



24

2009

24 WAYS

# Credits

---

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

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

# 2009

---

A year when books were winning (Five Simple Steps published *A Practical Guide to Designing for the Web* by Mark Boulton and *Designing with Web Standards* by Jeffrey Zeldman and Ethan Marcotte reached its third edition) and the web was losing (Yahoo! closed Geocities). Significant progress was made with web fonts and HTML5, and 24 ways delivered the Christmas gifts again.

Working With RGBA Colour.....	5
Breaking Out The Edges of The Browser .....	15
Have a Field Day with HTML5 Forms .....	30
What makes a website successful? It might not be what you expect!.....	46
HTML5: Tool of Satan, or Yule of Santa? .....	53
Front-End Code Reusability with CSS and JavaScript.....	60
Type-Inspired Interfaces .....	73

The Construction of Instruction.....	82
Don't Lose Your :focus.....	92
A New Year's Resolution .....	100
Incite A Riot .....	109
Self-Testing Pages with JavaScript.....	116
Rock Solid HTML Emails.....	128
Going Nuts with CSS Transitions .....	144
CSS Animations.....	155
Designing For The Switch .....	163
The Web Is Your CMS.....	173
A Pet Project is For Life, Not Just for Christmas .....	183
Spruce It Up.....	195
Cleaner Code with CSS3 Selectors .....	201
Make Out Like a Bandit.....	218
Real Fonts and Rendering: The New Elephant in the Room .....	228
Ignorance Is Bliss.....	235
Make Your Mockup in Markup.....	242

# 1. Working With RGBA Colour

---

Drew McLellan

[24ways.org/200901](http://24ways.org/200901)

When Tim and I were discussing the redesign of this site last year, one of the clear goals was to have a graphical style without making the pages heavy with a lot of images. When we launched, a lot of people were surprised that the design wasn't built with PNGs. Instead we'd used RGBA colour values, which is part of the CSS3 specification.

## WHAT IS RGBA COLOUR?

We're all familiar with specifying colours in CSS using by defining the mix of red, green and blue light required to achieve our tone. This is fine and dandy, but whatever values we specify have one thing in common — the colours are all solid, flat, and well, a bit *boring*.



## Flat RGB colours

---

CSS3 introduces a couple of new ways to specify colours, and one of those is RGBA. The A stands for *Alpha*, which refers to the level of opacity of the colour, or to put it another way, the amount of transparency. This means that we can set not only the red, green and blue values, but also control how much of what's behind the colour shows through. Like with layers in Photoshop.

### **DON'T WE HAVE OPACITY ALREADY?**

The ability to set the opacity on a colour differs subtly from setting the opacity on an element using the CSS `opacity` property. Let's look at an example.

Here we have an H1 with foreground and background colours set against a page with a patterned background.



Heading with no transparency applied

---

```
h1 {  
  color: rgb(0, 0, 0);  
  background-color: rgb(255, 255, 255);  
}
```

By setting the CSS `opacity` property, we can adjust the transparency of the entire element and its contents:



Heading with 50% opacity on the element

---

```
h1 {  
  color: rgb(0, 0, 0);  
  background-color: rgb(255, 255, 255);  
  opacity: 0.5;  
}
```

RGBA colour gives us something different – the ability to control the opacity of the individual colours rather than the entire element. So we can set the opacity on just the background:



50% opacity on just the background colour

---

```
h1 {
  color: rgb(0, 0, 0);
  background-color: rgba(255, 255, 255, 0.5);
}
```

Or leave the background solid and change the opacity on just the text:



50% opacity on just the foreground colour

---

```
h1 {
  color: rgba(0, 0, 0, 0.5);
  background-color: rgb(255, 255, 255);
}
```

## THE HOW-TO

You'll notice that above I've been using the `rgb()` syntax for specifying colours. This is a bit less common than the usual hex codes (like `#FFF`) but it makes sense when starting to use RGBA. As there's no way to specify opacity with hex codes, we use `rgba()` like so:

```
color: rgba(255, 255, 255, 0.5);
```



Just like `rgb()` the first three values are red, green and blue. You can specify these 0-255 or 0%-100%. The fourth value is the opacity level from 0 (completely transparent) to 1 (completely opaque).

You can use this anywhere you'd normally set a colour in CSS — so it's good for foregrounds and background, borders, outlines and so on. All the transparency effects on this site's current design are achieved this way.

## **SUPPORTING ALL BROWSERS**

Like a lot of the features we'll be looking at in this year's 24 ways, RGBA colour is supported by a lot of the newest browsers, but not the rest. Firefox, Safari, Chrome and Opera browsers all support RGBA, but Internet Explorer does not.

Fortunately, due to the robust design of CSS as a language, we can specify RGBA colours for browsers that support it *and* an alternative for browsers that do not.

### **Falling back to solid colour**

The simplest technique is to allow the browser to fall back to using a solid colour when opacity isn't available. The CSS parsing rules specify that any unrecognised value

should be ignored. We can make use of this because a browser without RGBA support will treat a colour value specified with `rgba()` as unrecognised and discard it.

So if we specify the colour first using `rgb()` for all browsers, we can then overwrite it with an `rgba()` colour for browsers that understand RGBA.

```
h1 {  
  color: rgb(127, 127, 127);  
  color: rgba(0, 0, 0, 0.5);  
}
```

## Falling back to a PNG

In cases where you're using transparency on a `background-color` (although not on borders or text) it's possible to fall back to using a PNG with alpha channel to get the same effect. This is less flexible than using CSS as you'll need to create a new PNG for each level of transparency required, but it can be a useful solution.

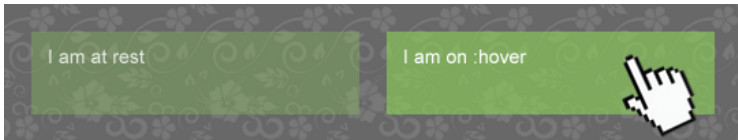
Using the same principal as before, we can specify the background in a style that all browsers will understand, and then overwrite it in a way that browsers without RGBA support will ignore.

```
h1 {  
  background: transparent url(black50.png);  
  background: rgba(0, 0, 0, 0.5) none;  
}
```

It's important to note that this works because we're using the background shorthand property, enabling us to set both the background colour and background image in a single declaration. It's this that enables us to rely on the browser ignoring the second declaration when it encounters the unknown `rgba()` value.

### NEXT STEPS

The really great thing about RGBA colour is that it gives us the ability to create far more graphically rich designs without the need to use images. Not only does that make for faster and lighter pages, but sites which are easier and quicker to build and maintain. CSS values can also be changed in response to user interaction or even manipulated with JavaScript in a way that's just not so easy using images.



Opacity can be changed on `:hover` or manipulated with JavaScript

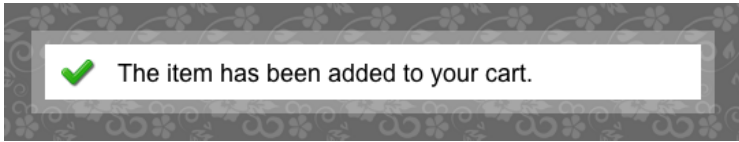
---

```
div {
  color: rgba(255, 255, 255, 0.8);
  background-color: rgba(142, 213, 87, 0.3);
}
div:hover {
```

---

```
color: rgba(255, 255, 255, 1);
background-color: rgba(142, 213, 87, 0.6);
}
```

Clever use of transparency in border colours can help ease the transition between overlay items and the page behind.



Borders can receive the RGBA treatment, too

---

```
div {
  color: rgb(0, 0, 0);
  background-color: rgb(255, 255, 255);
  border: 10px solid rgba(255, 255, 255, 0.3);
}
```

## IN CONCLUSION

That's a brief insight into RGBA colour, what it's good for and how it can be used whilst providing support for older browsers. With the current lack of support in Internet Explorer, it's probably not a technique that commercial designs will want to heavily rely on right away – simply because of the overhead of needing to think about fallback all the time.

It is, however, a useful tool to have for those smaller, less critical touches that can really help to finesse a design. As browser support becomes more mainstream, you'll already be familiar and practised with RGBA and ready to go.

### ABOUT THE AUTHOR



Drew McLellan is lead developer on your favourite small CMS, Perch. He is Director and Senior Developer at UK-based web development agency [edgeofmyseat.com](http://edgeofmyseat.com), and formerly Group

Lead at the Web Standards Project. When not publishing 24 ways, Drew keeps a **personal site** covering web development issues and themes, **takes photos** and **tweets a lot**.

## 2. Breaking Out The Edges of The Browser

---

Remy Sharp

24ways.org/200902

HTML5 contains more than just the new entities for a more meaningful document, it also contains an arsenal of JavaScript APIs. So many in fact, that some APIs have outgrown the HTML5 spec's backyard and have been sent away to grow up all on their own and been given the prestigious honour of being specs in their own right.

So when I refer to (bendy finger quote) "HTML5", I mean the HTML5 specification and a handful of other specifications that help us authors build web applications.

