.

```
<img src="checkout_step2.gif" alt="Shipping information
- Payment information - Place your order">
```

The alt text in the example above just copies all of the text found in the graphic, but it doesn't represent the status at all. So for every page in the checkout, the user sees or hears the same text. Sure, by the second or third page in the flow, the user has figured out what's going on, but she or he had to think about it. I don't think that's good.

```
<img src="checkout_step2.gif" alt="Checkout: Payment
information">
```

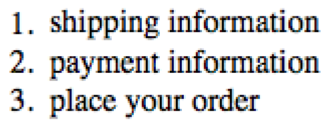
The above probably has the best alternate text out of these examples, because the user at least understands that they're in the Checkout process, on the Place your order page. But going through the flow with alt text like this, the user doesn't know how many steps are in the flow.

SEMANTIC PROGRESS

Of course, there are some sites that use an ordered list when marking up the progress meter. Hooray! Unfortunately, no text-only browser or screen reader would be able to describe the user's current state given this markup.

```
<ol class="progressmeter">
  <li class="one current">shipping information</li>
  <li class="two">payment information</li>
  <li class="three">place your order</li>
</ol>
```

Without CSS enabled, the above is rendered as follows:


1. shipping information
2. payment information
3. place your order

PROGRESS AT LAST

We all know that semantic markup makes for the best foundation, so we'll start with the markup found above. In order to make the state information accessible, let's add some additional text in paragraph and span elements.

```
<div class="progressmeter">
  <p>There are three steps in this checkout process.</p>
  <ol>
    <li class="one"><span>Enter your</span> shipping
information</li>
    <li class="two"><span>Enter your</span> payment
information</li>
    <li class="three"><span>Review details and</span>
place your order</li>
  </ol>
</div>
```

Add on some simple CSS to hide the paragraph and spans, and arrange the list items on a single line with a background image to represent the large number, and this is what you'll get:

There are three steps in this checkout process.

1. Enter your shipping information
2. Enter your payment information
3. Review details and place your order

To display and describe a state as active, add the class "current" to one of the list items. Then change the hidden content such that it better describes the state to the user.

```
<div class="progressmeter">
  <p>There are three steps in this checkout process.</p>
  <ol>
    <li class="one current"><span>You are currently
entering your</span> shipping information</li>
    <li class="two"><span>In the next step, you will
enter your</span> payment information</li>
    <li class="three"><span>In the last step, you will
review the details and</span> place your order</li>
  </ol>
</div>
```

The end result is an attractive progress meter that gives much greater semantic and contextual information.

There are three steps in this checkout process.

1. You are currently entering your shipping information

2. In the next step, you will enter your payment information
3. In the last step, you will review the details and place your order

For example, the above example renders in a text-only browser as follows:

> There are three steps in this checkout process.
>
> 1. You are currently entering your shipping information
> 2. In the next step, you will enter your payment information
> 3. In the last step, you will review the details and place your order

And the screen reader I use for testing announces the following:

> There are three steps in this checkout process. List of three items. You are currently entering your shipping information. In the next step, you will enter your payment information. In the last step, you will review the details and place your order. List end.

Here's a sample code page that summarises this approach.

Happy frustration-free online shopping with this improved progress meter!

ABOUT THE AUTHOR

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

13. The First Tool You Reach For

Kevin Yank 24ways.org/200813

Microsoft recently announced that Internet Explorer 8 will be released in the first half of 2009. Compared to the standards support of other major browsers, IE8 will not be especially great, but it *will* finally catch up with the state of the art in one specific area: support for CSS tables. This milestone has the potential to trigger an important change in the way you approach web design.

To show you just how big a difference CSS tables can make, think about how you might code a fluid, three-column layout from scratch. Just to make your life more difficult, give it one fixed-width column, with a background colour that differs from the rest of the page. Ready? Go!

Okay, since you're the sort of discerning web designer who reads 24ways, I'm going to assume you at least *considered* doing this without using HTML tables for the layout. If you're especially hardcore, I imagine you began thinking of CSS floats, negative margins, and faux columns. If you did, colour me impressed!

Now admit it: you probably also gave an inward sigh about the time it would take to figure out the math on the negative margin overlaps, check for dropped floats in Internet Explorer and generally wrestle each of the major browsers into giving you what you want. If after all that you simply gave up and used HTML tables, I can't say I blame you.

There are plenty of professional web designers out there who still choose to use HTML tables as their main layout tool. Sure, they may know that users with screen readers get confused by inappropriate use of tables, but they have a job to do, and they want tools that will make that job easy, not difficult.

Now let me show you how to do it with CSS tables. First, we have a `div` element for each of our columns, and we wrap them all in another two `div`s:

```
<div class="container">
  <div>
    <div id="menu">
    ⋮
    </div>
    <div id="content">
    ⋮
    </div>
    <div id="sidebar">
    ⋮
    </div>
  </div>
</div>
```

Don't sweat the "`div` clutter" in this code. Unlike tables, `div`s have no semantic meaning, and can therefore be used liberally (within reason) to provide hooks for the styles you want to apply to your page.

Using CSS, we can set the outer `div` to display as a table with collapsed borders (i.e. adjacent cells share a border) and a fixed layout (i.e. cell widths unaffected by their contents):

```
.container {
  display: table;
  border-collapse: collapse;
  table-layout: fixed;
}
```

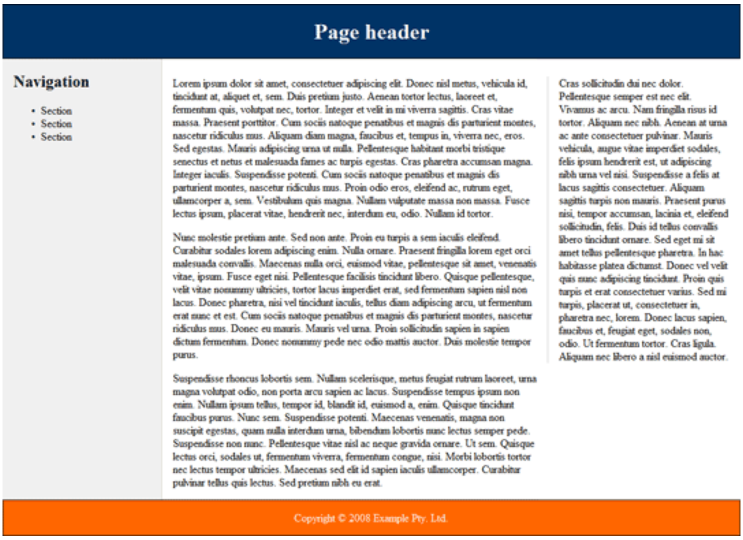
With another two rules, we set the middle `div` to display as a table row, and each of the inner `div`s to display as table cells:

```
.container > div {
  display: table-row;
}
.container > div > div {
  display: table-cell;
}
```

Finally, we can set the widths of the cells (and of the table itself) directly:

```
.container {
  width: 100%;
}
#menu {
  width: 200px;
}
#content {
  width: auto;
}
#sidebar {
  width: 25%;
}
```

And, just like that, we have a rock solid three-column layout, ready to be styled to your own taste, like in this example:


Page header

Navigation

- Section
- Section
- Section

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec nisl metus, vehicula id, tincidunt at, aliquet et, sem. Duis pretium justo. Aenean tortor lectus, laoreet et, fermentum quis, volutpat nec, tortor. Integer et velit in mi viverra sagittis. Cras vitae massa. Praesent porttitor. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam diam magna, faucibus et, tempus in, viverra nec, eros. Sed egestas. Mauris adipiscing urna ut nulla. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Cras pharetra accumsan magna. Integer iaculis. Suspendisse potenti. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Proin odio eros, eleifend ac, rutrum eget, ullamcorper a, sem. Vestibulum quis magna. Nullam vulputate massa non massa. Fusce lectus ipsum, placerat vitae, hendrerit nec, interdum eu, odio. Nullam id tortor.

Nunc molestie pretium ante. Sed non ante. Proin eu turpis a sem iaculis eleifend. Curabitur sodales lorem adipiscing enim. Nulla ornare. Praesent fringilla lorem eget orci malesuada convallis. Maecenas nulla orci, euismod vitae, pellentesque sit amet, venenatis vitae, ipsum. Fusce eget nisi. Pellentesque facilisis tincidunt libero. Quisque pellentesque, velit vitae nonummy ultricies, tortor lacus imperdiet erat, sed fermentum sapien nisl non lacus. Donec pharetra, nisi vel tincidunt iaculis, tellus diam adipiscing arcu, ut fermentum erat nunc et est. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec eu mauris. Mauris vel urna. Proin sollicitudin sapien in sapien dictum fermentum. Donec nonummy pede nec odio mattis auctor. Duis molestie tempor purus.

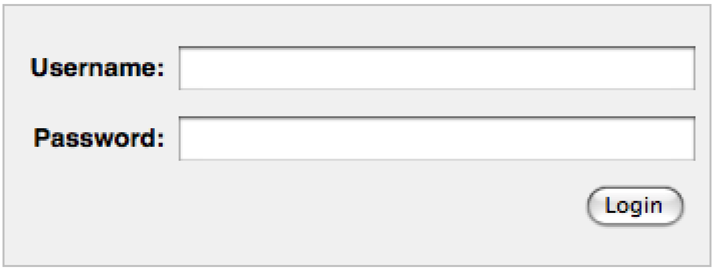
Suspendisse rhoncus lobortis sem. Nullam scelerisque, metus feugiat rutrum laoreet, urna magna volutpat odio, non porta arcu sapien ac lacus. Suspendisse tempus ipsum non enim. Nullam ipsum tellus, tempor id, blandit id, euismod a, enim. Quisque tincidunt faucibus purus. Nunc sem. Suspendisse potenti. Maecenas venenatis, magna non suscipit egestas, quam nulla interdum urna, bibendum lobortis nunc lectus semper pede. Suspendisse non nunc. Pellentesque vitae nisl ac neque gravida ornare. Ut sem. Quisque lectus orci, sodales ut, fermentum viverra, fermentum congue, nisi. Morbi lobortis tortor nec lectus tempor ultricies. Maecenas sed elit id sapien iaculis ullamcorper. Curabitur pulvinar tellus quis lectus. Sed pretium nibh eu erat.

Cras sollicitudin dui nec dolor. Pellentesque semper est nec elit. Vivamus ac arcu. Nam fringilla risus id tortor. Aliquam nec nibh. Aenean at urna ac ante consectetuer pulvinar. Mauris vehicula, augue vitae imperdiet sodales, felis ipsum hendrerit est, ut adipiscing nibh urna vel nisi. Suspendisse a felis at lacus sagittis consectetuer. Aliquam sagittis turpis non mauris. Praesent purus nisi, tempor accumsan, lacinia et, eleifend sollicitudin, felis. Duis id tellus convallis libero tincidunt ornare. Sed eget mi sit amet tellus pellentesque pharetra. In hac habitasse platea dictumst. Donec vel velit quis nunc adipiscing tincidunt. Proin quis turpis et erat consectetuer varius. Sed mi turpis, placerat ut, consectetuer in, pharetra nec, lorem. Donec lacus sapien, faucibus et, feugiat eget, sodales non, odio. Ut fermentum tortor. Cras ligula. Aliquam nec libero a nisl euismod auctor.

Copyright © 2008 Example Pty. Ltd.

This example will render perfectly in reasonably up-to-date versions of Firefox, Safari and Opera, as well as the current beta release of Internet Explorer 8.

CSS tables aren't only useful for multi-column page layout; they can come in handy in most any situation that calls for elements to be displayed side-by-side on the page. Consider this simple login form layout:



The incantation required to achieve this layout using CSS floats may be old hat to you by now, but try to teach it to a beginner, and watch his eyes widen in horror at the hoops you have to jump through (not to mention the assumptions you have to build into your design about the length of the form labels).

Here’s how to do it with CSS tables:

```
<form action="/login" method="post">
  <div>
    <div>
      <label for="username">Username:</label>
      <span class="input"><input type="text"
name="username" id="username"/></span>
    </div>
    <div>
      <label for="userpass">Password:</label>
      <span class="input"><input type="password"
name="userpass" id="userpass"/></span>
    </div>
    <div class="submit">
      <label for="login"></label>
      <span class="input"><input type="submit"
name="login" id="login" value="Login"/></span>
    </div>
  </div>
</form>
```

This time, we’re using a mixture of `div`s and `span`s as semantically transparent styling hooks. Let’s look at the CSS code.

First, we set up the outer `div` to display as a table, the inner `div`s to display as table rows, and the `label`s and `span`s as table cells (with right-aligned text):

```
form > div {
  display: table;
}
form > div > div {
  display: table-row;
}
form label,
form span {
  display: table-cell;
  text-align: right;
}
```

We want the first column of the table to be wide enough to accommodate our labels, but no wider. With CSS float techniques, we had to guess at what that width was likely to be, and adjust it whenever we changed our form labels. With CSS tables, we can simply set the `width` of the first column to something very small (`1em`), and then use the `white-space` property to force the column to the required width:

```
form label {
  white-space: nowrap;
  width: 1em;
}
```

To polish off the layout, we'll make our text and password fields occupy the full width of the table cells that contain them:

```
input[type=text],
input[type=password] {
  width: 100%;
}
```

The rest is margins, padding and borders to get the desired look. Check out the finished example.

As the first tool you reach for when approaching any layout task, CSS tables make a lot more sense to your average designer than the cryptic incantations called for by CSS floats. When IE8 is released and all major browsers support CSS tables, we can begin to gradually deploy CSS table-based layouts on sites that are more and more mainstream.