Examples of those specs I would include in the umbrella term would be: geolocation, web storage, web databases, web sockets and web workers, to name a few.

For all you guys and gals, on this special 2009 series of 24 ways, I'm just going to focus on data storage and offline applications: boldly taking your browser where no browser has gone before!

## WEB STORAGE

The Web Storage API is basically cookies on steroids, a unhealthy dosage of steroids. Cookies are always a pain to work with. First of all you have the problem of setting, changing and deleting them. Typically solved by Googling and blindly relying on PPK's solution. If that wasn't enough, there's the 4Kb limit that some of you have hit when you really don't want to.

The Web Storage API gets around all of the hoops you have to jump through with cookies. Storage supports around 5Mb of data per domain (the spec's recommendation, but it's open to the browsers to implement anything they like) and splits in to two types of storage objects:

1. `sessionStorage` - available to all pages on that domain while the window remains open
2. `localStorage` - available on the domain until manually removed



## Support

Ignoring beta browsers for our support list, below is a list of the major browsers and their support for the Web Storage API:

- Latest: Internet Explorer, Firefox, Safari (desktop & mobile/iPhone)
- Partial: Google Chrome (only supports `localStorage`)
- Not supported: Opera (as of 10.10)

## Usage

Both `sessionStorage` and `localStorage` support the same interface for accessing their contents, so for these examples I'll use `localStorage`.

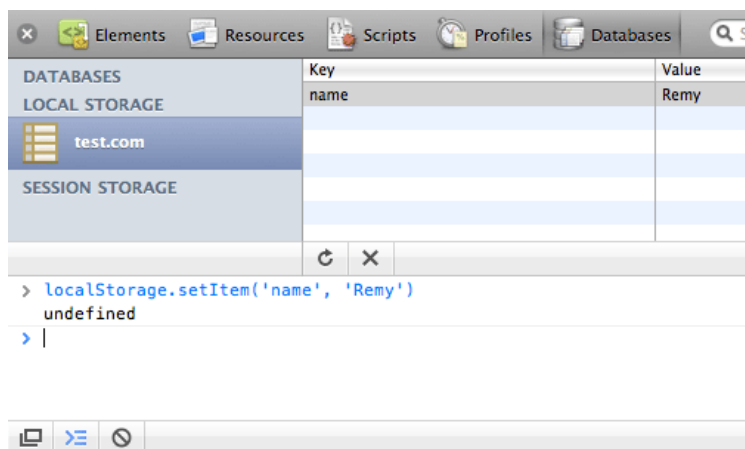
The storage interface includes the following methods:

- `setItem(key, value)`
- `getItem(key)`
- `key(index)`
- `removeItem(key)`
- `clear()`

In the simple example below, we'll use `setItem` and `getItem` to store and retrieve data:

```
localStorage.setItem('name', 'Remy');  
alert( localStorage.getItem('name') );
```

Using alert boxes can be a pretty lame way of debugging. Conveniently Safari (and Chrome) include database tab in their debugging tools (cmd+alt+i), so you can get a visual handle on the state of your data:



Viewing localStorage

---

As far as I know only Safari has this view on stored data natively in the browser. There *may* be a Firefox plugin (but I've not found it yet!) and IE... well that's just IE.

Even though we've used `setItem` and `getItem`, there's also a few other ways you can set and access the data.

In the example below, we're accessing the stored value directly using an `expando` and equally, you can also set values this way:

```
localStorage.name = "Remy";  
alert( localStorage.name ); // shows "Remy"
```

The Web Storage API also has a `key` method, which is zero based, and returns the key in which data has been stored. This should also be in the same order that you set the keys, for example:

```
alert( localStorage.getItem(localStorage.key(0)) );  
// shows "Remy"
```

I mention the `key()` method because it's not an unlikely name for a stored value. This can cause serious problems though.

When selecting the names for your keys, you need to be sure you don't take one of the method names that are already on the storage object, like `key`, `clear`, etc. As there are no warnings when you try to overwrite the methods, it means when you come to access the `key()` method, the call breaks as `key` is a string value and not a function.

You can try this yourself by creating a new stored value using `localStorage.key = "foo"` and you'll see that the Safari debugger breaks because it relies on the `key()` method to enumerate each of the stored values.

## Usage Notes

Currently all browsers only support storing strings. This also means if you store a numeric, it will get converted to a string:

```
localStorage.setItem('count', 31);
alert(typeof localStorage.getItem('count'));
// shows "string"
```

This also means you can't store more complicated objects natively with the storage objects. To get around this, you can use Douglas Crockford's JSON parser (though Firefox 3.5 has JSON parsing support baked in to the browser – yay!) `json2.js` to convert the object to a stringified JSON object:

```
var person = {
  name: 'Remy',
  height: 'short',
  location: 'Brighton, UK'
};
localStorage.setItem('person', JSON.stringify(person));
alert( JSON.parse(localStorage.getItem('person')).name
);
// shows "Remy"
```

## Alternatives

There are a few solutions out there that provide storage solutions that detect the Web Storage API, and if it's not available, fall back to different technologies (for instance,

using a flash object to store data). One comprehensive version of this is **Dojo's storage library**. I'm personally more of a fan of libraries that plug missing functionality under the same namespace, just as Crockford's JSON parser does (above).

For those interested it what that might look like, I've mocked together a simple implementation of `sessionStorage`. Note that it's incomplete (because it's missing the `key` method), and it could be refactored to not using the `JSON.stringify` (but you would need to ensure that the values were properly and safely encoded):

```
// requires json2.js for all browsers other than Firefox
3.5
if (!window.sessionStorage && JSON) {
  window.sessionStorage = (function () {
    // window.top.name ensures top level, and supports
    around 2Mb
    var data = window.top.name ?
JSON.parse(window.top.name) : {};
    return {
      setItem: function (key, value) {
        data[key] = value+""; // force to string
        window.top.name = JSON.stringify(data);
      },
      removeItem: function (key) {
        delete data[key];
        window.top.name = JSON.stringify(data);
      },
      getItem: function (key) {
        return data[key] || null;
      }
    };
  })();
}
```

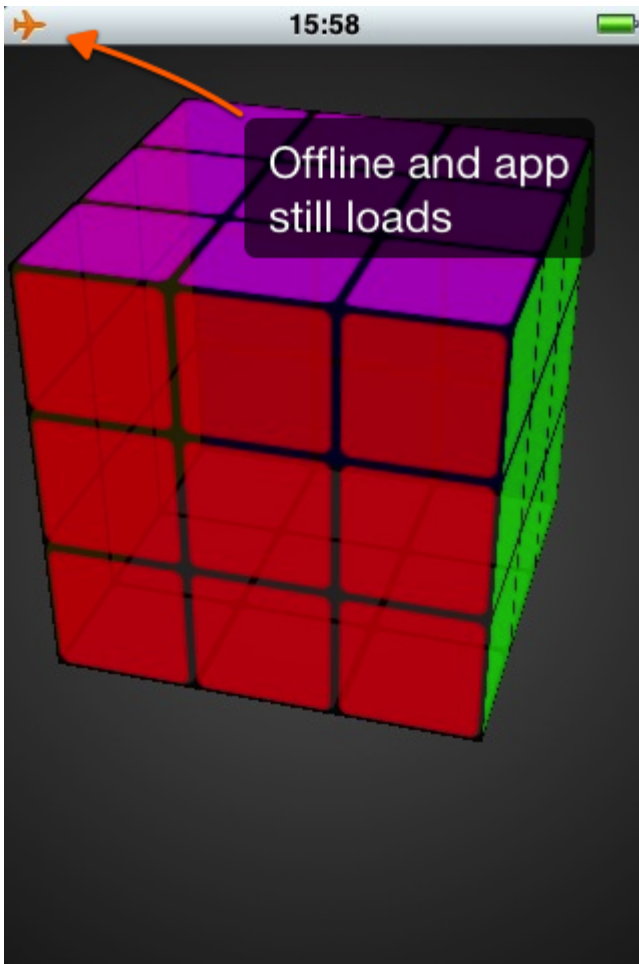
```
    },
    clear: function () {
        data = {};
        window.top.name = '';
    }
};
})();
}
```

Now that we've cracked the cookie jar with our oversized Web Storage API, let's have a look at how we take our applications offline entirely.

## OFFLINE APPLICATIONS

Offline applications is (still) part of the HTML5 specification. It allows developers to build a web app and have it still function without an internet connection. The app is access via the same URL as it would be if the user were online, but the contents (or what the developer specifies) is served up to the browser from a local cache. From there it's just an everyday stroll through open web technologies, i.e. you still have access to the Web Storage API and anything else you can do without a web connection.

For this section, I'll refer you to a prototype demo I wrote recently of a **contrived Rubik's cube** (contrived because it doesn't work and it only works in Safari because I'm using 3D transforms).



Offline Rubik's cube

---

## Support

Support for offline applications is still fairly limited, but the possibilities of offline applications is pretty exciting, particularly as we're seeing mobile support and support in applications such as Fluid (and I would expect other render engine wrapping apps).

Support currently, is as follows:

- Latest: Safari (desktop & mobile/iPhone)
- Sort of: Firefox<sup>‡</sup>
- Not supported: Internet Explorer, Opera, Google Chrome

‡ Firefox 3.5 was released to include offline support, but in fact has bugs where it doesn't work properly (certainly on the Mac), Minefield (Firefox beta) has resolved the bug.

## Usage

The status of the application's cache can be tested from the `window.applicationCache` object. However, we'll first look at how to enable your app for offline access.

You need to create a manifest file, which will tell the browser what to cache, and then we point our web page to that cache:

```
<!DOCTYPE html>
<html manifest="remy.manifest">
<!-- continues ... -->
```



For the manifest to be properly read by the browser, your server needs to serve the .manifest files as `text/manifest` by adding the following to your `mime.types`:

```
text/cache-manifest manifest
```

Next we need to populate our manifest file so the browser can read it:

```
CACHE MANIFEST
/demo/rubiks/index.html
/demo/rubiks/style.css
/demo/rubiks/jquery.min.js
/demo/rubiks/rubiks.js
# version 15
```

The first line of the manifest must read `CACHE MANIFEST`. Then subsequent lines tell the browser what to cache.

The HTML5 spec recommends that you include the calling web page (in my case `index.html`), but it's not required. If I didn't include `index.html`, the browser would cache it as part of the offline resources.

These resources are implicitly under the `CACHE` namespace (which you can specify any number of times if you want to).

In addition, there are two further namespaces: `NETWORK` and `FALLBACK`.

NETWORK is a whitelist namespace that tells the browser not to cache this resource and always try to request it through the network.

FALLBACK tells the browser that whilst in offline mode, if the resource isn't available, it should return the fallback resource.

Finally, in my example I've included a comment with a version number. This is because once you include a manifest, the **only** way you can tell the browser to reload the resources is if the manifest contents changes. So I've included a version number in the manifest which I can change forcing the browser to reload all of the assets.

## How it works

If you're building an app that makes use of the offline cache, I would strongly recommend that you add the manifest last. The browser implementations are very new, so can sometimes get a bit tricky to debug since once the resources are cached, they **really** stick in the browser.

These are the steps that happen during a request for an app with a manifest:

1. Browser: sends request for your app.html
2. Server: serves all associated resources with app.html – as normal

3. Browser: notices that app.html has a manifest, it re-request the assets in the manifest
4. Server: serves the requested manifest assets (again)
5. Browser: window.applicationCache has a status of UPDATEREADY
6. Browser: reloads
7. Browser: only request manifest file (which doesn't show on the net requests panel)
8. Server: responds with 304 Not Modified on the manifest file
9. Browser: serves all the cached resources locally

What might also add confusion to this process, is that the way the browsers work (currently) is if there is a cache already in place, it will use this first over updated resources. So if your manifest has changed, the browser will have already loaded the offline cache, so the user will only see the updated on the next reload.

This may seem a bit convoluted, but you can also trigger some of this manually through the applicationCache methods which can ease some of this pain.

If you bind to the online event you can manually try to update the offline cache. If the cache has then updated, swap the updated resources in to the cache and the next time the app loads it will be up to date. You could also prompt your user to reload the app (which is just a refresh) if there's an update available.

For example (though this is **just** pseudo code):

```
addEventListener(applicationCache, 'updateready', function () {
  applicationCache.swapCache();
  tellUserToRefresh();
});
addEventListener(window, 'online', function () {
  applicationCache.update();
});
```

## **BREAKING OUT OF THE BROWSER**

So that's two different technologies that you can use to break out of the traditional browser/web page model and get your apps working in a more **application-ny way**.

There's loads more in the **HTML5** and **non-HTML5 APIs** to play with, so take your Christmas break to check them out!

## ABOUT THE AUTHOR



**Remy Sharp** is a developer, speaker, blogger, author of upcoming **jQuery for Designers** (Manning) and contributing author of **The jQuery Cookbook** (O'Reilly), and most recently a conference organiser: **Full Frontal JavaScript Conference**. Remy started in web development 10 years ago as the sole developer for a finance web site, and as such, was exposed to all aspects running the web site during, and long after, the dotcom boom.

Curator of **Full Frontal JavaScript**, a UK JavaScript conference, contributing author to **HTML5 Doctor**, developer of **JS Bin**, **HTML5 Demos**, **Snap Bird** (JavaScript driven twitter search tool), jQuery team member and evangelist and developer on a bunch of other JavaScript related apps. Yeah, Remy likes his JavaScript!

# 3. Have a Field Day with HTML5 Forms

---

Inayaili de León Persson

[24ways.org/200903](http://24ways.org/200903)

Forms are usually seen as that obnoxious thing we have to markup and style. I respectfully disagree: forms (on a par with tables) are the most *exciting* thing we have to work with.

Here we're going to take a look at how to style a beautiful HTML5 form using some advanced CSS and latest CSS3 techniques. I promise you will *want* to style your own forms after you've read this article.

Here's what we'll be creating:

**Step 1: Your details**

Name

Email

Phone

**Step 2: Delivery address**




Address

Post code

Country

**Step 3: Card details**

Card type

 VISA   AmEx   Mastercard

Card number

Security code

Name on card

**BUY IT!**

## MEANINGFUL MARKUP

We're going to style a simple payment form. There are three main sections on this form:

- The person's details
- The address details
- The credit card details

We are also going to use some of HTML5's new input types and attributes to create more meaningful fields and use less unnecessary classes and ids:

- `email`, for the email field
- `tel`, for the telephone field
- `number`, for the credit card number and security code
- `required`, for required fields
- `placeholder`, for the hints within some of the fields
- `autofocus`, to put focus on the first input field when the page loads

There are a million more new input types and form attributes on HTML5, and you should definitely take a look at [what's new on the W3C website](#). Hopefully this will give you a good idea of how much more fun form markup can be.



## A GOOD FOUNDATION

Each section of the form will be contained within its own `fieldset`. In the case of the radio buttons for choosing the card type, we will enclose those options in another nested `fieldset`.

We will also be using an ordered list to group each `label / input` pair. This will provide us with a (kind of) semantic styling hook and it will also make the form easier to read when viewing with no CSS applied:

---

**Your details**

---

1. Name

2. Email

3. Phone

---

**Delivery address**

---

1. Address

2. Post code

3. Country

---

**Card details**

---

1. Card type

- 1.  VISA
- 2.  AmEx
- 3.  Mastercard

---

2. Card number

3. Security code

4. Name on card

---

---

The unstyled form

---

So here's the markup we are going to be working with:

## Have a Field Day with HTML5 Forms

```
<form id=payment>
  <fieldset>
    <legend>Your details</legend>
    <ol>
      <li>
        <label for=name>Name</label>
        <input id=name name=name type=text
placeholder="First and last name" required autofocus>
      </li>
      <li>
        <label for=email>Email</label>
        <input id=email name=email type=email
placeholder="example@domain.com" required>
      </li>
      <li>
        <label for=phone>Phone</label>
        <input id=phone name=phone type=tel
placeholder="Eg. +447500000000" required>
      </li>
    </ol>
  </fieldset>
  <fieldset>
    <legend>Delivery address</legend>
    <ol>
      <li>
        <label for=address>Address</label>
        <textarea id=address name=address rows=5
required></textarea>
      </li>
      <li>
        <label for=postcode>Post code</label>
        <input id=postcode name=postcode type=text
required>
      </li>
    </ol>
  </fieldset>
</form>
```

```

    <li>
      <label for=country>Country</label>
      <input id=country name=country type=text
required>
    </li>
  </ol>
</fieldset>
<fieldset>
  <legend>Card details</legend>
  <ol>
    <li>
      <fieldset>
        <legend>Card type</legend>
        <ol>
          <li>
            <input id=visa name=cardtype type=radio>
            <label for=visa>VISA</label>
          </li>
          <li>
            <input id=amex name=cardtype type=radio>
            <label for=amex>AmEx</label>
          </li>
          <li>
            <input id=mastercard name=cardtype
type=radio>
            <label for=mastercard>Mastercard</label>
          </li>
        </ol>
      </fieldset>
    </li>
    <li>
      <label for=cardnumber>Card number</label>
      <input id=cardnumber name=cardnumber type=number
required>

```

```
    </li>
    <li>
      <label for=secure>Security code</label>
      <input id=secure name=secure type=number
required>
    </li>
    <li>
      <label for=namecard>Name on card</label>
      <input id=namecard name=namecard type=text
placeholder="Exact name as on the card" required>
    </li>
  </ol>
</fieldset>
<fieldset>
  <button type=submit>Buy it!</button>
</fieldset>
</form>
```

## MAKING THINGS LOOK NICE

First things first, so let's start by adding some defaults to our form by resetting the margins and paddings of the elements and adding a default font to the page:

```
html, body, h1, form, fieldset, legend, ol, li {
  margin: 0;
  padding: 0;
}
body {
  background: #ffffff;
  color: #111111;
```

```
font-family: Georgia, "Times New Roman", Times, serif;
padding: 20px;
}
```

Next we are going to style the form element that is wrapping our fields:

```
form#payment {
  background: #9cbc2c;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  padding: 20px;
  width: 400px;
}
```

We will also remove the border from the `fieldset` and apply some bottom margin to it. Using the `:last-of-type` pseudo-class, we remove the bottom margin of the last `fieldset` — there is no need for it:

```
form#payment fieldset {
  border: none;
  margin-bottom: 10px;
}
form#payment fieldset:last-of-type {
  margin-bottom: 0;
}
```

Next we'll make the legends big and bold, and we will also apply a light-green text-shadow, to add that little extra special detail:

```
form#payment legend {  
  color: #384313;  
  font-size: 16px;  
  font-weight: bold;  
  padding-bottom: 10px;  
  text-shadow: 0 1px 1px #c0d576;  
}
```

Our legends are looking great, but how about adding a clear indication of how many steps our form has? Instead of adding that manually to every legend, we can use automatically generated counters.