In our new book, *Everything You Know About CSS Is Wrong!*, Rachel Andrew and I explore in much greater detail how CSS tables work as a page layout tool in the real world. CSS tables have their quirks just like floats do, but they don't tend to affect common layout tasks, and the workarounds tend to be less fiddly too. Check it out, and get ready for the next big step forward in web design with CSS.

ABOUT THE AUTHOR

Kevin Yank is the Technical Director of SitePoint, a respected publisher for web professionals. An accomplished speaker and writer, he has written books about PHP, JavaScript, and CSS, and a weekly newsletter with nearly 400,000 subscribers. He lives in Melbourne.

14. Rocking Restrictions

Tim Van Damme 24ways.org/200814

I love my job. I live my job. For every project I do, I try to make it look special. I'll be honest: I have a fetish for comments like "I never saw anything like that!" or, "I wish I thought of that!". I know, I have an ego-problem. (Eleven I's already)

But sometimes, you run out of inspiration. Happens to everybody, and everybody hates it. "I'm the worst designer in the world." "Everything I designed before this was just pure luck!" No it wasn't.

Countless articles about finding inspiration have already been written. Great, but they're not the magic potion you'd expect them to be when you need it. Here's a list of small tips that can have immediate effect when applying them/using them. Main theme: **Liberate yourself from the designers' block by restricting yourself.**

DO'S

Grids

If you aren't already using grids, you're doing something wrong. Not only are they a great help for aligning your design, they also restrict you to certain widths and heights. (For more information about grids, I suggest you read Mark Boulton's series on designing grid systems. Oh, he's also publishing a book I think.)

So what's the link between grids and restrictions? Instead of having the option to style a piece of layout with a width of 1 to 960 pixels, you have to choose from values like 60 pixels, 140, 220, 300, …

Start small

Having a hard time finding a style for the layout, why don't you start with one small object? No, not **that** small object, I meant a piece of a form, or a link, or try styling your headers (`h1` – `h6`).

Let's take a submit button of a form: it's small, but needs much attention. People will click it. People will hover it. Maybe sometimes it's disabled? Also: a button needs to look like a button, so typically it requires more styling then a regular link. Once you've got the button, move on, following the button's style.

Color palettes

There are lots of resources on the web for finding inspiration for color palettes. Some of the most famous are COLOURlovers, wear palettes and Adobe's Kuler. Browse through them (or create your own from a picture), pick a color palette you like and which works with the subject you're handling, and stick with it. 4-5 colors, maybe with some tonal variations, but that's it.

Fonts

There aren't many fonts available for the web (Richard Rutter has a great article on this subject), but you'd be surprised how long they go. A simple `text-transform: uppercase;` or `font-style: italic;` can change a dull looking font into something entirely fresh.

Play around with the fonts you want to use and the variations you'll be using, and make a list. Pick five combinations of fonts and their variations, and stick with them throughout the layout.

Single-task

Most of us use multiple monitors. They're great to increase productivity, but make it harder to focus on a single task. Here's what you do: try using only your smallest monitor. Maybe it's the one from your laptop, maybe it's an old 1024×768 you found in the attic. Having

Photoshop (or Fireworks or…) taking over your entire workspace blocks out all the other distractions on your screen, and works quite liberating.

Mute everything…

…but not entirely. I noticed I was way more focused when I set NetNewsWire to refresh it's feeds only once every two hours. After two hours, I need a break anyway. Turning off Twitterrific was a mistake, as it's my window to the world, and it's the place where the people I like to call colleagues live. You can't exactly ask them to bring you a cup of coffee when they go to the vending machine, but they do keep you fresh, and it stops you from going human-shy. Instead I changed the settings to not play a notification sound when new Tweets arrive so it doesn't disturb me when I'm zoning.

DON'TS

CSS galleries

Don't start browsing all kinds of CSS galleries. Either you'll feel bad, or you just start using elements in a way you can't call "inspired" anymore. Instead gather your own collection of inspiration. Example: I use LittleSnapper in which I dump everything I find inspiring. This goes from a smart layout idea, to a failed picture someone posted on Flickr. **Everything is inspiring.**

Panicking

Don't panic. It's the worst thing you could do. Instead, get away from the computer, and go to bed early. A good night of sleep combined with a hot/cold shower can give you a totally new perspective on a design. Got a deadline by tomorrow? Well, you should've started earlier. Got a good excuse to start on this design this late? Tell your client it was either that or a bad design.

120-hour work-week

Don't work all day long, including evenings and early mornings. Write off that first hour, you don't really think you'll get anything productive done before 9AM?! I don't even think you should work on one and the same design all day long. If you're stuck, try working in blocks of 1 or 2 hours on a certain design. Mixing projects isn't for everyone, but it might just do the trick for you.

SUMMARY

- Use grids, not only for layout purposes.
- Pick a specific element to start with.
- Use a colour palette.

- Limit the amount of fonts and variations you'll use.
- Search for the smallest monitor around, and restrict yourself to that one.
- Reduce the amount of noise.
- Don't start looking on the internet for inspiration. Build your own little *inspirarchive*.
- Work in blocks.

ABOUT THE AUTHOR

Tim Van Damme is a freelance interface designer at Made by Elephant. Not afraid to push the limits, friend of all things living, blabbermouth, honest chap, passionate about the web, always in the mood for a chat, blogger at Maxvoltar, boyfriend of Gwenny, Belgian, Twitter addict.

15. Making Modular Layout Systems

Jason Santa Maria 24ways.org/200815

For all of the advantages the web has with distribution of content, I've always lamented the handiness of the WYSIWYG design tools from the print publishing world. When I set out to redesign my personal website, I wanted to have some of the same abilities that those tools have, laying out pages how I saw fit, and that meant a flexible system for dealing with imagery.

Building on some of the CSS that Eric Meyer employed a few years back on the A List Apart design, I created a set of classes to use together to achieve the variety I was after. Employing multiple classes isn't a new technique, but most examples aren't coming at this from strictly editorial and visual perspectives; I wanted to have options to vary my layouts depending on content.

If you want to skip ahead, you can view the example first.

LAYING THE FOUNDATION

We need to be able to map out our page so that we have predictable canvas, and then create a system of image sizes that work with it. For the sake of this article, let's use a simple uniform 7-column grid, consisting of seven 100px-wide columns and 10px of space between each column, though you can use any measurements you want as long as they remain constant.

All of our images will have a width that references the grid column widths (in our example, 100px, 210px, 320px, 430px, 540px, 650px, or 760px), but the height can be as large as needed.

Once we know our images will all have one of those widths, we can setup our CSS to deal with the variations in layout. In the most basic form, we're going to be dealing with three classes: one each that represent an identifier, a size, and a placement for our elements.

This is really a process of abstracting the important qualities of what you would do with a given image in a layout into separate classes, allowing you to quickly customize their appearance by combining the appropriate classes. Rather than trying to serve up a one-size-fits-all approach to styling, we give each class only one or two attributes and rely on the combination of classes to get us there.

Identifier

This specifies what kind of element we have: usually either an image (`pic`) or some piece of text (`caption`).

Size

Since we know how our grid is constructed and the potential widths of our images, we can knock out a space equal to the width of any number of columns. In our example, that value can be `one`, `two`, `three`, `four`, `five`, `six`, or `seven`.

Placement

This tells the element where to go. In our example we can use a class of `left` or `right`, which sets the appropriate floating rule.

Additions

I created a few additions that be tacked on after the "placement" in the class stack: `solo`, for a bit more space beneath images without captions, `frame` for images that need a border, and `inset` for an element that appears inside of a block of text. Outset images are my default, but you could easily switch the default concept to use inset images and create a class of `outset` to pull them out of the content columns.

THE CSS

```
/* I D E N T I F I E R */
.pic p, .caption {
    font-size: 11px;
    line-height: 16px;
    font-family: Verdana, Arial, sans-serif;
    color: #666;
    margin: 4px 0 10px;
}
/* P L A C E M E N T */
.left {float: left; margin-right: 20px;}
.right {float: right; margin-left: 20px;}
.right.inset {margin: 0 120px 0 20px;} /* img floated
right within text */
.left.inset {margin-left: 230px;} /* img floated left
within text */
/* S I Z E */
.one {width: 100px;}
.two {width: 210px;}
.three {width: 320px;}
.four {width: 430px;}
.five {width: 540px;}
.six {width: 650px;}
.seven {width: 760px;}
.eight {width: 870px;}
```

```
/* A D D I T I O N S */
.frame {border: 1px solid #999;}
.solo img {margin-bottom: 20px;}
```

In Use

You can already see how powerful this approach can be. If you want an image and a caption on the left to stretch over half of the page, you would use:

```
<div class="pic four left">
  <img src="image.jpg" />
  <p>Caption goes here.</p>
</div>
```

Or, for that same image with a border and no caption:

```
<img src="image.jpg" class="pic four left frame solo"/>
```

You just tack on the classes that contain the qualities you need. And because we've kept each class so simple, we can apply these same stylings to other elements too:

```
<p class="caption two left">Caption goes here.</p>
```

Caveats

Obviously there are some potential semantic hang-ups with these methods. While classes like `pic` and `caption` stem the tide a bit, others like `left` and `right` are tougher to justify. This is something that you have to decide for yourself; I'm fine with the occasional `four` or `left` class because I think there's a good tradeoff. Just as a fully semantic solution to this problem would likely be imperfect, this solution is imperfect from the other side of the semantic fence. Additionally, IE6 doesn't understand the chain of classes within a CSS selector (like `.right.inset`). If you need to support IE6, you may have to write a few more CSS rules to accommodate any discrepancies.

Opportunities

This is clearly a simple example, but starting with a modular foundation like this leaves the door open for opportunity. We've created a highly flexible and human-readable system for layout manipulation. Obviously, this is something that would need to be tailored to the spacing and sizes of your site, but the systematic approach is very powerful, especially for editorial websites whose articles might have lots of images of varying sizes. It may not get us fully to the flexibility of WYSIWYG print layouts, but methods like this point us in a direction of designs that can adapt to the needs of the content.

View the example: without grid and with grid.

ABOUT THE AUTHOR

Jason Santa Maria is a graphic designer from sunny Brooklyn, NY. He currently works as Creative Director for Happy Cog Studios, a web design consultancy, and A List Apart, an online magazine for people who make websites. He maintains a personal site where discussion of design, film, and sock monkeys can often be observed. His work has garnered him awards and pleasantries ranging from firm handshakes to forceful handshakes with a little hitting. Ever the design obsessif, Jason is known to take drunken arguments to fisticuffs over such frivolities as kerning and white space.

16. What Your Turkey Can Teach You About Project Management

Meri Williams 24ways.org/200816

The problem with project management is that everyone thinks it's boring. Well, that's not really the problem. The problem is that everyone thinks it's boring but it's still really important. Project management is what lets you deliver your art – whether that be design or development.

In the same way, a Christmas dinner cooked by a brilliant chef with no organizational skills is disastrous – courses arrive in the wrong order, some things are cold whilst others are raw and generally it's a trip to the ER waiting to happen. Continuing the Christmas dinner theme, here are my top tips for successful projects, wrapped up in a nice little festive analogy. Enjoy!

TIP 1: KNOW WHAT YOU'RE AIMING FOR

(Turkey? Ham? Both??)

The underlying cause for the failure of so many projects is mismatched expectations. Christmas dinner cannot be a success if you serve glazed ham and your guests view turkey as the essential Christmas dinner ingredient. It doesn't matter how delicious and well executed your glazed ham is, it's still fundamentally just not turkey. You might win one or two adventurous souls over, but the rest will go home disappointed.

Add to the mix the fact that most web design projects are nowhere near as emotive as Christmas dinner (trust me, a ham vs turkey debate will rage much longer than a fixed vs fluid debate in normal human circles) and the problem is compounded. In particular, as technologists, we forget that our ability to precisely imagine the outcome of a project, be it a website, a piece of software, or similar, is much more keenly developed than the average customer of such projects.

So what's the solution? **Get very clear, from the very beginning, on exactly what the project is about.** What are you trying to achieve? How will you measure success? Is the presence of turkey a critical success factor?

Summarize all this information in some form of document (in PM-speak, it's called a Project Initiation Document typically). Ideally, get the people who are the real decision makers to sign their agreement to that summary in their own blood. Well, you get the picture, I suppose actual blood is not strictly necessary, but a bit of gothic music to set the tone can be useful!

TIP 2: PLAN AT THE RIGHT LEVEL OF DETAIL

Hugely detailed and useless Gantt charts are a personal bugbear of mine. For any project, you should plan at the appropriate level of detail (and in an appropriate format) for the project itself. In our Christmas dinner example, it may be perfectly fine to have a list of tasks for the preparation work, but for the intricate interplay of oven availability and cooking times, something more complex is usually due. Having cooked roast dinners for fourteen in a student house where only the top oven and two of the rings on the hob actually worked, I can attest to the need for sequence diagrams in some of these situations!

The mistake many small teams make is to end up with a project plan that is really the amalgamation of their individual todo lists. What is needed is a project plan that will:

1. reflect reality
2. be easy to update

3. help to track progress (i.e. are we on track or not?)

A good approach is to break your project into stages (each representing something tangible) and then into deliverables (again, something tangible for each milestone, else you'll never know if you've hit it or not!).

My personal rule of thumb is that the level of granularity needed on most projects is 2-3 days – i.e. we should never be more than two to three days from a definitive milestone which will either be complete or not. The added advantage of this approach is that if find yourself off track, you can only be two to three days off track… much easier to make up than if you went weeks or even months working hard but not actually delivering what was needed!

In our Christmas dinner example, there are a number of critical milestones – a tick list of questions. Do we have all the ingredients? Check. Has the turkey been basted? Check. On the actual day, the sequencing and timing will mean more specific questions: It's 12pm. Are the Brussels sprouts cooked to death yet? Check. (Allowing for the extra hour of boiling to go from soft and green to mushy and brown… Yeuch!)

TIP 3: ACTIVELY MANAGE RISKS AND ISSUES

A risk is something that could go wrong. An issue is something that has already gone wrong. Risks and issues are where project management superstars are born. Anyone can manage things when everything is going according to plan; it's what you do when Cousin Jim refuses to eat anything but strawberry jam sandwiches that sorts the men from the boys.

The key with a Christmas dinner, as with any project, is to have contingency plans for the most likely and most damaging risks. These depend on your own particular situation, but some examples might be:

RISK	CONTINGENCY PLAN
Cousin Jim is a picky eater.	Have strawberry jam and sliced white bread on hand to placate.
Prime organic turkey might not be available at Waitrose on Christmas eve.	Shop in advance!
You live somewhere remote that seems to lose power around Christmas on a disturbingly regular basis.	(number of options here depending on how far you want to go…) Buy a backup generator. Invent a new cooking method using only candles. Stock up on “Christmas dinner in a tin”.
Your mother in law is likely to be annoying.	Bottle of sherry at the ready (whether it’s for you or her, you can decide!).

The point of planning in advance is so that most of your issues don’t blindside you – you can spring into action with the contingency plan immediately. This leaves you with plenty of ingenuity and ability to cope in reserve for those truly unexpected events.

Back in your regular projects, you should have a risk management plan (developed at the beginning of the project and regularly reviewed) as well as an issue list, tracking open, in progress and closed issues. Importantly, your issue list should be separate from any kind of bug list – issues are at a project level, bugs are at a technical level.

TIP 4: HAVE A PROJECT BOARD

A project board consists of the overall sponsor of your project (often, but not always, the guy with the cheque book) and typically a business expert and a technical expert to help advise the sponsor. The project board is the entity that is meant to make the big, critical decisions. As a project manager, your role is to prepare a recommendation, but leave the actual decision up to the board.

Admittedly this is where our Christmas dinner analogy has to stretch the most, but if you imagine that instead of just cooking for your family you are the caterer preparing a Christmas feast for a company. In this case, you obviously want to please the diners who will be eating the food, but key decisions are likely to be taken by whoever is organizing the event. They, in turn, will involve the boss if there are really big decisions that would affect the project drastically – for instance, having to move it to January, or it exceeding the set budget by a significant amount.

Most projects suffer from not having a project board to consult for these major decisions, or from having the wrong people selected. The first ailment is eased by ensuring that you have a functioning project board, with whom you either meet regularly to update on status, or where there is a special process for convening the board if they are needed. The second problem is a little more subtle. Key questions to ask yourself are:

- Who is funding this project?
- Who has the authority to stop the project if it was the right thing to do?
- Who are the right business and technical advisors?
- Who are the folks who don't look like they are powerful on the org chart, but in fact might scupper this project? (e.g. administrators, tech support, personal assistants…)

TIP 5: FINISH UNEQUIVOCABLY AND WELL

No one is ever uncertain as to when Christmas dinner ends. Once the flaming pudding has been consumed and the cheese tray picked at, the end of the dinner is heralded by groaning and everyone collapsing in their chairs. Different households have different rituals, so you might only open your presents after Christmas dinner (unlikely if you have small children!), or you might round off the afternoon watching the Queen's speech (in Britland, certainly) or if you live in warmer climes you might round off Christmas dinner with a swim (which was our tradition in Cape Town – after 30 mins of food settling so you didn't get cramp, of course!).

The problem with projects is that they are one time efforts and so nowhere near as ritualized. Unless you have been incredibly lucky, you've probably worked on a project where you thought you were finished but seemed unable to lose your "zombie customers" – those folks who just didn't realise it was over and kept coming back with more and more requests. You might even have fallen prey to this yourself, believing that the website going live was the end of the project and not realising that a number of things still needed to be wrapped up.

The essence of this final tip is to inject some of that end-of-Christmas finality ritual into your projects. Find your own ritual for closing down projects – more than just sending the customer the invoice and archiving the files. Consider things like documentation, support structure handover and training to make sure that those zombies are going to the right people (hopefully not you!).

So, to summarise:

1. Make sure you start your projects well – with an agreed (written) vision of what you're trying to achieve.

2. Plan your projects at the right level of detail and in an appropriate format – never be more than a few days away from knowing for sure whether you're on track or not.
3. Plan for likely and important risks and make sure you track and resolve those you actually encounter.
4. Institute a project board, made up of the people with the real power over your project.
5. Create rituals for closing projects well – don't leave anyone in doubt that the project has been delivered, or of who they should go to for further help.

ABOUT THE AUTHOR

Meri Williams is a geek, a manager and a manager of geeks. She's led teams ranging in size from 30 to 300, mostly with folks spread across the world. After starting her career as a developer, she moved on to project and then product management before moving into engineering and operations management.

A published author and speaker, she sponsors scholarships to encourage more young women into STEM careers in her hometown of Stellenbosch, South Africa. You can follow her on Twitter at @Geek_Manager or read her blog at blog.geekmanager.co.uk

17. A Festive Type Folly

Jon Tan 24ways.org/200817

‘Tis the season to be jolly, so the carol singers tell us. At 24 ways, we’re keeping alive another British tradition that includes the odd faux-Greco-Roman building dotted around the British countryside, Tower Bridge built in 1894, and your Dad’s Christmas jumper with the dancing reindeer motif. ‘Tis the season of the folly!

24 Ways to impress your friends

The example is not an image, just text. You may wish to see a screenshot in Safari to compare with your own operating system and browser rendering.

Like all follies this is an embellishment — a bit of web typography fun. It’s similar to the masthead text at my place, but it’s also a hyperlink. Unlike the architectural follies of the past, no child labour was used to fund or build it, just some HTML flavoured with CSS, and a heavy dose of Times New Roman. Why Times New Roman, you ask? Well, after a few wasted hours experimenting with heaps of typefaces, seeking an elusive consistency of positioning and rendering across platforms, it proved to be the most consistent. Who’d‘a thought? To make things more interesting, I wanted to use a traditional scale and make the whole thing elastic by using relative lengths that would react to a person’s font size. So, to the meat of this festive frippery:

There are three things we rely on to create this indulgence:

1. Descendant selectors

2. Absolute positioning
3. Inheritance

HTML & Descendant Selectors

The markup for the folly might seem complex at first glance. To semantics pedants and purists it may seem outrageous. If that’s you, read on at your peril! Here it is with lots of whitespace:

```
<div id="type">
<h1>
  <a href="/">
    <em>2
      <span>4
        <span>w
          <span>a
            <span>y
              <span>s</span>
            </span>
          </span>
        </span>
      </span>
    </em>
    <strong>to
      <span>i
        <span>m
          <span>pre
            <span>s
              <span>s
                <span>your
                  <span>friends</span>
                </span>
              </span>
            </span>
          </span>
        </span>
      </span>
    </strong>
  </a>
</h1>
</div>
```

Why so much markup? Well, we want to individually style many of the glyphs. By nesting the elements, we can pick out the bits we need as descendant selectors.

To retain a smidgen of semantics, the text is wrapped in `<h1>` and `<a>` elements. The two phrases, "24 ways" and "to impress your friends" are wrapped in `<em>` and `<strong>` tags, respectively. Within those loving arms, their descendant `<span>`s cascade invisibly, making a right mess of our source, but ready to be picked out in our CSS rules.

So, to select the "2" from the example we can simply write, `#type h1 em{ }`. Of course, that selects everything within the `<em>` tags, but as we drill down the document tree, selecting other glyphs, any property / value styles can be reset or changed as required.

PIXELS VERSUS EMS

Before we get stuck into the CSS, I should say that the goal here is to have everything expressed in relative "em" lengths. However, when I'm starting out, I use pixels for all values, and only convert them to ems after I've finished. It saves re-calculating the em length for every change I make as the folly evolves, but still makes the final result elastic, *without* relying on browser zoom.

To skip ahead, see the complete CSS.

Absolutely Positioned Glyphs

If a parent element has `position: relative`, or `position: absolute` applied to it, all children of that parent can be positioned absolutely relative to it. (See Dave Shea's excellent introduction to this.) That's exactly how the folly is achieved. As the parent, `#type` also has a `font-size` of 16px set, a width and height, and some basic style with a background and border:

```
#type{
  font-size: 16px;
  text-align: left;
  background: #e8e9de;
  border: 0.375em solid #fff;
  width: 22.5em;
  height: 13.125em;
  position: relative;
}
```

The `h1` is also given a default style with a `font-size` of 132px in ems relative to the parent `font-size` of 16px:

```
#type h1{
  font-family: "Times New Roman", serif;
  font-size: 8.25em; /* 132px */
  line-height: 1em;
  font-weight: 400;
  margin: 0;
  padding: 0;
}
```

To get the em value, we divide the *required size* in pixels by the actual parent `font-size` in pixels

132 ÷ 16 = 8.25

We also give the descendants of the `h1` some default properties. The line height, style and weight are normalised, they are positioned absolutely relative to `#type`, and a border and padding is applied:

```
#type h1 em,
#type h1 strong,
#type h1 span{
  line-height: 1em;
  font-style: normal;
  font-weight: 400;
  position: absolute;
  padding: 0.1em;
  border: 1px solid transparent;
}
```

The padding ensures that some browsers don't forget about parts of a glyph that are drawn outside of their invisible container. When this happens, IE will trim the glyph, cutting off parts of descenders, for example. The border is there to make sure the glyphs have layout. Without this, positioning can be problematic. IE6 will not respect the `transparent` border colour — it uses the actual text colour — but in all other respects renders the example. You can hack around it, but it seemed unnecessary for this example.

Once these defaults are established, the rest is trial and error. As a quick example, the numeral "2" is first to be positioned:

```
#type h1 a em{
  font-size: 0.727em; /* (2) 96px */
  left: 0.667em;
  top: 0;
}
```

Every element of the folly is positioned in exactly the same way as you can see in the complete CSS. When converting pixels to ems, the `font-size` is set first. Then, because we know what that is, we calculate the equivalent x- and y-position accordingly.

Inheritance

CSS inheritance gave me a headache a long time ago when I first encountered it. After the penny dropped I came to experience something disturbingly close to affection for this characteristic. What it basically means is that children inherit the characteristics of their parents. For example:

1. We gave `#type` a `font-size` of 16px.
2. For `#type h1` we changed it by setting `font-size: 8.25em;`. Than means that `#type h1` now has a computed `font-size` of 8.25 × 16px = 132px.
3. Now, all children of `#type h1` in the document tree will inherit a `font-size` of 132px unless we explicitly change it as we did for `#type h1 a em`.

The “2” in the example — selected with `#type h1 a em` — is set at 96px with `left` and `top` positioning calculated relatively to that. So, the `left` position of `0.667em` is 0.667 × 96 = 64px, approximately (three decimal points in em lengths don’t always give exact pixel equivalents).

One way to look at inheritance is as a cascade of dependancy: In our example, the computed font size of any given element *depends* on that of the parent, and the absolute x- and y-position *depends* on the computed font size of the element itself.

LINK COLOURS

The same descendant selectors we use to set and position the type are also used to apply the colour by combining them with pseudo-selectors like `:focus` and `:hover`. Because the descendant selectors are available to us, we can pretty much pick out any glyph we like. First, we need to disable the underline:

```
#type h1 a:link,
#type h1 a:visited{
  text-decoration: none;
}
```

In our example, the "24" has a unique default state (colour):

```
#type h1 a:link em,
#type h1 a:visited em{
  color: #624;
}
```

The rest of the "Ways" text has a different colour, which it shares with the large "s" in "impress":

```
#type h1 a:link em span span,
#type h1 a:visited em span span,
#type h1 a:link strong span span span span,
#type h1 a:visited strong span span span span{
  color: #b32720;
}
```

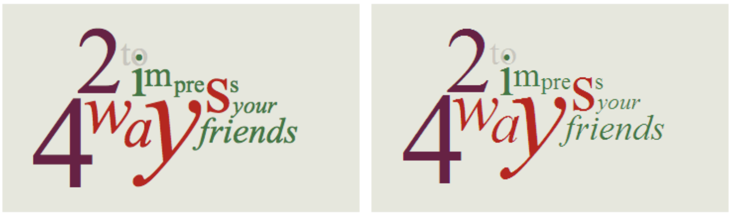
"24" changes on `:focus`, `:hover` and `:active`. Critically though, the whole of the "24 Ways" text, and the large "s" in "impress" all have the same style in this instance:

```
#type h1 a:focus em,
#type h1 a:hover em,
#type h1 a:active em,
#type h1 a:focus em span span,
#type h1 a:hover em span span,
#type h1 a:active em span span,
#type h1 a:focus strong span span span span,
#type h1 a:hover strong span span span span,
#type h1 a:active strong span span span span{
  color: #804;
}
```

If a descendant selector has a `:link` and `:visited` state set as a pseudo element, it needs to also have the corresponding `:focus`, `:hover` and `:active` states set.

A FINAL NOTE ABOUT WEB TYPOGRAPHY

From grids to basic leading to web fonts, and even absolute positioning, there's a wealth of things we can do to treat type on the Web with love and respect. However, experiments like this can highlight the vagaries of rasterisation and rendering that limit our ability to achieve truly subtle and refined results. At the operating system level, the differences in type rendering are extreme, and even between sequential iterations in Windows — from Standard to ClearType — they can be daunting. Add to that huge variations in screen quality, and even the paper we print our type onto has many potential variations. Compare our example in Safari 3.2.1 / OS X 10.5.5 (left) and IE7 / Win XP (right). Both rendered on a 23” Apple Cinema HD (LCD):


24 ways to impress your friends
24 ways to impress your friends

Browser developers continue to make great strides. However, those of us who set type on the Web need more consistency and quality if we want to avoid technologies like Flash and evolve web typography. Although web typography is inevitably — and mistakenly — compared unfavourably to print, it has the potential to achieve the same refinement in a different way. Perhaps one day, the glyphs of our favourite faces, so carefully crafted, kerned and hinted for the screen, will be rendered with the same precision with which they were drawn by type designers and styled by web designers. That would be my wish for the new year. Happy holidays!