To add a counter to an element, we have to use either the `:before` or `:after` pseudo-elements to add content via CSS. We will follow these steps:

- create a counter using the `counter-reset` property on the form element
- call the counter with the `content` property (using the same name we've created before)
- with the `counter-increment` property, indicate that for each element that matches our selector, that counter will be increased by 1

```
form#payment > fieldset > legend:before {  
  content: "Step " counter(fieldsets) ": ";  
  counter-increment: fieldsets;  
}
```

Finally, we need to change the style of the legend that is part of the radio buttons group, to make it look like a label:

```
form#payment fieldset legend {
  color: #111111;
  font-size: 13px;
  font-weight: normal;
  padding-bottom: 0;
}
```

## STYLING THE LISTS

For our list elements, we'll just add some nice rounded corners and semi-transparent border and background. Because we are using RGBa colors, we should provide a fallback for browsers that don't support them (that comes *before* the RBGa color). For the nested lists, we will remove these properties because they would be overlapping:

```
form#payment ol li {
  background: #b9cf6a;
  background: rgba(255,255,255,.3);
  border-color: #e3ebc3;
  border-color: rgba(255,255,255,.6);
  border-style: solid;
  border-width: 2px;
  -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  border-radius: 5px;
  line-height: 30px;
```



```
list-style: none;
padding: 5px 10px;
margin-bottom: 2px;
}
form#payment ol ol li {
background: none;
border: none;
float: left;
}
```

### FORM CONTROLS

Now we only need to style our labels, inputs and the button element.

All our labels will look the same, with the exception of the one for the radio elements. We will float them to the left and give them a width.

For the credit card type labels, we will add an icon as the background, and override some of the properties that aren't necessary. We will be using the attribute selector to specify the background image for each label — in this case, we use the `for` attribute of each label.

To add an extra user-friendly detail, we'll add a `cursor: pointer` to the radio button labels on the `:hover` state, so the user knows that he can simply click them to select that option.

```

form#payment label {
    float: left;
    font-size: 13px;
    width: 110px;
}
form#payment fieldset fieldset label {
    background:none no-repeat left 50%;
    line-height: 20px;
    padding: 0 0 0 30px;
    width: auto;
}
form#payment label[for=visa] {
    background-image: url(visa.gif);
}
form#payment label[for=amex] {
    background-image: url(amex.gif);
}
form#payment label[for=mastercard] {
    background-image: url(mastercard.gif);
}
form#payment fieldset fieldset label:hover {
    cursor: pointer;
}

```

Almost there! Now onto the input elements. Here we want to match all inputs, except for the radio ones, and the textarea. For that we will use the negation pseudo-class (:not()). With it we can target all input elements except for the ones with type of radio.

We will also make sure to add some :focus styles and add the appropriate styling for the radio inputs:

```
form#payment input:not([type=radio]),
form#payment textarea {
  background: #ffffff;
  border: none;
  -moz-border-radius: 3px;
  -webkit-border-radius: 3px;
  -khtml-border-radius: 3px;
  border-radius: 3px;
  font: italic 13px Georgia, "Times New Roman", Times,
serif;
  outline: none;
  padding: 5px;
  width: 200px;
}
form#payment input:not([type=submit]):focus,
form#payment textarea:focus {
  background: #eaeaea;
}
form#payment input[type=radio] {
  float: left;
  margin-right: 5px;
}
```

And finally we come to our submit button. To it, we will just add some nice typography and text-shadow, align it to the center of the form and give it some background colors for its different states:

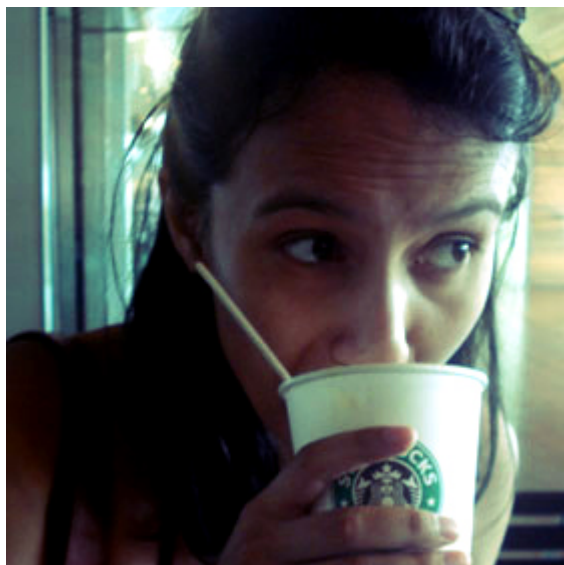
```
form#payment button {
  background: #384313;
  border: none;
  -moz-border-radius: 20px;
  -webkit-border-radius: 20px;
```

```
-khtml-border-radius: 20px;
border-radius: 20px;
color: #ffffff;
display: block;
font: 18px Georgia, "Times New Roman", Times, serif;
letter-spacing: 1px;
margin: auto;
padding: 7px 25px;
text-shadow: 0 1px 1px #000000;
text-transform: uppercase;
}
form#payment button:hover {
    background: #1e2506;
    cursor: pointer;
}
```

And that's it! [See the completed form.](#)

This form *will not* look the same on every browser. Internet Explorer and Opera don't support `border-radius` (at least not for now); the new input types are rendered as just normal inputs on some browsers; and some of the most advanced CSS, like the counter, `:last-of-type` or `text-shadow` are not supported on some browsers. But that doesn't mean you can't use them right now, and simplify your development process. My gift to you!

## ABOUT THE AUTHOR



**Inayaili de León Persson** (or just **Yaili**) is a web designer and author. She's Lead Web Designer at Canonical, the company that delivers **Ubuntu**. She's Panamanian Portuguese, born in the USSR, and has been living in London since 2008 – her favourite city in the world. She loves cats and naps.

## 4. What makes a website successful? It might not be what you expect!

---

Paul Boag

24ways.org/200904

What makes some sites succeed and others fail? Put another way, when you are asked to redesign an existing website, what problems are you looking out for and where do you concentrate your efforts?

I would argue that as web designers we spend too much time looking at the wrong kind of problem.

I recently ran a free open door **consultancy clinic** to celebrate the launch of **my new book** (yes I know, two shameless plugs in one sentence). This involved various website owners volunteering their sites for review. Both myself and the audience then provided feedback.

What quickly became apparent is that the feedback being given by the audience was biased towards design and development.

What makes a website successful? It might not be what you expect!

Although their comments were excellent it focused almost exclusively on the quality of code, site aesthetics and usability. To address these issues in isolation is similar to treating symptoms and ignoring the underlying illness.

## **CURE THE ILLNESS NOT THE SYMPTOMS**

Poor design, bad usability and terribly written code are symptoms of bigger problems. Often when we endeavour to address these symptoms, we meet resistance from our clients and become frustrated. This is because our clients are still struggling with fundamental concepts we take for granted.

Before we can address issues of aesthetics, usability and code, we need to tackle business objectives, calls to action and user tasks. Without dealing with these fundamental principles our clients' website will fail.

Let me address each in turn:

## **UNDERSTAND THE BUSINESS OBJECTIVES**

Do you ask your clients why they have a website? It feels like an obvious question. However, it is surprising how many clients do not have an answer.

Without having a clear idea of the site's business objectives, the client has no way to know whether it is succeeding. This means they have no justification for further investment and that leads to quibbling over every penny.

However most importantly, without clearly defined business aims they have no standard against which to base their decisions. Everything becomes subjective and that will inevitably lead to problems.

Before we start discussing design, usability and development, we need to focus our clients on establishing concrete business objectives. This will provide a framework for decision making during the development phase.

This will not only help the client make decisions, it will also focus them on the business and away from micro managing the design.

## **ESTABLISH CLEAR CALLS TO ACTION**

Once business objectives have been set this opens up the possibility to establish clear calls to action.

I am amazed at how few website owners can name their calls to action. However, I am even more staggered at how few web designers ask about them.