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.

18. Shiny Happy Buttons

John Allsopp 24ways.org/200818

Since Mac OS X burst onto our screens, glossy, glassy, shiny buttons have been almost de rigeur, and have essentially, along with reflections and rounded corners, become a cliché of Web 2.0 “design”. But if you can’t beat ‘em you’d better join ‘em. So, in this little contribution to our advent calendar, we’re going to take a plain old boring HTML button, and 2.0 it up the wazoo.

But, here’s the catch. We’ll use no images, either in our HTML or our CSS. No sliding doors, no image replacement techniques. Just straight up, CSS, CSS3 and a bit of experimental CSS. And, it will be compatible with pretty much any browser (though with some progressive enhancement for those who keep up with the latest browsers).

THE HTML

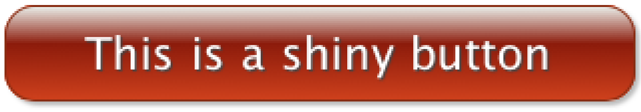
We’ll start with our HTML.

```
<button type="submit">This is a shiny button</button>
```

OK, so it's not shiny yet – but boy will it ever be.

Before styling, that's going to look like this.

Ironically, depending on the operating system and browser you are using, it may well be a shiny button already, but that's not the point. We want to make it shiny 2.0. Our mission is to make it look something like this



If you want to follow along at home keep in mind that depending on which browser you are using you may see fewer of the CSS effects we've added to create the button. As of writing, only in Safari are all the effects we'll apply supported.

Taking a look at our finished product, here's what we've done to it:

1. We've given the button some padding and a width.
2. We've changed the text color, and given the text a drop shadow.
3. We've given the button a border.
4. We've given the button some rounded corners.
5. We've given the button a drop shadow.
6. We've given the button a gradient background.

and remember, all without using any images.

STYLING THE BUTTON

So, let’s get to work.

First, we’ll add given the element some padding and a width:

```
button {
  padding: .5em;
  width: 15em;
}
```

Next, we’ll add the text color, and the drop shadow:

```
color: #ffffff;
text-shadow: 1px 1px 1px #000;
```

A note on text-shadow

If you’ve not seen text-shadows before well, here’s the quick back-story. Text shadow was introduced in CSS2, but only supported in Safari (version 1!) some years later. It was removed from CSS2.1, but returned in CSS3 (in the text module). It’s now supported in Safari, Opera and Firefox (3.1). Internet Explorer has a shadow filter, but the syntax is completely different.

So, how do text-shadows work? The three length values specify respectively a horizontal offset, a vertical offset and a blur (the greater the number the more blurred the shadow will be), and finally a color value for the shadow.

Rounding the corners

Now we'll add a border, and round the corners of the element:

```
border: solid thin #882d13;
-webkit-border-radius: .7em;
-moz-border-radius: .7em;
border-radius: .7em;
```

Here, we've used the same property in three slightly different forms. We add the browser specific prefix for Webkit and Mozilla browsers, because right now, both of these browsers only support border radius as an experimental property. We also add the standard property name, for browsers that do support the property fully in the future.

The benefit of the browser specific prefix is that if a browser only partly supports a given property, we can easily avoid using the property with that browser simply by not adding the browser specific prefix. At present, as you might guess, `border-radius` is supported in Safari and Firefox, but in each the relevant prefix is required.

`border-radius` takes a length value, such as pixels. (It can also take two length values, but that's for another Christmas.) In this case, as with padding, I've used ems, which means that as the user scales the size of text up and down, the radius will scale as well. You can test the difference by making the radius have a value of say 5px, and then zooming up and down the text size.

We're well and truly on the way now. All we need to do is add a shadow to the button, and then a gradient background.

In CSS3 there's the `box-shadow` property, currently only supported in Safari 3. It's very similar to `text-shadow` – you specify a horizontal and vertical offset, a blur value and a color.

```
-webkit-box-shadow: 2px 2px 3px #999;
box-shadow: 2px 2px 2px #bbb;
```

Once more, we require the "experimental" `-webkit-` prefix, as Safari's support for this property is still considered by its developers to be less than perfect.

Gradient Background

So, all we have left now is to add our shiny gradient effect. Now of course, people have been doing this kind of thing with images for a long time. But if we can avoid them all the better. Smaller pages, faster downloads, and more scalable designs that adapt better to the user's font size preference. But how can we add a gradient background without an image?

Here we'll look at the only property that is not as yet part of the CSS standard – Apple's gradient function for use anywhere you can use images with CSS (in this case backgrounds). In essence, this takes SVG gradients, and makes them available via CSS syntax.

Here's what the property and its value looks like:

```
background-image: -webkit-gradient(linear, left top,
left bottom, from(#e9ede8), to(#ce401c),color-stop(0.4,
#8c1b0b));
```

Zooming in on the gradient function, it has this basic form:

```
-webkit-gradient(type, point, point, from(color),
to(color),color-stop(where, color));
```

Which might look complicated, but is less so than at first glance.

The name of the function is `gradient` (and in this case, because it is an experimental property, we use the – `webkit-` prefix).

You might not have seen CSS functions before, but there are others, including the `attr()` function, used with generated content. A function returns a value that can be used as a property value – here we are using it as a background image.

Next we specify the type of the gradient. Here we have a linear gradient, and there are also radial gradients.

After that, we specify the start and end points of the gradient – in our case the top and bottom of the element, in a vertical line.

We then specify the start and end colors – and finally one stop color, located at 40% of the way down the element. Together, this creates a gradient that smoothly transitions from the start color in the top, vertically to the stop color, then smoothly transitions to the end color.

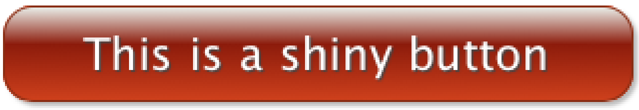
There's one last thing. What color will the background of our button be if the browser doesn't support gradients? It will be white (or possibly some default color for buttons). Which may make the text difficult or impossible to read. So, we'll add a background color as well (see why the validator is always warning you when a color but not a background color is specified for an element?).

If we put it all together, here's what we have:

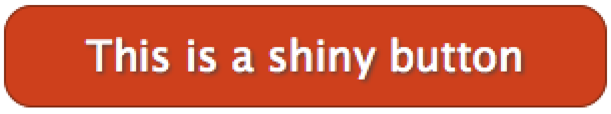
```
button {
  width: 15em;
  padding: .5em;
  color: #ffffff;
  text-shadow: 1px 1px 1px #000;
  border: solid thin #882d13;
  -webkit-border-radius: .7em;
  -moz-border-radius: .7em;
  border-radius: .7em;
  -webkit-box-shadow: 2px 2px 3px #999;
  box-shadow: 2px 2px 2px #bbb;
  background-color: #ce401c;
  background-image: -webkit-gradient(linear, left top,
left bottom, from(#e9ede8), to(#ce401c),color-stop(0.4,
#8c1b0b));
}
```

Which looks like this in various browsers:

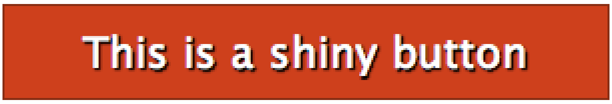
In Safari (3)



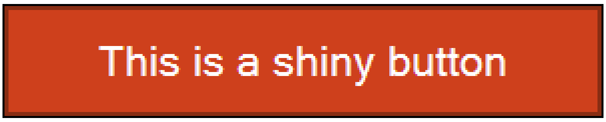
In Firefox 3.1 (3.0 supports border-radius but not text-shadow)



In Opera 10



and of course in Internet Explorer (version 8 shown here)



But it looks different in different browsers

Yes, it **does** look different in different browsers, but we all know the answer to the question “do web sites need to look the same in every browser?“.

Even if you really think sites should look the same in every browser, hopefully this little tutorial has whet your appetite for what CSS3 and experimental CSS that’s already supported in widely used browsers (and we haven’t even touched on animations and similar effects!).

I hope you’ve enjoyed out little CSSMas present, and look forward to seeing your shiny buttons everywhere on the web.

Oh, and there's just a bit of homework – your job is to use the `:hover` selector, and make a gradient in the hover state.

ABOUT THE AUTHOR

John Allsopp is a founder of Westciv, an Australian web software development and training company, which provides some of the best CSS resources and tutorials on the web. Westciv's software and training are used in dozens of countries around the World. The head developer of the leading cross platform CSS editor, Style Master, John has written on web development issues for numerous web and print publications and was one of the earliest members of the Web Standards Project.

19. Moo'y Christmas

Brian Suda

A note from the editors: Moo has changed their API since this article was written.

24ways.org/200819

As the web matures, it is less and less just about the virtual world. It is becoming entangled with our world and it is harder to tell what is virtual and what is real. There are several companies who are blurring this line and make the virtual just an extension of the physical. Moo is one such company.

Moo offers simple print on demand services. You can print business cards, moo mini cards, stickers, postcards and more. They give you the ability to upload your images, customize them, then have them sent to your door. Many companies allow this sort of digital to physical interaction, but Moo has taken it one step further and has built an API.

PRINTABLE STOCKING STUFFERS

The Moo API consists of a simple XML file that is sent to their servers. It describes all the information needed to dynamically assemble and print your object. This is very helpful, not just for when you want to print your own stickers, but when you want to offer them to your customers, friends, organization or community with no hassle. Moo handles the check-out and shipping, all you need to do is what you do best, create!

Now using an API sounds complicated, but it is actually very easy. I am going to walk you through the options so you can easily be printing in no time.

Before you can begin sending data to the Moo API, you need to register and get an API key. This is important, because it allows Moo to track usage and to credit you. To register, visit http://www.moo.com/api/ and click "Request an API key".

In the following examples, I will use `{YOUR API KEY HERE}` as a place holder, replace that with your API key and everything will work fine.

First thing you need to do is to create an XML file to describe the check-out basket. Open any text-editor and start with some XML basics. Don't worry, this is pretty simple and Moo gives you a few tools to check your XML for errors before you order.

```
<?xml version="1.0" encoding="UTF-8"?>
<moo xsi:noNamespaceSchemaLocation="http://www.moo.com/
xsd/api_0.7.xsd" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
   <request>
```

```
     <version>0.7</version>
     <api_key>{YOUR API KEY HERE}</api_key>
     <call>build</call>
     <return_to>http://www.example.com/
return.html</return_to>
     <fail_to>http://www.example.com/fail.html</fail_to>
   </request>
   <payload>
   ...
   </payload>
</moo>
```

Much like HTML's `<head>` and `<body>`, Moo has created `<request>` and `<payload>` elements all wrapped in a `<moo>` element.

The `<request>` element contains a few pieces of information that is the same across all the API calls. The `<version>` element describes which version of the API is being used. This is more important for Moo than for you, so just stick with "0.7" for now.

The `<api_key>` allows Moo to track sales, referrers and credit your account.

The `<call>` element can only take "build" so that is pretty straight forward. The `<return_to>` and `<fail_to>` elements are URLs. These are optional and are the URLs the customer is redirected to if there is an error, or when the check out process is complete. This allows for some basic branding and a custom “thank you” page which is under your control. That’s it for the `<request>` element, pretty easy so far!

Next up is the `<payload>` element. What goes inside here describes what is to be printed. There are two possible elements, we can put `<chooser>` or we can put `<products>` directly inside `<payload>`. They work in a similar ways, but they drop the customer into different parts of the Moo checkout process.

If you specify `<products>` then you send the customer straight to the Moo payment process. If you specify `<chooser>` then you send the customer one-step earlier where they are allowed to pick and choose some images, remove the ones they don’t like, adjust the crop, etc. The example here will use `<chooser>` but with a little bit of homework you can easily adjust to `<products>` if you desire.

```
...
<chooser>
   <product_type>sticker</product_type>
   <images>
     <url>http://example.com/images/christmas1.jpg</url>
   </images>
</chooser>
...
```

Inside the `<chooser>` element, we can see there are two basic piece of information. The type of product we want to print, and the images that are to be printed. The `<product_type>` element can take one of five options and is required! The possibilities are: minicard, notecard, sticker, postcard or greetingcard. We'll now look at two of these more closely.

MOO STICKERS

In the Moo sticker books you get 90 small squarish stickers in a small little booklet.

The simplest XML you could send would be something like the following payload:

```
...
<payload>
  <chooser>
    <product_type>sticker</product_type>
    <images>
      <url>http://example.com/image1.jpg</url>
    </images>
    <images>
      <url>http://example.com/image2.jpg</url>
    </images>
    <images>
      <url>http://example.com/image3.jpg</url>
    </images>
  </chooser>
</payload>
...
```

This creates a sticker book with only 3 unique images, but 30 copies of each image. The Sticker books always print 90 stickers in multiples of the images you uploaded. That example only has 3 `<images>` elements, but you can easily duplicate the XML and send up to 90. The `<url>` should be the full path to your image and the image needs to be a minimum of 300 pixels by 300 pixels.

You can add more XML to describe cropping, but the simplest option is to either, let your customers choose or to pre-crop all your images square so there are no issues.

The full XML you would post to the Moo API to print sticker books would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<moo xsi:noNamespaceSchemaLocation="http://www.moo.com/
xsd/api_0.7.xsd" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
  <request>
    <version>0.7</version>
    <api_key>{YOUR API KEY HERE}</api_key>
    <call>build</call>
    <return_to>http://www.example.com/
return.html</return_to>
    <fail_to>http://www.example.com/fail.html</fail_to>
  </request>
  <payload>
    <chooser>
      <product_type>sticker</product_type>
      <images>
        <url>http://example.com/image1.jpg</url>
      </images>
      <images>
        <url>http://example.com/image2.jpg</url>
      </images>
      <images>
        <url>http://example.com/image3.jpg</url>
      </images>
    </chooser>
  </payload>
</moo>
```

MINI-CARDS

The mini-cards are the small cute business cards in 14×35 dimensions and come in packs of 100.

Since the mini-cards are print on demand, this allows you to have 100 unique images on the back of the cards.

Just like the stickers example, we need the same XML setup. The `<moo>` element and `<request>` elements will be the same as before. The part you will focus on is the `<payload>` section.

Since you are sending along specific information, we can't use the `<chooser>` option any more. Switch this to `<products>` which has a child of `<product>`, which in turn has a `<product_type>` and `<designs>`. This might seem like a lot of work, but once you have it set up you won't need to change it.

```
...
<payload>
  <products>
    <product>
      <product_type>minicard</product_type>
      <designs>
        ...
      </designs>
    </product>
  </products>
</payload>
...
```

So now that we have the basic framework, we can talk about the information specific to minicards. Inside the `<designs>` element, you will have one `<design>` for each card. Much like before, this contains a way to describe the image. Note that this time the element is called `<image>`, not images plural.

Inside the `<image>` element you have a `<url>` which points to where the image lives and a `<type>`. The `<type>` should just be set to 'variable'. You can pass crop information here instead, but we're going to keep it simple for this tutorial. If you are interested in how that works, you should refer to the official API documentation.

```
...
<design>
  <image>
    <url>http://example.com/image1.jpg</url>
    <type>variable</type>
  </image>
</design>
...
```

So far, we have managed to build a pack of 100 Moo mini-cards with the same image on the front. If you wanted 100 different images, you just need to replicate this snippit, 99 more times.

That describes the front design, but the flip-side of your mini-cards can contain 6 lines of text, which is customizable in a variety of colors, fonts and styles.

The API allows you to create different text on the back of each mini-card, something the web interface doesn't implement. To describe the text on the mini-card we need to add a `<text_collection>` element inside the `<design>` element. If you skip this element, the back of your mini-card will just be blank, but that's not very festive!

Inside the `<text_collection>` element, we need to describe the type of text we want to format, so we add a `<minicard>` element, which in turn contains all the lines of text. Each of Moo's printed products take different numbers of lines of text, so if you are not planning on making mini-cards, be sure to consult the documentation.

For mini-cards, we can have 6 distinct lines, each with their own style and layout. Each line is represented by an element `<text_line>` which has several optional children. The `<id>` tells which line of the 6 to print the text one. The

`<string>` is the text you want to print and it must be shorter than 38 characters. The `<bold>` element is false by default, but if you want your text bolded, then add this and set it to true.

The `<align>` element is also optional. By default it is set to align left. You can also set this to right or center if you desirer. The `<font>` element takes one of 3 types, modern, traditional or typewriter. The default is modern. Finally, you can set the `<colour>`, yes that's color with a 'u', Moo is a British company, so they get to make the rules. When you start a print on demand company, you can spell it however you want. The `<colour>` element takes a 6 character hex value with a leading #.

```
<design>
  ...
  <text_collection>
    <minicard>
      <text_line>
        <id>(1-6)</id>
        <string>String, I must be less than 38 
chars!</string>
        <bold>true</bold>
        <align>left</align>
        <font>modern</font>
        <colour>#ff0000</colour>
      </text_line>
    </minicard>
  </text_collection>
</design>
```

If you combine all of this into a mini-card request you'd get this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<moo xsi:noNamespaceSchemaLocation="http://www.moo.com/
xsd/api_0.7.xsd" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
  <request>
    <version>0.7</version>
    <api_key>{YOUR API KEY HERE}</api_key>
    <call>build</call>
    <return_to>http://www.example.com/
return.html</return_to>
    <fail_to>http://www.example.com/fail.html</fail_to>
  </request>
  <payload>
    <products>
      <product>
        <product_type>minicard</product_type>
        <designs>
          <design>
            <image>
              <url>http://example.com/image1.jpg</url>
              <type>variable</type>
            </image>
            <text_collection>
              <minicard>
                <text_line>
                <id>1</id>
                <string>String, I must be less than 38
chars!</string>
                <bold>true</bold>
                <align>left</align>
                <font>modern</font>
                <colour>#ff0000</colour>
                </text_line>
              </minicard>
            </text_collection>
          </design>
        </designs>
      </product>
    </products>
  </payload>
</moo>
```

Now you know how to construct the XML that describes what to print. Next, you need to know how to send it to Moo to make it happen!

POSTING TO THE API

So your XML is file ready to go. First thing we need to do is check it to make sure it's valid. Moo has created a simple validator where you paste in your XML, and it alerts you to problems.

When you have a fully valid XML file, you'll want to send that to the Moo API. There are a few ways to do this, but the simplest is with an HTML form.

This is the sample code for an HTML form with a big "Buy My Stickers" button. Once you know that it is working, you can use all your existing HTML knowledge to style it up any way you like.

```
<form method="POST" action="http://www.moo.com/api/
api.php">
  <input type="hidden" name="xml" value="<?xml
version="1.0" encoding="UTF-8"?> <moo
xsi:noNamespaceSchemaLocation="http://www.moo.com/xsd/
api_0.7.xsd" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"> <request>....</request>
<payload>...</payload> </moo> ">
  <input type="submit" name="submit" value="Buy My
Stickers"/>
</form>
```

This is just a basic `<form>` element that submits to the Moo API, `http://www.moo.com/api/api.php`, when someone clicks the button. There is a hidden input called "xml" which contains the value the XML file we created previously.

For those of you who need to "view source" to fully understand what's happening can see a working version and peek under the hood.

Using the API has advantages over uploading the images directly yourself. The images and text that you send via the API can be dynamic. Some companies, like Dopplr, have taken user profiles and dynamic data that changes every minute to generate customer stickers of places that you've travelled to or mini-cards with a world map of all the cities you have visited. Every single customer has different travel plans and therefore different sets of stickers and mini-card maps. The API allows for the utmost current information to be printed, on demand, in real-time.

GO FORTH AND MOO'LTIPLY

See, making an API call wasn't that hard was it? You are now 90% of the way to creating anything with the Moo API. With a bit of reading, you can learn that extra 10% and print any Moo product. Be on the lookout in 2009 for the official release of the 1.0 API with improvements and some extras that were not available when this article was written.

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. Ghosts On The Internet

Gavin Bell 24ways.org/200820

By rights the internet should be full of poltergeists, poor rootless things looking for their real homes. Many events on the internet are not properly associated with their correct timeframe. I don't mean a server set to the wrong time, though that happens too. Much of the content published on the internet is separated from any proper reference to its publication time. What does publication even mean? Let me tell you a story…

> "It is 2019 and this is Kathy Clees reporting on the story of the moment, the shock purchase of Microsoft by Apple Inc. A Internet Explorer security scare story from 2008 was responsible, yes from 11 years ago, accidently promoted by an analyst, who neglected to check the date of their sources."

If you think this is fanciful nonsense, then cast your mind back to September 2008, this story in Wired or The Times (UK) about a huge United Airlines stock tumble. A Florida newspaper had a automated popular story section. A random reader looking at a story about United's 2002 Bankruptcy proceedings caused this story to get picked up by Google's later visit to the South Florida Sun Sentinel's news home page.

The story was undated, Google's news engine apparently gave it a 2008 date, an analyst picked it up and pushed it to Bloomberg and within minutes the United stock was tumbling. Their stock price dropped from $12 to $3, then recovered to $11 over the day. An eight percent fall in share price over a mis-configured date

Completing this out of order Christmas Carol, lets look at what is current practice and how dates are managed, we might even get to clank some chains. Publication date used to be inseparable from publication, the two things where stamped on the same piece of paper. How can we determine when things have been published, now?

DETERMINING PUBLICATION DATES

Time as defined by http://www.w3.org/TR/NOTE-datetime extends ISO 8601, mandating the use of a year value. This is pretty well defined, we can even get very accurate timings down to milliseconds, Ruby and other languages can even handle Calendar reformation. So accuracy is not the issue.

One problem is that there are many dates which could be interpreted as the publication date. Publication can mean any of date written or created; date placed on server; last modified date; or the current date from the web server. Created and modified have parallels with file systems, but the large number of database driven websites means that this no longer holds much meaning, as there are no longer any files.

Checking web server `HEAD` may also not correspond, it might give the creation time for the HTML file you are viewing or it might give the last modified time for a file from disk. It is too unreliable and lacking in context to be of real value. So if the web server will not help, then how can we get the right timeframe for our content?

We are left with URLs and the actual page content.

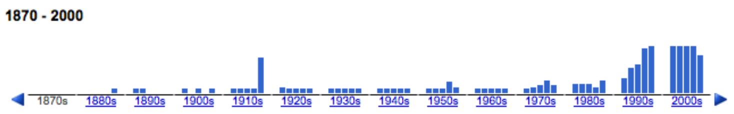
Looking at Flickr, this picture (by Douglas County History Research Center) has four date values which can be associated with it. It was taken around 1900, scanned in 1992 and placed on Flickr on July 29th, 2008 and replaced later that day. Which dates should be represented here?

This is hard question to answer, but currently the date of upload to Flickr is the best represented in terms of the date URL, `/photos/douglascountyhistory/archives/date-posted/2008/07/29/`, plus some Dublin Core RDF for the year. Flickr uses 2008 as the value for this image. Not accurate, but a reasonable compromise for the millions of other images on their site.

Flickr represents location much better than it represents time. For the most part this is fine, but once you go back in time to the 1800s then the maps of the world start to change a lot and you need to reference both time and place.

The Google timeline search offers another interesting window on the world, showing results organised by decade for any search term. Being able to jump to a specific occurrence of a term makes it easier to get primary results rather than later reporting.

The 1918 "Spanish flu" results jump out in this timeline.



Any major news event will have multiple analysis articles after the event, finding the original reporting of hurricane Katrina is harder now. Many publishers are putting older content online, e.g. Harpers or Nature or The Times, often these use good date based URLs, sometimes they are unhelpful database references. If this content is available for free, then how much better would it be to provide good metadata on date of publication.

DATE BASED URLS

A quick word on date based URLs, they can be brilliant at capturing first published date. However they can be hard to interpret. Is `/03/04` a date in March or April, what about `08/03/04`? Obviously `2008/03/04` is easier to understand, it is probably March 4th. Including a proper timestamp in the page content avoid this kind of guesswork.

Many sites represent the date as a plain text string; a few hook an HTML class of `date` around it, a very few provide an actual timestamp. Associating the date with the individual content makes it harder to get the date wrong.

Movable Type and TypePad are a notable exceptions, they will embed Dublin Core RDF to represent each posting e.g. `dc:date="2008-12-18T02:57:28-08:00"`. WordPress doesn't support date markup out of the box, though there is a patch and a howto for hAtom available.

In terms of newspapers, the BBC use `<meta name="OriginalPublicationDate" content="2008/12/18 18:52:05" />` along with opaque URLs such as `http://news.bbc.co.uk/1/hi/technology/7787335.stm`.

The Guardian use nice clear URLs `http://www.guardian.co.uk/business/2008/dec/18/car-industry-recession` but have no marked up date on the page.

The New York Times are similar to the Guardian with nice URLs, `http://www.nytimes.com/2008/12/19/business/19markets.html`, but again no timestamps. All of these papers have all the data available, but it is not marked up in a useful manner.

SYNDICATION FORMATS

Syndication formats are better at supporting dates, RSS uses RFC 822 for dates, just like email so dates such as `Wed, 17 Dec 2008 12:52:40 GMT` are valid, with all the white space issues that entails.

The Atom syndication format uses the much clearer `http://tools.ietf.org/html/rfc3339` with timestamps of the form `1996-12-19T16:39:57-08:00`. Both syndication formats encourage the use of last modified. This is understandable, but a pity as published date is a very useful value. The Atom syndication format supports "published" and mandates "updated" as timestamps, see the Atom RFC 4287 for more detail.

MARKING UP DATES

However the aim of this short article is to encourage you to use microformats or RDF to encode dates. A good example of this is Twitter, they use hAtom for each individual entry, `http://twitter.com/zzgavin/status/`

`1065835819` contains the following markup, which represents a human and a machine readable version of the time of that tweet.

```
<span class="published"
title="2008-12-18T22:01:27+00:00">about 3 hours
ago</span>
```

The spec for datetime is still draft at the minute and there is still ongoing conversation around the right format and semantics for representing date and time in microformats, see the datetime design pattern for details.

The hAtom example page shows the minimal changes required to implement hAtom on well formed blog post content and for other less well behaved content. You have the information already in your content publication systems, this is not some additional onerous content entry task, simply some template formatting.

I started to see this as a serious issue after reading Stewart Brand's Clock of the Long Now about five years ago. Brand's book explores the issues of short term thinking that permeate our society, thinking beyond the end of the financial year is a stretch for many people. The Long Now has a world view of a 10,000 year timeframe, see http://longnow.org/ for much more information. Freebase from Long Now Board member Danny Hillis, supports dates quite well – see the entry for A Christmas Carol.

IN CONCLUSION

I feel we should be making it easier for people searching for our content in the future. We've moved through tagging content and on to geo-tagging content. Now it is time to get the timestamps right on our content. How do I know when something happened and how can I find other things that happened at the same time is a fair question. This should be something I can satisfy simply and easily. There are a range of tools available to us in either hAtom or RDF to specify time accurately alongside the content, so what is stopping you?