What makes a website successful? It might not be what you expect!

Calls to action are not just limited to ecommerce sites. Whether you are asking people to sign up for a newsletter or complete a contact us form, every site should have a desired objective for users.

What is more, *each page* of a site should have micro calls to action that always draw users on and never leave them at a dead end.

Without clearly defined calls to action you cannot successfully design a site, structure the user experience or measure its success. They bring focus to the site and encourage the client to concentrate their efforts on helping people reach those goals.

Of course in order to know if a call to action is going to work, it is necessary to do some user testing.

## **TEST AGAINST THE RIGHT TASKS**

As web designers we all like to boast about being ‘user centric’ whatever that means! However, in reality I think many of us are paying lip service to the subject.

Sure, we ask our clients about who their users are and maybe even do some usability testing. However, usability testing is no good if we are not asking the right questions.

Again we find ourselves working on a superficial level rather than tackling the deeper issues.

Clients find it relatively easy to tell you who their target audience is. Admittedly the list they come back with is often overly long and contains a lot of edge cases. However, where they begin to struggle is articulating what these users will want to achieve on the website. They know who they want to reach. However, they cannot always tell you why those people would be interested in the site.

These user tasks are another fundamental building block for any successful website. Although it is important for a website owner to understand what their objectives are and what they want users to do, it is even more important that they understand the users objectives as well.

Again, this provides context for the decisions they are making about design, usability and functionality. Without it the site will become self serving, largely ignoring the needs of users.

User tasks help to focus the client's mind on the needs of their user, rather than what they can get out of them.

So am I claiming that design, usability and code do not matter? Well the shocking truth is that to some extent I am!

What makes a website successful? It might not be what you expect!

## THE SHOCKING TRUTH

Whether we like it or not there is significant evidence that you can create a successful website with bad design, terrible code and without ever running a usability test session.

You only need to look at the design of Craigslist or the code of Amazon to see that this is true.

However, I do not believe it is possible to build a successful website without business objectives, calls to action and a clear idea of user tasks.

Do not misunderstand me. I do believe design, usability and code matters. I just believe that they only matter if the fundamentals are already in place. These things improve a solid foundation but are no use in their own right.

As web designers it is our responsibility to ensure fundamental questions are being asked, before we start exploring other issues. If we do not, our websites will look great, be well coded and have gone through endless usability tests, however it will not be truly successful.

## ABOUT THE AUTHOR



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

## 5. HTML5: Tool of Satan, or Yule of Santa?

---

Bruce Lawson

24ways.org/200905

It would lead to unseasonal arguments to discuss the title of this piece here, and the arguments are as indigestible as the fourth turkey curry of the season, so we'll restrict our article to the practical rather than the philosophical: what HTML5 can you reasonably expect to be able to use reliably cross-browser in the early months of 2010?

The answer is that you can use more than you might think, due to the seasonal tinsel of feature-detection and using the sparkly pixie-dust of IE-only VML (but used in a way that won't damage your Elf).

### **CANVAS**

canvas is a 2D drawing API that defines a blank area of the screen of arbitrary size, and allows you to draw on it using JavaScript. The pictures can be animated, such as in

this canvas mashup of *Wolfenstein 3D* and Flickr. (The difference between canvas and SVG is that SVG uses vector graphics, so is infinitely scalable. It also keeps a DOM, whereas canvas is just pixels so you have to do all your own book-keeping yourself in JavaScript if you want to know where aliens are on screen, or do collision detection.)

Previously, you needed to do this using Adobe Flash or Java applets, requiring plugins and potentially compromising keyboard accessibility. Canvas drawing is supported now in Opera, Safari, Chrome and Firefox. The reindeer in the corner is, of course, Internet Explorer, which currently has zero support for canvas (or SVG, come to that).

Now, don't pull a face like all you've found in your Yuletide stocking is a mouldy satsuma and a couple of nuts—that's not the end of the story. Canvas was originally an Apple proprietary technology, and Internet Explorer had a similar one called **Vector Markup Language** which was submitted to the W3C for standardisation in 1998 but which, unlike canvas, was not blessed with retrospective standardisation.

What you need, then, is some way for Internet Explorer to translate canvas to VML on-the-fly, while leaving the other, more standards-compliant browsers to use the HTML5. And such a way exists—it's a JavaScript library

called `excanvas`. It's downloadable from <http://code.google.com/p/explorercanvas/> and it's simple to include it via a conditional comment in the head for IE:

```
<!--[if IE]>  
<script src="excanvas.js"></script>  
<![endif]-->
```

Simply include this, and your canvas will be natively supported in the modern browsers (and the library won't even be downloaded) whereas IE will suddenly render your canvas using its own VML engine. Be sure, however, to check it carefully, as the IE JavaScript engine isn't so fast and you'll need to be sure that performance isn't too degraded to use.

## FORMS

Since the beginning of the Web, developers have been coding forms, and then writing JavaScript to check whether an input is a correctly formed email address, URL, credit card number or conforms to some other pattern. The cumulative labour of the world's developers over the last 15 years makes whizzing round in a sleigh and delivering presents seem like popping to the corner shop in comparison.

With HTML5, that's all about to change. As **Yaili** began to explore on **Day 3**, a host of new attributes to the input element provide built-in validation for email address

formats (`input type=email`), URLs (`input type=url`), any pattern that can be expressed with a JavaScript-syntax regex (pattern="`[0-9][A-Z]{3}`") and the like. New attributes such as `required`, `autofocus`, `input type=number min=3 max=50` remove much of the tedious JavaScript from form validation.

Other, really exciting input types are available (see **all input types**). The `datalist` is reminiscent of a select box, but allows the user to enter their own text if they don't want to choose one of the pre-defined options. `input type=range` is rendered as a slider, while `input type=date` pops up a date picker, all natively in the browser with no JavaScript required at all.

Currently, support is most complete in an experimental implementation in Opera and a number of the new attributes in Webkit-based browsers. But don't let that stop you! The clever thing about the specification of the new Web Forms is that all the new input types are attributes (rather than elements). `input` defaults to `input type=text`, so if a browser doesn't understand a new HTML5 type, it gracefully degrades to a plain text input.

So where does that leave validation in those browsers that don't support Web Forms? The answer is that you don't retire your pre-existing JavaScript validation just yet, but you leave it as a fallback after doing some feature detection. To detect whether (say) `input type=email` is



supported, you make a new `input type=email` with JavaScript but don't add it to the page. Then, you interrogate your new element to find out what its `type` attribute is. If it's reported back as "email", then the browser supports the new feature, so let it do its work and don't bring in any JavaScript validation. If it's reported back as "text", it's fallen back to the default, indicating that it's not supported, so your code should branch to your old validation routines. Alternatively, use the small (7K) **Modernizr library** which will do this work for you and give you JavaScript booleans like `Modernizr.inputtypes[email]` set to true or false.

So what does this buy you? Well, first and foremost, you're future-proofing your code for that time when all browsers support these hugely useful additions to forms. Secondly, you buy a usability and accessibility win. Although it's tempting to style the stuffing out of your form fields (which can, incidentally, **lead to madness**), whatever your branding people say, it's better to leave forms as close to the browser defaults as possible. A browser's slider and date pickers will be the same across different sites, making it much more comprehensible to users. And, by using native controls rather than faking sliders and date pickers with JavaScript, your forms are much more likely to be accessible to users of assistive technology.

## HTML5 DOCTYPE

You can use the new DOCTYPE `!doctype html` now and – hey presto – you’re writing HTML5, as it’s pretty much a superset of HTML4. There are some useful advantages to doing this. The first is that the HTML5 validator (I use <http://html5.validator.nu>) also validates ARIA information, whereas the HTML4 validator doesn’t, as ARIA is a new spec developed after HTML4. (Actually, it’s more accurate to say that it doesn’t validate your ARIA attributes, but it doesn’t automatically report them as an error.)

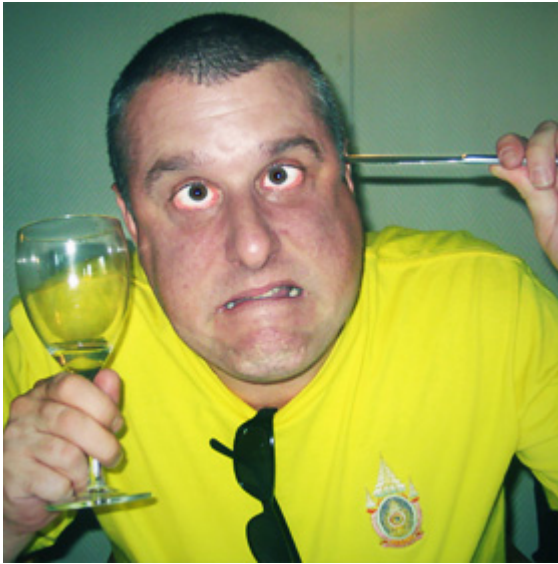
Another advantage is that **HTML5 allows `tabindex` as a global attribute** (that is, on any element). Although originally designed as an accessibility bolt-on, I ordinarily advise you don’t use it; a well-structured page should provide a logical tab order through links and form fields already.