Thinking of the long term it is hard for us to know now what will be of relevance for future generations, so we should aim to raise the floor for publishing tools so that all content has the right timeframe associated with it. We are moving from publishing words and pictures on the internet to being able to associate publication with an individual via XFN and OpenID. We can associate place quite well too, the last piece of useful metadata is timeframe.

ABOUT THE AUTHOR

Gavin Bell designs web applications and social software for the Nature Publishing Group. Large scale web applications covering identity, on-demand media and social software have been the main focus of his work. Since the early 90s he has worked in academia, advertising, publishing and developed multimedia software.

He is the author of a forthcoming book entitled Building Social Web Applications for O'Reilly Media Inc. He lives in London with his wife and two sons. He keeps track of the world on take one onion, you can keep track of him on twitter and gavinbell.com were he generally avoids the third person.

Photo: James Duncan Davidson

21. Geotag Everywhere with Fire Eagle

Ben Ward

24ways.org/200821

A note from the editors: Since this article was written Yahoo! has retired the Fire Eagle service.

Location, they say, is everywhere. Everyone has one, all of the time. But on the web, it's taken until this year to see the emergence of location in the applications we use and build.

The possibilities are broad. Increasingly, mobile phones provide SDKs to approximate your location wherever you are, browser extensions such as Loki and Mozilla's Geode provide browser-level APIs to establish your location from the proximity of wireless networks to your laptop. Yahoo's Brickhouse group launched Fire Eagle, an ambitious location broker enabling people to take their location from any of these devices or sources, and provide it to a plethora of web services. It enables you to take the location information that only your iPhone knows about and use it anywhere on the web.

That said, this is still a time of location as an emerging technology. Fire Eagle stores your location on the web (protected by application-specific access controls), but to try and give an idea of how useful and powerful your location can be — regardless of the services you use now — today's 24ways is going to build a bookmarklet to call up your location on demand, in any web application.

LOCATION SUPPORT ON THE WEB

Over the past year, the number of applications implementing location features has increased dramatically. Plazes and Brightkite are both full featured social networks based around where you are, whilst Pownce rolled in Fire Eagle support to allow geotagging of all the content you post to their microblogging service. Dipity's beautiful timeline shows for you moving from place to place and Six Apart's activity stream for Movable Type started exposing your movements.

The number of services that hook into Fire Eagle will increase as location awareness spreads through the developer community, but you can use your location on other sites indirectly too.

Consider Flickr. Now world renowned for their incredible mapping and places features, geotagging on Flickr started out as a grassroots extension of regular tagging. That same technique can be used to start rolling geotagging in any publishing platform you come across, for any kind of content. Machine-tags (`geo:lat=` and `geo:lon=`) and the adr and geo microformats can be used to enhance anything you write with location information.

A CRASH COURSE IN AVIAN INFLAMMABILITY

Fire Eagle is a location store. A broker between services and devices which provide location and those which consume it. It's a switchboard that controls which pieces of your location different applications can see and use, and keeps hidden anything you want kept private. A blog widget that displays your current location in public can be restricted to display just your current city, whilst a service that provides you with a list of the nearest ATMs will operate better with a precise street address.

Even if your iPhone tells Fire Eagle exactly where you are, consuming applications only see what you want them to see. That's important for users to realise that they're in control, but also important for application developers to remember that you cannot rely on having super-accurate information available all the time. *You need to build location aware applications which degrade gracefully*, because users will provide fuzzier information — either through choice, or through less accurate sources.

Application specific permissions are controlled through an OAuth API. Each application has a unique key, used to request a second, user-specific key that permits access to that user's information. You store that user key and it remains valid until such a time as the user revokes your application's access. Unlike with passwords, these keys are unique per application, so revoking the access rights of one application doesn't break all the others.

BUILDING YOUR FIRST FIRE EAGLE APP; GEOMARKLET

Fire Eagle's developer documentation can take you through examples of writing simple applications using server side technologies (PHP, Python). Here, we're going to write a client-side bookmarklet to make your location available in *every* site you use. It's designed to fast-track the experience of having location available everywhere on web, and show you how that can be really handy. Hopefully, this will set you thinking about how location can enhance the new applications you build in 2009.

An oddity of bookmarklets

Bookmarklets (or 'favlets', for those of an MSIE persuasion) are a strange environment to program in. Critically, you have no persistent storage available. As such, using token-auth APIs in a static environment requires you to build you application in a slightly strange way; authing yourself in advance and then hardcoding the keys into your script.

Get started

Before you do anything else, go to http://fireeagle.com and log in, get set up if you need to and by all means take a look around. Take a look at the mobile updaters section of the application gallery and perhaps pick out an app that will update Fire Eagle from your phone or laptop.

Once that's done, you need to register for an application key in the developer section. Head straight to /developer/create and complete the form. Since you're building a standalone application, choose 'Auth for desktop applications' (rather than web applications), and select that you'll be 'accessing location', not updating.

At the end of this process, you'll have two application keys, a 'Consumer Key' and a 'Consumer Secret', which look like these:

Consumer Key
: luKrM9U1pMnu

Consumer Secret
: ZZl9YXXoJX5KLiKyVrMZffNEaBnxnd6M

These keys combined allow your application to make requests to Fire Eagle.

Next up, you need to auth yourself; granting your new application permission to use your location. Because bookmarklets don't have local storage, you can't integrate the auth process into the bookmarklet itself — it would have no way of storing the returned key. Instead, I've put together a simple web frontend through which you can auth with your application.

Head to Auth me, Amadeus!, enter the application keys you just generated and hit 'Authorize with Fire Eagle'. You'll be taken to the Fire Eagle website, just as in regular Fire Eagle applications, and after granting access to your app, be redirected back to Amadeus which will provide you your user tokens. These tokens are used in subsequent requests to read your location.

And, skip to the end…

The process of building the bookmarklet, making requests to Fire Eagle, rendering it to the page and so forth follows, but if you're the impatient type, you might like to try this out right now. Take your four API keys from above, and drag the following link to your Bookmarks Toolbar; it contains all the code described below. Before you can use it, you need to edit in your own API keys. Open your browser's bookmark editor and where you find text like 'YOUR_CONSUMER_KEY_HERE', swap in the corresponding key you just generated.

Get Location

Bookmarklet Basics

To start on the bookmarklet code, set out a basic JavaScript module-pattern structure:

```
var Geomarklet = function() {
  return ({
    callback: function(json) {},
    run: function() {}
  });
};
Geomarklet.run();
```

Next we'll add the keys obtained in the setup step, and also some basic Fire Eagle support objects:

```
var Geomarklet = function() {
  var Keys = {
      consumer_key: 'IuKrJUHU1pMnu',
      consumer_secret:
'ZZl9YXXoJX5KLiKyVEERTfNEaBnxnd6M',
      user_token: 'xxxxxxxxxxxx',
      user_secret: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
  };
  var LocationDetail = {
      EXACT: 0,
      POSTAL: 1,
      NEIGHBORHOOD: 2,
      CITY: 3,
      REGION: 4,
      STATE: 5,
      COUNTRY: 6
  };
  var index_offset;
  return ({
    callback: function(json) {},
    run: function() {}
  });
};
Geomarklet.run();
```

The Location Hierarchy

A successful Fire Eagle query returns an object called the 'location hierarchy'. Depending on the level of detail shared, the index of a particular piece of information in the array will vary. The `LocationDetail` object maps the array indices of each level in the hierarchy to something comprehensible, whilst the `index_offset` variable is an adjustment based on the detail of the result returned.

The location hierarchy object looks like this, providing a granular breakdown of a location, in human consumable and machine-friendly forms.

```
"user": {
    "location_hierarchy": [{
      "level": 0,
      "level_name": "exact",
      "name": "707 19th St, San Francisco, CA",
      "normal_name": "94123",
      "geometry": {
          "type": "Point",
          "coordinates": [ - 0.2347530752, 67.232323]
      },
      "label": null,
      "best_guess": true,
      "id": ,
      "located_at": "2008-12-18T00:49:58-08:00",
      "query": "q=707%2019th%20Street,%20Sf"
  },
  {
      "level": 1,
      "level_name": "postal",
      "name": "San Francisco, CA 94114",
      "normal_name": "12345",
      "woeid": ,
      "place_id": "",
      "geometry": {
          "type": "Polygon",
          "coordinates": [],
          "bbox": []
      },
      "label": null,
      "best_guess": false,
      "id": 59358791,
      "located_at": "2008-12-18T00:49:58-08:00"
  },
  {
      "level": 2,
      "level_name": "neighborhood",
      "name": "The Mission, San Francisco, CA",
      "normal_name": "The Mission",
      "woeid": 23512048,
      "place_id": "Y12JWsKbApmnSQpbQg",
      "geometry": {
          "type": "Polygon",
          "coordinates": [],
          "bbox": []
      },
      "label": null,
      "best_guess": false,
      "id": 59358801,
      "located_at": "2008-12-18T00:49:58-08:00"
    },
  }
```

In this case the first object has a `level` of `0`, so the `index_offset` is also `0`.

Prerequisites

To query Fire Eagle we call in some existing libraries to handle the OAuth layer and the Fire Eagle API call. Your bookmarklet will need to add the following scripts into the page:

- The SHA1 encryption algorithm
- The OAuth wrapper
- An extension for the OAuth wrapper
- The Fire Eagle wrapper itself

When the bookmarklet is first run, we'll insert these scripts into the document. We're also inserting a stylesheet to dress up the UI that will be generated.

If you want to follow along any of the more mundane parts of the bookmarklet, you can download the full source code.

Rendering

This bookmarklet can be extended to support any formatting of your location you like, but for sake of example I'm going to build three common formatters that you'll find useful for common location scenarios: Sites which already ask for your location; and in publishing systems that accept tags or HTML mark-up.

All the rendering functions are items in a `renderers` object, so they can be iterated through easily, making it trivial to add new formatting functions as your find new use cases (just add another function to the object).

```
var renderers = {
geotag: function(user) {
  if(LocationDetail.EXACT !== index_offset) {
      return false;
  }
  else {
    var coords =

user.location_hierarchy[LocationDetail.EXACT].geometry.coordinates;
    return "geo:lat=" + coords[0] + ", geo:lon=" +
coords[1];
  }
},
city: function(user) {
  if(LocationDetail.CITY < index_offset) {
    return false;
  }
  else {
    return user.location_hierarchy[LocationDetail.CITY -
index_offset].name;
  }
}
```

You should always fail gracefully, and in line with catering to users who choose not to share their location precisely, always check that the location has been returned at the level you require. Geotags are expected to be precise, so if an exact location is unavailable, returning `false` will tell the rendering aspect of the bookmarklet to ignore the function altogether.

These first two are quite simple, `geotag` returns `geo:lat=-0.2347530752, geo:lon=67.232323` and `city` returns `San Francisco, CA`.

This final renderer creates a chunk of HTML using the adr and geo microformats, using all available aspects of the location hierarchy, and can be used to geotag any content you write on your blog or in comments:

```
html: function(user) {
  var geostring = '';
  var adrstring = '';
  var adr = [];
  adr.push('<p class="adr">');
  // city
  if(LocationDetail.CITY >= index_offset) {
    adr.push(
      '\n     <span class="locality">'
    +
user.location_hierarchy[LocationDetail.CITY-index_offset].normal_name
    + '</span>,'
    );
  }
  // county
  if(LocationDetail.REGION >= index_offset) {
    adr.push(
      '\n     <span class="region">'
    +
user.location_hierarchy[LocationDetail.REGION-index_offset].normal_nam
    + '</span>,'
      );
  }
  // locality
  if(LocationDetail.STATE >= index_offset) {
    adr.push(
      '\n     <span class="region">'
    +
user.location_hierarchy[LocationDetail.STATE-index_offset].normal_name
    + '</span>,'
    );
  }
  // country
  if(LocationDetail.COUNTRY >= index_offset) {
    adr.push(
      '\n     <span class="country-name">'
    +
user.location_hierarchy[LocationDetail.COUNTRY-index_offset].normal_na
    + '</span>'
    );
  }
  // postal
  if(LocationDetail.POSTAL >= index_offset) {
    adr.push(
      '\n     <span class="postal-code">'
    +
user.location_hierarchy[LocationDetail.POSTAL-index_offset].normal_nam
    + '</span>,'
    );
  }
  adr.push('\n</p>\n');
  adrstring = adr.join('');
  if(LocationDetail.EXACT === index_offset) {
    var coords =

user.location_hierarchy[LocationDetail.EXACT].geometry.coordinates;
    geostring = '<p class="geo">'
      +'\n    <span class="latitude">'
      + coords[0]
      + '</span>;'
      + '\n     <span class="longitude">'
      + coords[1]
      + '</span>\n</p>\n';
  }
  return (adrstring + geostring);
}
```

Here we check the availability of every level of location and build it into the adr and geo patterns as appropriate. Just as for the geotag function, if there's no exact location the geo markup won't be returned.

Finally, there's a rendering method which creates a container for all this data, renders all the applicable location formats and then displays them in the page for a user to copy and paste. You can throw this together with DOM methods and some simple styling, or roll in some components from YUI or JQuery to handle drawing full featured overlays.

You can see this simple implementation for rendering in the full source code.