However, `tabindex="-1"` is a legal value in HTML5 as it allows for the element to be programmatically focussable by JavaScript. It’s also very useful for correcting a bug in Internet Explorer when used with a keyboard; **in-page links go nowhere** if the destination doesn’t have a proprietary property called `hasLayout` set or a `tabindex` of `-1`.

## HTML5: Tool of Satan, or Yule of Santa?

So, whether it is the tool of Satan or yule of Santa, HTML5 is just around the corner. Some you can use now, and by the end of 2010 I predict you'll be able to use a whole lot more as new browser versions are released.

### ABOUT THE AUTHOR



**Bruce Lawson** evangelises Open Web Standards for the Opera web browser, and has spent far too much time in 2009 thinking about HTML5.

# 6. Front-End Code Reusability with CSS and JavaScript

---

Trevor Morris

24ways.org/200906

Most web standards-based developers are more than familiar with creating their sites with semantic HTML with lots and lots of CSS. With each new page in a design, the CSS tends to grow and grow and more elements and styles are added. But CSS can be used to better effect.

The idea of object-oriented CSS isn't new. Nicole Sullivan has written a [presentation](#) on the subject and outlines two main concepts: separate structure and visual design; and separate container and content. Jeff Croft talks about [Applying OOP Concepts to CSS](#):

I can make a class of `.box` that defines some basic layout structure, and another class of `.rounded` that provides rounded corners, and classes of `.wide` and `.narrow` that define some widths, and then easily create boxes of varying widths and styles by assigning multiple classes to an element, without having to duplicate code in my CSS.

This concept helps reduce CSS file size, allows for great flexibility, rapid building of similar content areas and means greater consistency throughout the entire design. You can also take this concept one step further and apply it to site behaviour with JavaScript.

### **Build a versatile slideshow**

I will show you how to build multiple slideshows using jQuery, allowing varying levels of functionality which you may find on one site design. The code will be flexible enough to allow you to add previous/next links, image pagination and the ability to change the animation type. More importantly, it will allow you to apply any combination of these features.

Image galleries are simply a list of images, so the obvious choice of marking the content up is to use a `<ul>`. Many designs, however, do not cater to non-JavaScript versions of the website, and thus don't take in to account large

multiple images. You could also simply hide all the other images in the list, apart from the first image. This method can waste bandwidth because the other images might be downloaded when they are never going to be seen.

Taking this second concept — only showing one image — the only code you need to start your slideshow is an `<img>` tag. The other images can be loaded dynamically via either a per-page JavaScript array or via AJAX.

The slideshow concept is built upon the very versatile **Cycle jQuery Plugin** and is structured in to another reusable jQuery plugin. Below is the HTML and JavaScript snippet needed to run every different type of slideshow I have mentioned above.

```

<script type="text/javascript">
  jQuery().ready(function($) {
    $('img.slideshow').slideShow({
      images: ['1.jpg', '2.jpg', '3.jpg']
    });
  });
</script>
```

## Slideshow plugin

If you're not familiar with jQuery or how to write and author your own plugin there are plenty of articles to help you out.

jQuery has a chainable interface and this is something your plugin must implement. This is easy to achieve, so your plugin simply returns the collection it is using:

```
return this.each(  
    function () {}  
);
```

### LOCAL VARIABLES

To keep the JavaScript clean and avoid any conflicts, you must set up any variables which are local to the plugin and should be used on each collection item. Defining all your variables at the top under one statement makes adding more and finding which variables are used easier. For other tips, conventions and improvements check out [JSLint](#), the “JavaScript Code Quality Tool”.

```
var $$, $div, $images, $arrows, $pager,  
    id, selector, path, o, options,  
    height, width,  
    list = [], li = 0,  
    parts = [], pi = 0,  
    arrows = ['Previous', 'Next'];
```

### CACHE JQUERY OBJECTS

It is good practice to cache any calls made to jQuery. This reduces wasted DOM calls, can improve the speed of your JavaScript code and makes code more reusable.

The following code snippet caches the current selected DOM element as a jQuery object using the variable name `$$`. Secondly, the plugin makes its settings available to the **Metadata** plugin† which is best practice within jQuery plugins.

For each slideshow the plugin generates a `<div>` with a class of `slideshow` and a unique `id`. This is used to wrap the slideshow images, pagination and controls.

The base path which is used for all the images in the slideshow is calculated based on the existing image which appears on the page. For example, if the path to the image on the page was `/img/flowers/1.jpg` the plugin would use the path `/img/flowers/` to load the other images.

```
$$ = $(this);
o = $.metadata ? $.extend({}, settings, $$metadata()) :
settings;
id = 'slideshow-' + (i++ + 1);
$div = $('<div />').addClass('slideshow').attr('id', id);
selector = '#' + id + ' ';
path = $.attr('src').replace(/[\d-9]\.jpg/g, '');
options = {};
height = $$height();
width = $$width();
```

Note: the plugin uses conventions such as folder structure and numeric filenames. These conventions help with the reusable aspect of plugins and best practices.



## BUILD THE IMAGES

The cycle plugin uses a list of images to create the slideshow. Because we chose to start with one image we must now build the list programmatically. This is a case of looping through the images which were added via the plugin options, building the appropriate HTML and appending the resulting `<ul>` to the DOM.

```
$.each(o.images, function () {
  list[li++] = '<li>';
  list[li++] = '';
  list[li++] = '</li>';
});
$images = $('<ul />').addClass('cycle-images');
$images.append(list.join('')).appendTo($div);
```

Although jQuery provides the append method it is much faster to create one really long string and append it to the DOM at the end.

## UPDATE THE OPTIONS

Here are some of the options we're making available by simply adding classes to the `<img>`. You can change the slideshow effect from the default *fade* to the *sliding* effect. By adding the class of `stopped` the slideshow will not auto-play and must be controlled via pagination or previous and next links.

```

// different effect
if ($$.is('.slide')) {
    options.fx = 'scrollHorz';
}
// don't move by default
if ($$.is('.stopped')) {
    options.timeout = 0;
}

```

If you are using the same set of images throughout a website you may wish to start on a different image on each page or section. This can be easily achieved by simply adding the appropriate starting class to the <img>.

```

// based on the class name on the image
if ($$.is('[class*=start-]')) {
    options.startingSlide =
    parseInt($$.attr('class').replace(/.*start-([0-9]+).*/g,
"$1"), 10) - 1;
}

```

For example:

```



```

By default, and without JavaScript, the third image in this slideshow is shown. When the JavaScript is applied to the page the slideshow must know to start from the correct place, this is why the start class is required.

You could capture the default image name and parse it to get the position, but only the default image needs to be numeric to work with this plugin (and could easily be changed in future). Therefore, this extra specifically defined option means the plugin is more tolerant.

### PREVIOUS/NEXT LINKS

A common feature of slideshows is previous and next links enabling the user to manually progress the images. The Cycle plugin supports this functionality, but you must generate the markup yourself. Most people add these directly in the HTML but normally only support their behaviour when JavaScript is enabled. This goes against progressive enhancement. To keep with the best practice progress enhancement method the previous/next links should be generated with JavaScript.

The follow snippet checks whether the slideshow requires the previous/next links, via the `arrows` class. It restricts the Cycle plugin to the specific slideshow using the *selector* we created at the top of the plugin. This means multiple slideshows can run on one page without conflicting each other.

The code creates a `<ul>` using the `arrows` array we defined at the top of the plugin. It also adds a class to the slideshow container, meaning you can style different combinations of options in your CSS.

```

// create the arrows
if ($.is('.arrows') && list.length > 1) {
  options.next = selector + '.next';
  options.prev = selector + '.previous';
  $arrows = $('<ul />').addClass('cycle-arrows');
  $.each(arrows, function (i, val) {
    parts[pi++] = '<li class="" + val.toLowerCase() +
">';
    parts[pi++] = '<a href="#" + val.toLowerCase() +
">';
    parts[pi++] = '<span>' + val + '</span>';
    parts[pi++] = '</a>';
    parts[pi++] = '</li>';
  });
  $arrows.append(parts.join('')).appendTo($div);
  $div.addClass('has-cycle-arrows');
}

```

The arrow array could be placed inside the plugin settings to allow for localisation.

## PAGINATION

The Cycle plugin creates its own HTML for the pagination of the slideshow. All our plugin needs to do is create the list and selector to use. This snippet creates the pagination container and appends it to our specific slideshow container. It sets the Cycle plugin pager option, restricting it to the specific slideshow using the *selector*

we created at the top of the plugin. Like the previous/next links, a class is added to the slideshow container allowing you to style the slideshow itself differently.

```
// create the clickable pagination
if ($$.is('.pagination') && list.length > 1) {
  options.pager = selector + '.cycle-pagination';
  $pager = $('<ul />').addClass('cycle-pagination');
  $pager.appendTo($div);
  $div.addClass('has-cycle-pagination');
}
```

Note: the Cycle plugin creates a `<ul>` with anchors listed directly inside without the surrounding `<li>`.

Unfortunately this is invalid markup but the code still works.

### Demos

Well, that describes all the ins-and-outs of the plugin, but demos make it easier to understand! Viewing the source on the demo page shows some of the combinations you can create with a simple `<img>`, a few classes and some thought-out JavaScript.

[View the demos](#) →

## Decide on defaults

The slideshow plugin uses the exact same settings as the Cycle plugin, but some are explicitly set within the slideshow plugin when using the classes you have set.

When deciding on what functionality is going to be controlled via this class method, be careful to choose your defaults wisely. If all slideshows should auto-play, don't make this an option — make the option to stop the auto-play. Similarly, if every slideshow should have previous/next functionality make this the default and expose the ability to remove them with a class such as “no-pagination”.

In the examples presented on this article I have used a class on each `<img>`. You can easily change this to anything you want and simply apply the plugin based on the jQuery selector required.

## Grab your images

If you are using AJAX to load in your images, you can speed up development by deciding on and keeping to a folder structure and naming convention. There are two methods: basing the image path based on the current URL; or based on the src of the image. The first allows a different slideshow on each page, but in many instances a site will have a couple of sets of images and therefore the second method is probably preferred.

## **Metadata ‡**

A method which allows you to directly modify settings in certain plugins, which also uses the classes from your HTML already exists. This is a jQuery plugin called **Metadata**. This method allows for finer control over the plugin settings themselves. Some people, however, may dislike the syntax and prefer using normal classes, like above which when sprinkled with a bit more JavaScript allows you to control what you need to control.

## **The takeaway**

Hopefully you have understood not only what goes in to a basic jQuery plugin but also learnt a new and powerful idea which you can apply to other areas of your website.

The idea can also be applied to other common interfaces such as lightboxes or mapping services such as Google Maps – for example creating markers based on a list of places, each with different pin icons based the anchor class.

## ABOUT THE AUTHOR



**Trevor Morris** a twenty-something web developer based in the Midlands, UK. He is fluent in both front- and back-end coding, and develops usable and accessible front-end interfaces using web standards.

He very occasionally writes on his personal site at [trovster.com](http://trovster.com) but for more up to date commentary you can follow him on **Twitter**. He is also a prominent member of the **The Multipack**, a community of multi-talented Web professionals from across the West Midlands.



## 7. Type-Inspired Interfaces

---

Dan Mall

[24ways.org/200907](http://24ways.org/200907)

One of the things that terrifies me most about a new project is the starting point. How is the content laid out? What colors do I pick? Once things like that are decided, it becomes significantly easier to continue design, but it's the blank page where I spend the most time.

To that end, I often start by choosing type. I don't need to worry about colors or layout or anything else... just the right typefaces that support the art direction. (This article won't focus on *how* to choose a typeface, but there are some **really great resources** if you interested in that sort of thing.)

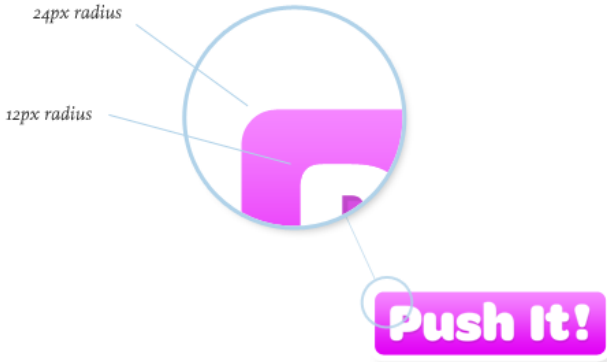
And just like that, all your work is done. "Hold it just a second," you might say. "All I've done is pick type. I still have to do the rest!"

To which I would reply, “Silly rabbit. You already have!” You see, picking the right typeface gets you farther than you might think. Here are a few tips on taking cues from type to design interfaces and interface elements.

## PERFECTING WEB 2.0

If you’re going for that beloved rounded corner look, you might class it up a bit by choosing the wonderful **Omnes Pro** by Joshua Darden. As the typeface already has a rounded aesthetic, making buttons that fit the style should be pretty easy.

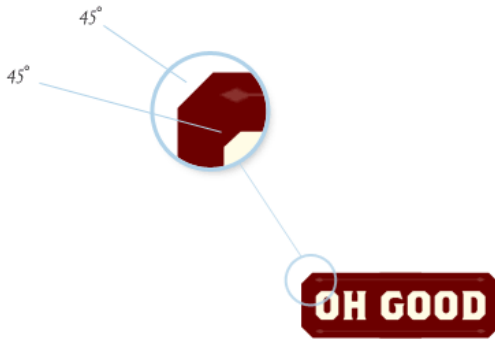
I’ve found that using multiples helps to keep your interfaces looking balanced and proportional. Noticing that the top left edge of the letter “P” has about an 12px corner radius, let’s choose a 24px radius for our button (a multiple of 2), so that we get **proper rounded corners**. By taking mathematical measurements from the typeface, our button looks more thought out than just “place arbitrary text on arbitrarily-sized button.” Pretty easy, eh?



## WHAT'S IN A NAME(PLATE)?

Rounded buttons are pretty popular buttons nowadays, so let's try something a bit more stylized.

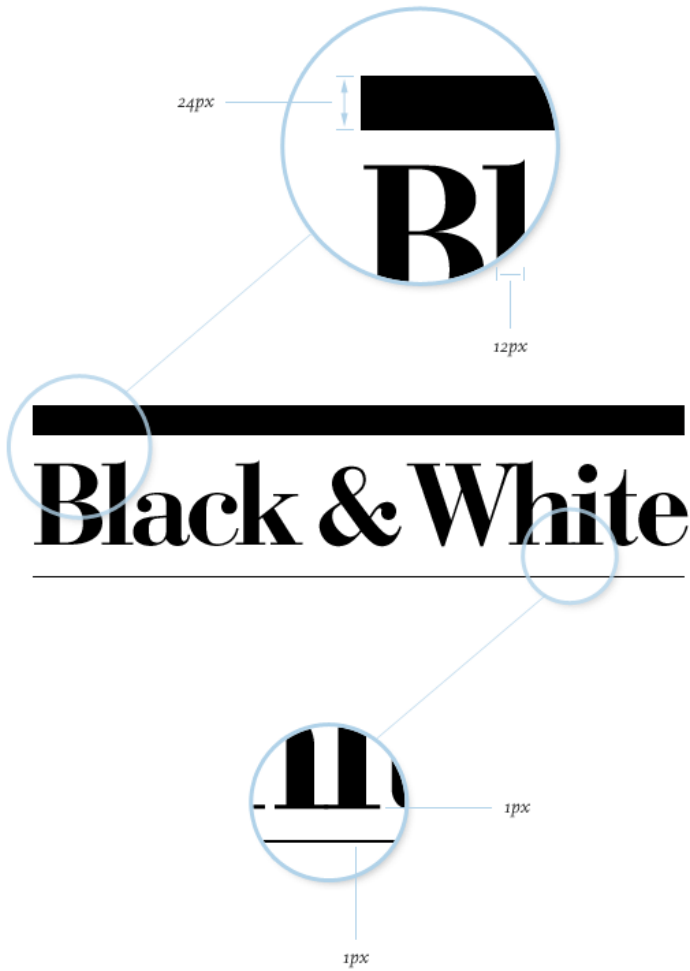
Have a gander at **Brothers**, a sturdy face from Emigre. The chiseled edges give us a perfect cue for a stylized button. Using the same slope, you can make plated-looking buttons that fit a different kind of style.



## HEADLINING

You might even take some cues from the style of the typeface itself. **Didone serifs** are known for their lack of brackets—that is, a gradual transition from the stem to the serif. Instead, they typically connect at a right angle. Another common characteristic is the high contrast in the strokes: very thick stems, very thin serifs.

So, when using a high contrast typeface, you can use it to your advantage to enhance hierarchy. Following our “multiples” guideline, a 12px measurement from the stems helps us create a top rule with a height of 24px (a multiple of 2). We can take the exact 1px measurement from the serif—a multiple of 1—to create the bottom rule. Voilà! I use this technique a lot.



## SWASHBUCKLERS

And don't forget the importance of visual "speed bumps" to break up long passages of text. A beautiful face like Alejandro Paul's *Ministry Script* has over a thousand

characters that can be manipulated or even combined to create elegant interface elements. Altering the partial differential character ( $\partial$ ) creates a delightful ornament that can help to guide the eye through content.

## A Child is Born

SED UT PERSPICIATIS UNDE OMNIS ISTE NATUS ERROR SIT DOLOREMQUE laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.



Totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsa voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptaem.

Et harum quidem rerum facilis est et expdita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda estisa dolor repellendus.



## STAGGER & SWAGGER

What about layout? How can we use typography to inform how our content is displayed?