Make the call

With a framework in place to render Fire Eagle's location hierarchy, the only thing that remains is to actually request your location. Having already authed through Amadeus earlier, that's as simple as instantiating the Fire Eagle JavaScript wrapper and making a single function call. It's a big deal that whilst a lot of new technologies like OAuth add some complexity and require new knowledge to work with, APIs like Fire Eagle are really very simple indeed.

```
return {
  run: function() {
    insert_prerequisites();
    setTimeout(
      function() {
        var fe = new FireEagle(
          Keys.consumer_key,
          Keys.consumer_secret,
          Keys.user_token,
          Keys.user_secret
        );
        var script = document.createElement('script');
        script.type = 'text/javascript';
        script.src = fe.getUserUrl(
          FireEagle.RESPONSE_FORMAT.json,
          'Geomarklet.callback'
        );
        document.body.appendChild(script);
      },
      2000
    );
  },
  callback: function(json) {
    if(json.rsp && 'fail' == json.rsp.stat) {
      alert('Error ' + json.rsp.code + ": " +
json.rsp.message);
    }
    else {
      index_offset =
json.user.location_hierarchy[0].level;
      draw_selector(json);
    }
  }
};
```

We first insert the prerequisite scripts required for the Fire Eagle request to function, and to prevent trying to instantiate the `FireEagle` object before it's been loaded over the wire, the remaining instantiation and request is wrapped inside a `setTimeout` delay.

We then create the request URL, referencing the `Geomarklet.callback` callback function and then append the script to the document body — allowing a cross-domain request.

The callback itself is quite simple. Check for the presence and value of `rsp.status` to test for errors, and display them as required. If the request is successful set the `index_offset` — to adjust for the granularity of the location hierarchy — and then pass the object to the renderer.

The result? When `Geomarklet.run()` is called, your location from Fire Eagle is read, and each `renderer` displayed on the page in an easily copy and pasteable form, ready to be used however you need.

DEPLOY

The final step is to convert this code into a long string for use as a bookmarklet. Easiest for Mac users is the JavaScript bundle in TextMate — choose Bundles: JavaScript: Copy as Bookmarklet to Clipboard. Then create a new 'Get Location' bookmark in your browser of choice and paste in.

Those without TextMate can shrink their code down into a single line by first running their code through the JSLint tool (to ensure the code is free from errors and has all the required semi-colons) and then use a find-and-replace tool to remove line breaks from your code (or even run your code through JSMin to shrink it down).

With the bookmarklet created and added to your bookmarks bar, you can now call up your location on any page at all. Get a feel for a web where your location is just another reliable part of the browsing experience.

WHERE NEXT?

So, the Geomarklet you've been guided through is a pretty simple premise and pretty simple output. But from this base you can start to extend: Add code that will insert each of the location renderings directly into form fields, perhaps, or how about site-specific handlers to add your location tags into the correct form field in Wordpress or Tumblr? Paste in your current location to Google Maps? Or Flickr?

Geomarklet gives you a base to start experimenting with location on your own pages and the sites you browse daily.

The introduction of consumer accessible geo to the web is an adventure of discovery; not so much discovering new locations, but discovering location itself.

ABOUT THE AUTHOR

Ben Ward is a Front End Engineer at YDN — the Yahoo Developer Network. Until recently, he had worked in Yahoo's Brickhouse group, where he wrangled HTML, CSS and JavaScript for fun things like Fire Eagle.

By night he put his efforts into the microformats.org community, working as an admininstrator and getting pedantic about semantics. He regularly speaks at conferences on microformats, is a credited specification author and active editor.

When not trying to make the internet better he's slowly but surely settling into his new life in San Francisco, embracing an eclectic cultural combination of imported cheese, fine wines and Rock Band 2.

His blog is at http://ben-ward.co.uk.

Photo: Nathan Ward

22. Absolute Columns

Dan Rubin 24ways.org/200822

CSS layouts have come quite a long way since the dark ages of web publishing, with all sorts of creative applications of floats, negative margins, and even background images employed in order to give us that most basic building block, the column. As the title implies, we are indeed going to be discussing columns today—more to the point, a handy little application of absolute positioning that may be exactly what you've been looking for…

CARE FOR A NIGHTCAP?

If you've been developing for the web for long enough, you may be familiar with this little children's fable, passed down from wizened Shaolin monks sitting atop the great Mt. Geocities: "Once upon a time, multiple columns of the same height could be easily created using TABLES." Now, though we're all comfortably seated on the standards train (and let's be honest: even if you like to think you've fallen off, if you've given up using tables for layout, rest assured your sleeper car is still reserved), this particular—and as page layout goes, quite basic—trick is still a thorn in our CSSides compared to the ease of achieving the same effect using said Tables of Evil™.

SEE, THE ORANGE JUICE MASKS THE FLAVOR…

Creative solutions such as Dan Cederholm's Faux Columns do a good job of making it appear as though adjacent columns maintain equal height as content expands, using a background image to fill the space that the columns cannot.

Now, the Holy Grail of CSS columns behaving exactly how they would as table cells—or more to the point, as columns—still eludes us (cough CSS3 Multi-column layout module cough), but sometimes you just need, for example, a secondary column (say, a sidebar) to match the height of a primary column, without involving the creation of images. This is where a little absolute positioning can save you time, while possibly giving your layout a little more flexibility.

SHAKEN, NOT STIRRED

You're probably familiar by now with the concept of Making the Absolute, Relative as set forth long ago by Doug Bowman, but let's quickly review just in case: an element set to `position:absolute` will position itself relative to its nearest ancestor set to `position:relative`, rather than the browser window (see Figure 1).

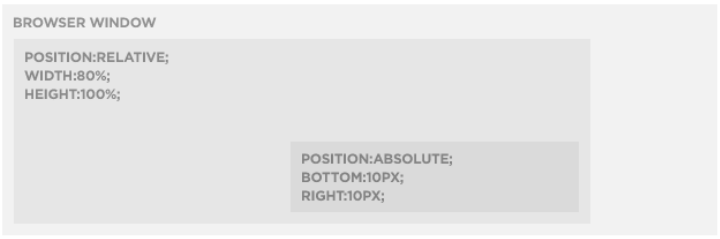


Figure 1.

However, what you may not know is that we can anchor more than two sides of an absolutely positioned element. Yes, that's right, all four sides (top, right, bottom, left) can be set, though in this example we're only going to require the services of three sides (see Figure 2 for the end result).

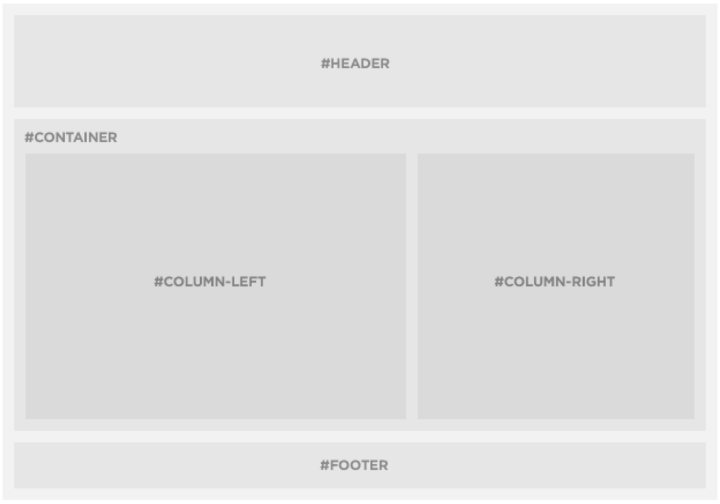


Figure 2.

TRUST ME, THIS WILL MAKE YOU FEEL BETTER

Our requirements are essentially the same as the standard “absolute-relative” trick—a container `<div>` set to `position:relative`, and our sidebar `<div>` set to `position:absolute` — plus another `<div>` that will serve as our main content column. We’ll also add a few other common layout elements (wrapper, header, and footer) so our example markup looks more like a real layout and less like a test case:

```
<div id="wrapper">
  <div id="header">
    <h2>#header</h2>
  </div>
  <div id="container">
    <div id="column-left">
      <h2>#left</h2>
      <p>Lorem ipsum dolor sit amet…</p>
    </div>
    <div id="column-right">
      <h2>#right</h2>
    </div>
  </div>
  <div id="footer">
    <h2>#footer</h2>
  </div>
</div>
```

In this example, our main column (`#column-left`) is only being given a width to fit within the context of the layout, and is otherwise untouched (though we're using pixels here, this trick will of course work with fluid layouts as well), and our right keeping our styles nice and minimal:

```
#container {
  position: relative;
}
#column-left {
  width: 480px;
}
#column-right {
  position: absolute;
  top: 10px;
  right: 10px;
  bottom: 10px;
  width: 250px;
}
```

The trick is a simple one: the `#container <div>` will expand vertically to fit the content within `#column-left`. By telling our sidebar `<div>` (`#column-right`) to attach itself not only to the top and right edges of `#container`, but also to the bottom, it too will expand and contract to match the height of the left column (duplicate the "lorem ipsum" paragraph a few times to see it in action).

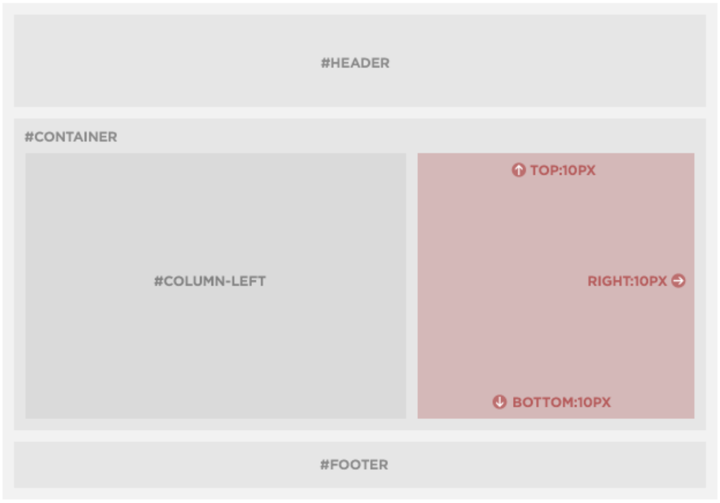


Figure 3.

ON THE ROCKS

"But wait!" I hear you exclaim, "when the right column has more content than the left column, it doesn't expand! My text runneth over!" Sure enough, that's exactly what happens, and what's more, it's supposed to: Absolutely positioned elements do exactly what you tell them to do, and unfortunately aren't very good at thinking outside the box (get it? sigh…).

However, this needn't get your spirits down, because there's an easy way to address the issue: by adding `overflow:auto` to `#column-right`, a scrollbar will automatically appear if and when needed:

```
#column-right {
  position: absolute;
  top: 10px;
  right: 10px;
  bottom: 10px;
  width: 250px;
  overflow: auto;
}
```

While this may limit the trick's usefulness to situations where the primary column will almost always have more content than the secondary column—or where the secondary column's content can scroll with wild abandon—a little prior planning will make it easy to incorporate into your designs.

DRIVING US TO DRINK

It just wouldn't be right to have a friendly, festive holiday tutorial without inviting IE6, though in this particular instance there will be no shaming that old browser into admitting it has a problem, nor an intervention and subsequent 12-step program. That's right my friends, this tutorial has abstained from IE6-abuse now for 30 days, thanks to the wizard Dean Edwards and his amazingly talented IE7 Javascript library.

Simply drop the Conditional Comment and `<script>` element into the `<head>` of your document, along with one tiny CSS hack that only IE6 (and below) will ever see, and that browser will be back on the straight and narrow:

```
<!--[if lt IE 7]>
<script src="http://ie7-js.googlecode.com/svn/version/
2.0(beta3)/IE7.js" type="text/javascript"></script>
<style type="text/css" media="screen">
  #container {
    zoom:1; /* helps fix IE6 by initiating hasLayout */
  }
</style>
<![endif]-->
```

EGGNOG IS SUPPOSED TO BE SPIKED, RIGHT?

Of course, this is one simple example of what can be a much more powerful technique, depending on your needs and creativity. Just don't go coding up your wildest fantasies until you've had a chance to sleep off the Christmas turkey and whatever tasty liquids you happen to imbibe along the way…

ABOUT THE AUTHOR

Dan Rubin is a highly accomplished user interface designer and usability consultant, with over ten years of experience as a leader in the fields of web standards and usability, specifically focusing on the use of (X)HTML and CSS to streamline development and improve accessibility.

His passion for all things creative and artistic isn't a solely selfish endeavor either—you'll frequently find him waxing educational about a cappella jazz and barbershop harmony, interface design, usability, web standards, typography, and graphic design in general.

In addition to his contributions to sites including Blogger, the CSS Zen Garden, Yahoo! Small Business and Microsoft's ASP.net portal, Dan is a contributing author of Cascading Style Sheets: Separating Content from Presentation (2nd Edition, friends of ED, 2003), technical reviewer for Beginning CSS Web Development (Apress, 2006), The Art & Science of CSS (SitePoint, 2007) and Sexy Web Design (SitePoint, 2009), coauthor of Pro CSS Techniques (Apress, 2006), and Web Standards Creativity (friends of ED, 2007), writes about web standards, design and life in general on his blog, SuperfluousBanter.org, and spends his professional time on a variety of online and offline projects for Sidebar Creative, Webgraph and Black Seagull, and consults on design, user interaction and online publishing for Garcia Media.

23. Contract Killer

Andrew Clarke 24ways.org/200823

When times get tough, it can often feel like there are no good people left in the world, only people who haven't yet turned bad. These bad people will go back on their word, welch on a deal, put themselves first. You owe it to yourself to stay on top. You owe it to yourself to ensure that no matter how bad things get, you'll come away clean. You owe it yourself and your business not to be the guy lying bleeding in an alley with a slug in your gut.

But you're a professional, right? Nothing bad is going to happen to you.

You're a good guy. You do good work for good people.

Think again chump.

Maybe you're a gun for hire, a one man army with your back to the wall and nothing standing between you and the line at a soup kitchen but your wits. Maybe you work for the agency, or like me you run one of your own. Either way, when times get tough and people get nasty, you'll need more than a killer smile to save you. You'll need a killer contract too.

It was exactly ten years ago today that I first opened my doors for business. In that time I've thumbed through enough contracts to fill a filing cabinet. I've signed more contracts than I can remember, many so complicated that I *should* have hired a lawyer (or detective) to make sense of their complicated jargon and solve their cross-reference puzzles. These documents had not been written to be understood on first reading but to spin me around enough times so as to give the other player the upper-hand.

If signing a contract I didn't fully understand made me a stupid son-of-a-bitch, not asking my customers to sign one just makes me plain dumb. I've not always been so careful about asking my customers to sign contracts with me as I am now. Somehow in the past I felt that insisting on a contract went against the friendly, trusting relationship that I like to build with my customers. Most of the time the game went my way. On rare the occasions when a fight broke out, I ended up bruised and bloodied. I learned that asking my customers to sign a contract matters to both sides, but what also matters to me is that these contracts should be more meaningful, understandable and less complicated than any of those that I have ever autographed.

WRITING A KILLER CONTRACT

If you are writing a contract between you and your customers it doesn’t have to conform to the seemingly standard format of jargon and complicated legalese. You can be creative. A killer contract will clarify what is expected of both sides and it can also help you to communicate your approach to doing business. It will back-up your brand values and help you to build a great relationship between you and your customers. In other words, a creative contract can be a killer contract.

Your killer contract should cover:

- A simple overview of who is hiring who, what they are being hired to do, when and for how much
- What both parties agree to do and what their respective responsibilities are
- The specifics of the deal and what is or isn’t included in the scope
- What happens when people change their minds (as they almost always do)
- A simple overview of liabilities and other legal matters
- You might even include a few jokes

To help you along, I will illustrate those bullet points by pointing both barrels at the contract that I wrote and have been using at Stiffs & Nonsense for the past year. My contract has been worth its weight in lead and you are welcome to take all or any part of it to use for yourself. It's packing a creative-commons attribution share-a-like license. That means you are free to re-distribute it, translate it and otherwise re-use it in ways I never considered. In return I only ask you mention my name and link back to this article. As I am only an amateur detective, you should have it examined thoroughly by your own, trusted legal people if you use it.

NB: The specific details of this killer contract work for me and my customers. That doesn't mean that they will work for you and yours. The ways that I handle design revisions, testing, copyright ownership and other specifics are not the main focus of this article. That you handle each of them carefully when you write your own killer contract is.

KISS ME, DEADLY

Setting a tone and laying foundations for agreement

The first few paragraphs of a killer contract are the most important. Just like a well thought-out web page, these first few words should be simple, concise and include the key points in your contract. As this is the part of the contract that people absorb most easily, it is important that you make it count. Start by setting the overall tone and explaining how your killer contract is structured and why it is different.

> We will always do our best to fulfill your needs and meet your goals, but sometimes it is best to have a few simple things written down so that we both know what is what, who should do what and what happens if stuff goes wrong. In this contract you won't find complicated legal terms or large passages of unreadable text. We have no desire to trick you into signing something that you might later regret. We do want what's best for the safety of both parties, now and in the future.
>
> In short
>
> You (**[customer name]**) are hiring us (**[company name]**) located at **[address]** to **[design and develop a web site]** for the estimated total price of **[total]** as outlined in our previous correspondence. Of course it's a little more complicated, but we'll get to that.

THE BIG KILL

What both parties agree to do

Have you ever done work on a project in good faith for a junior member of a customer's team, only to find out later that their spending hadn't been authorized? To make damn sure that does not happen to you, you should ask your customer to confirm that not only are they authorized to enter into your contract but that they will fulfill all of their obligations to help you meet yours. This will help you to avoid any gunfire if, as deadline day approaches, you have fulfilled your side of the bargain but your customer has not come up with the goods.

> As our customer, you have the power and ability to enter into this contract on behalf of your company or organization. You agree to provide us with everything that we need to complete the project including text, images and other information as and when we need it, and in the format that we ask for. You agree to review our work, provide feedback and sign-off approval in a timely manner too. Deadlines work two ways and you will also be bound by any dates that we set together. You also agree to stick to the payment schedule set out at the end of this contract.
>
> We have the experience and ability to perform the services you need from us and we will carry them out in a professional and timely manner. Along the way we will endeavor to meet all the deadlines set but we can't be responsible for a missed launch date or a deadline if you have been late in supplying materials or have not approved or signed off our work on-time at any stage. On top of this we will also maintain the confidentiality of any information that you give us.

MY GUN IS QUICK

Getting down to the nitty gritty

What appear at first to be a straight-forward projects can sometimes turn long and complicated and unless you play it straight from the beginning your relationship with your customer can suffer under the strain. Customers do, and should have the opportunity to, change their minds and give you new assignments. After-all, projects should be flexible and few customers know from the get-go exactly what they want to see. If you handle this well from the beginning you will help to keep yourself and your customers from becoming frustrated. You will also help yourself to dodge bullets in the event of a fire-fight.

> We will create designs for the look-and-feel, layout and functionality of your web site. This contract includes one main design plus the opportunity for you to make up to two rounds of revisions. If you're not happy with the designs at this stage, you will pay us in full for all of the work that we have produced until that point and you may either cancel this contract or continue to commission us to make further design revisions at the daily rate set out in our original estimate.
>
> We know from plenty of experience that fixed-price contracts are rarely beneficial to you, as they often limit you to your first idea about how something should look, or how it might work. We don't want to limit either your options or your opportunities to change your mind.
>
> The estimate/quotation prices at the beginning of this document are based on the number of days that we estimate we'll need to accomplish everything that you have told us you want to achieve. If you do want to change your mind, add extra pages or templates or even add new functionality, that won't be a problem. You will be charged the daily rate set out in the estimate

> we gave you. Along the way we might ask you to put requests in writing so we can keep track of changes.

As I like to push my luck when it comes to CSS, it never hurts to head off the potential issue of progressive enrichment right from the start. You should do this too. But don’t forget that when it comes to technical matters your customers may have different expectations or understanding, so be clear about what you will and won’t do.

> If the project includes XHTML or HTML markup and CSS templates, we will develop these using valid XHTML 1.0 Strict markup and CSS2.1 + 3 for styling. We will test all our markup and CSS in current versions of all major browsers including those made by Apple, Microsoft, Mozilla and Opera. We will also test to ensure that pages will display visually in a 'similar', albeit not necessarily an identical way, in Microsoft Internet Explorer 6 for Windows as this browser is now past it's sell-by date.
>
> We will not test these templates in old or abandoned browsers, for example Microsoft Internet Explorer 5 or 5.5 for Windows or Mac, previous versions of Apple's Safari, Mozilla Firefox or Opera unless otherwise specified. If you need to show the same or similar visual design to visitors using these older browsers, we will charge you at the daily rate set out in our original estimate for any necessary additional code and its testing.

THE TWISTED THING

It is not unheard of for customers to pass off stolen goods as their own. If this happens, make sure that you are not the one left holding the baby. You should also make it clear who owns the work that you make as customers often believe that because they pay for your time, that they own everything that you produce.

Copyrights

> You guarantee to us that any elements of text, graphics, photos, designs, trademarks, or other artwork that you provide us for inclusion in the web site are either owned by your good selfs, or that you have permission to use them. When we receive your final payment, copyright is automatically assigned as follows:
>
> You own the graphics and other visual elements that we create for you for this project. We will give you a copy of all files and you should store them really safely as we are not required to keep them or provide any native source files that we used in making them.
>
> You also own text content, photographs and other data you provided, unless someone else owns them. We own the XHTML markup, CSS and other code and we license it to you for use on only this project.

VENGEANCE IS MINE!

The fine print

Unless your work is pro-bono, you should make sure that your customers keep you in shoe leather. It is important that your customers know from the outset that they must pay you on time if they want to stay on good terms.

> We are sure you understand how important it is as a small business that you pay the invoices that we send you promptly. As we're also sure you'll want to stay friends, you agree to stick tight to the following payment schedule.
>
> [Payment schedule]

No killer contract would be complete without you making sure that you are watching your own back. Before you ask your customers to sign, make it clear-cut what your obligations are and what will happen if any part of your killer contract finds itself laying face down in the dirt.

> We can't guarantee that the functions contained in any web page templates or in a completed web site will always be error-free and so we can't be liable to you or any third party for damages, including lost profits, lost savings or other incidental, consequential or special damages arising out of the operation of or inability to operate this web site and any other web pages, even if you have advised us of the possibilities of such damages.
>
> Just like a parking ticket, you cannot transfer this contract to anyone else without our permission. This contract stays in place and need not be renewed. If any provision of this agreement shall be unlawful, void, or for any reason unenforceable, then that provision shall be deemed severable from this agreement and shall not affect the validity and enforceability of any remaining provisions.
>
> Phew.
>
> Although the language is simple, the intentions are serious and this contract is a legal document under exclusive jurisdiction of **[English]** courts. Oh and don't forget those men with big dogs.

SURVIVAL… ZERO!

Take it from me, packing a killer contract will help to keep you safe when times get tough, but you must still keep your wits about you and stay on the right side of the law.

Don't be a turkey this Christmas.

Be a contract killer.

Update, May 2010: For a follow-on to this article see Contract Killer: The Next Hit

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. Recession Tips For Web Designers

Jeffrey Zeldman 24ways.org/200824

For web designers, there are four keys to surviving bad economic times: do good work, charge a fair price, lower your overhead, and be sure you are communicating with your client. As a reader of 24 ways, you already do good work, so let's focus on the rest.

I know something about surviving bad times, having started my agency, Happy Cog, at the dawn of the dot-com bust. Of course, the recession we're in now may end up making the dot-com bust look like the years of bling and gravy. But the bust was rough enough at the time.

Bad times are hard on overweight companies and over-leveraged start-ups, but can be kind to freelancers and small agencies. Clients who once had money to burn and big agencies to help them burn it suddenly consider the quality of work more important than the marquee value of the business card. Fancy offices and ten people at every meeting are out. A close relationship with an individual or small team that listens is in.

THIN IS IN

If you were good in client meetings when you were an employee, print business cards and pick a name for your new agency. Once some cash rolls in, see an accountant.

If the one-person entrepreneur model isn't you, it's no problem. Form a virtual agency with colleagues who complement your creative, technical, and business skills. Athletics is a Brooklyn-based multi-disciplinary "art and design collective." Talk about low overhead: they don't have a president, a payroll, or a pension plan. But that hasn't stopped clients like adidas, Nike, MTV, HBO, Disney, DKNY, and Sundance Channel from knocking on their (virtual) doors.

Running a traditional business is like securing a political position in Chicago: it costs a fortune. That's why bad times crush so many companies. But you are a creature of the internets. You don't need an office to do great work. I ran Happy Cog out of my apartment for far longer than anyone realized. My clients, when they learned my secret, didn't care.

Keep it lean: if you can budget your incoming freelance money, you don't have to pay yourself a traditional salary. Removing the overhead associated with payroll means more of the budget stays in your pocket, enabling you to price your projects competitively, while still within industry norms. (Underpricing is uncool, and clients who knowingly choose below-market-rate vendors tend not to treat those vendors with respect.)

GETTING GIGS

Web design is a people business. If things are slow, email former clients. If you just lost your job, email former agency clients with whom you worked closely to inform them of your freelance business and find out how they're doing. Best practice: focus the email on wishing them a happy holiday and asking how they're doing. Let your email signature file tell them you're now the president of Your Name Design. Leading with the fact that you just lost your job may earn sympathy (or commiseration: the client may have lost her job, too) but it's not exactly a sure-fire project getter.

The qualities that help you land a web design project are the same in good times or bad. Have a story to tell about the kind of services you offer, and the business benefits they provide. (If you design with web standards, you already have one great story line. What are the others?)

Don’t be shy about sharing your story, but don’t make it the focus of the meeting. The client is the focus. Before you meet her, learn as much as you can about her users, her business, and her competitors. At the very least, read her site’s About pages, and spend some quality time with Google.

Most importantly, go to the meeting knowing how much you *don’t* know. Arrive curious, and armed with questions. Maintain eye contact and keep your ears open. If a point you raise causes two people to nod at each other, follow up on that point, don’t just keep grinding through your Keynote presentation.

If you pay attention and think on your feet, it tells the potential client that they can expect you to listen and be flexible. (Clients are like unhappy spouses: they’re dying for someone to finally listen.) If you stick to a prepared presentation, it might send the message that you are inflexible or nervous or both. “Nervous” is an especially bad signal to send. It indicates that you are either dishonest or inexperienced. Neither quality invites a client to sign on. Web design is a people business for the client, too: they should feel that their interactions with you will be pleasant and illuminating. And that you’ll listen. Did I mention that?

GIVE IT TIME

Securing clients takes longer and requires more effort in a recession. If two emails used to land you a gig, it will now take four, plus an in-person meeting, plus a couple of follow-up calls. This level of salesmanship is painful to geeks and designers, who would rather spend four hours kerning type or debugging a style sheet than five minutes talking business on the telephone. I know. I'm the same way. But we must overcome our natural shyness and inwardness if we intend not to fish our next meal out of a neighbor's garbage can.

As a bonus, once the recession ends, your hard-won account management skills will help you take your business to the next level. By the time jobs are plentiful again, you may not want to work for anyone but yourself. You'll be a captain of our industry. And talented people will be emailing to ask you for a job.

ABOUT THE AUTHOR

Jeffrey Zeldman is the founder and executive creative director of Happy Cog™, an agency of web design specialists, and the co-founder (with Eric Meyer) of An Event Apart.

In 1995, the former art director and copywriter launched one of the first personal sites (Jeffrey Zeldman Presents) and began publishing web design tutorials. In 1998 he co-founded (and for several years led) The Web Standards Project, a grassroots coalition that brought standards to our browsers. That same year, he launched *A List Apart* "for people who make websites."

Jeffrey has written many articles and two books, notably the foundational web standards text *Designing With Web Standards*, now in its third edition.

Photo: John Morrison