Let's take a typeface like **Assembler**. We might use this for a design that needs to feel uneasy or uncomfortable. In design terms, that might translate into using irregular shapes and asymmetry. Using the proportional distances and degrees from the perpendiculars, we could easily create a multi-column layout that jives with the general tone. And for all you skeptics that don't think a layout like this is doable on the web, stranger things have happened.

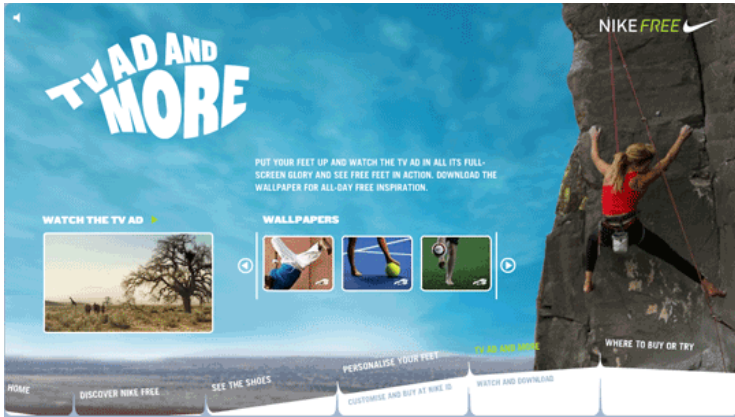


Background texture generously offered by **Bittbox**.

---

## OVERALL DESIGN DIRECTION

Finally, your typography could impact the entire look of the site, from the navigation to the interaction and everything in between. Check out how the (now-defunct) Nike Free site's typography echoes the product itself, and in turn influences the navigation.



## FIND YOUR TYPE

With thousands of fonts to choose from, the possibilities are ridiculously open. From angles to radii to color to weight, you've got endless fodder before you. Great type designers spent countless hours slaving over these



detailed letterforms; take advantage of it! Don't feel like you have to limit yourself to the same old Helvetica and wet floors... unless your design calls for it.

Happy hunting!

## ABOUT THE AUTHOR



**Dan Mall** is an award-winning interactive art director and designer. He is an enthralled husband, Senior Designer at **Big Spaceship**, former Interactive Director at **Happy Cog**, technical editor for **A List Apart**, co-founder of **Typedia**, and singer/keyboard player for contemporary-Christian band **Four24**. Dan writes about design and other issues on **Twitter** and his industry-recognized site, [danielmall.com](http://danielmall.com).

## 8. The Construction of Instruction

---

Relly Annett-Baker

24ways.org/200908

If the world were made to my specifications, all your clients would be happy to pay for a web writer to craft every sentence into something as elegant as it was functional, and the client would have planned the content so that you had it just when you asked, but we both know that won't happen every time. Sometimes you just know they are going to write the About page, two company blog pages and a Facebook fan page before resigning their position as chief content writer and *you* are going to end up filling in all the details that will otherwise just be Lorem Ipsum.

Welcome to the big world of microcopy:

A man walks into a bar. The bartender nods a greeting and watches as the man scans the bottles behind the bar.

“Er, you have a lot of gin here. Is there one you would recommend?”

“Yes sir.”

Long pause.

“... Never mind, I’ll have the one in the green bottle.”

“Certainly, sir. But you can’t buy it from this part of the bar. You need to go through the double doors there.”

“But they look like they lead into the kitchen.”

“Really, sir? Well, no, that’s where we allow customers to purchase gin.”

The man walks through the doors. On the other side he is greeted by the same bartender.

“Y-you!” he stammers but the reticent bartender is now all but silent.

Unnerved, the man points to a green bottle, “Er, I’d like to buy a shot of that please. With ice and tonic water.”

The bartender mixes the drink and puts it on the bar just out of the reach of the man and looks up.

“Um, do you take cards?” the man asks, ready to present his credit card.

The bartender goes to take the card to put it

through the machine.

“Wait! How much was it – with sales tax and everything? Do you take a gratuity?”

The bartender simply shrugs.

The man eyes him for a moment and decides to try his luck at the bar next door.

In the **Choose Your Own Adventure** version of this story there are plenty of ways to stop the man giving up. You could let him buy the gin right where he was; you could make the price more obvious; you could signpost the place to buy gin. The mistakes made by the bar and bartender are painfully obvious. And yet, there are websites losing users everyday due to the same lack of clear instruction.

A smidgen of well written copy goes a long way to reassure the nervous prospect. Just imagine if our man walked into the bar and the bartender explained that although the bar was here, sales were conducted in the next room because people were not then able to overhear the man’s card details. Instead, he is left to fend for himself. Online, we kick customers through the anonymous double doors with a merry ‘Paypal will handle your transaction!’.

Recently I worked on a site where the default error message, to account for anything happening that the developers hadn’t accounted for, was ‘SOMETHING HAS

GONE WRONG!'. It might have been technically accurate but this is not how to inspire confidence in your customers that they can make a successful purchase *through you*. As everyone knows they can shop just fine, thank you very much, it is your site they will blame. Card declined? It's the site. Didn't know my email address has changed? It's the site. Can't log in? It's the site.

Yes, yes. I know. None of these things are related to your site, or you the developer, but drop outs will be high and you'll get imploring emails from your client asking you to wade knee deep into the site analytics to find a solution by **testing 41 shades of blue** because if it worked for Google...? Before you try a visual fix involving the Dulux paint chart breeding with a Pantone swatch, take an objective look at the information you are giving customers. How much are you assuming they know? How much are you relying on age-old labels and prompts without clarification?

Here's a fun example for non-North Americans: ask your Granny to write out her billing address. If she looks at you blankly, tell her it is the address where the bank sends her statements. Imagine how many fewer instances of the wrong address there would be if we routinely added that information when people purchased from the UK? Instead, we rely on a language convention that hasn't

much common usage without explanation because, well, because we always have since the banks told us how we could take payments online.

So. Your client is busying themselves with writing the ultimate Facebook fan page about themselves and here you are left with creating a cohesive signup process or basket or purchase instructions. Here are five simple rules for ~~bending puny humans to your will~~ creating instructive instructions and constructive error messages that ultimately mean less hassle for you.

## **PLAN WHAT YOU WANT TO SAY AND PLAN IT OUT AS EARLY AS POSSIBLE**

This goes for all content. Walk a virtual mile in the shoes of your users. What specific help can you offer customers to actively encourage continuation and ensure a minimal amount of dropouts? Make space for that information.

One of the most common web content mistakes is jamming too much into a space that has been defined by physical boundaries rather than planned out. If you manage it, the best you can hope for is that no-one notices it was a last-minute job. Mostly it reads like a bad game of Tetris with content sticking out all over the place.

## USE YOUR WORDS

Microcopy often says a lot in a few words but without those words you could leave room for doubt. When doubt creeps in a customer wants reassurance just like Alice:

This time (Alice) found a little bottle... with the words ‘DRINK ME’ beautifully printed on it in large letters. It was all very well to say ‘Drink me,’ but the wise little Alice was not going to do that in a hurry. ‘No, I’ll look first,’ she said, ‘and see whether it’s marked “poison” or not’

Alice in Wonderland, Lewis Carroll.

Value clarity over brevity. Or a little more prosaically, “If in doubt, spell it out.” Thanks, Jeremy!

## BE PREPARED TO HELP

‘Login failed: email/password combination is incorrect.’

Oh.

‘Login failed: email/password combination is incorrect.

Are you typing in all capitals? Caps Lock may be on.

Have you changed your email address recently and not updated your account with us? Try your old email address first.

Can’t remember your password? We can help you reset it.’

Ah!

## **BE DIRECT AND BE INFORMATIVE**

There is rarely a site that doesn’t suffer from some degree of jargon. Squash it early by setting a few guidelines about what language and tone of voice you will use to converse with your users. Be consistent. Equally, try to be as specific as possible when giving error messages or instructions and allay fears upfront.

Card payments are handled by paypal but you do not need a paypal account to pay.

We will not display your email address but we might need it to contact you.

Sign up for our free trial (no credit card required).



## **COMBINE COPY AND VISUAL CUES, LEARN FROM OTHERS AND TEST NEW COMBINATIONS**

While visual design and copy can work independently, they work best together. New phrases and designs are being tested all the time so take a peek at [abtests.com](http://abtests.com) for more ideas, then test some new ideas and add your own results. Have a look at the [microcopy pool on Flickr](#) for some wonderful examples of little words and pictures working together. And yes, you absolutely should join the group and **post more examples**.

A man walks into a bar. The bartender greets him in a friendly manner and asks him what he would like to drink.

“Gin and Tonic, please.”

“Yes sir, we have our house gin on offer but we also have a particularly good import here too.”

“The import, please.”

“How would you like it? With a slice of lemon? Over ice?”

“Both”

“That’s £3.80. We accept cash, cards or you could open a tab.”

“Card please.”

“Certainly sir. Move just over here so that you can’t be observed. Now, please enter your pin number.”

“Thank you.”

“And here is your drink. Do let me know if there is a problem with it. I shall just be here at the bar. Enjoy.”

Cheers!

## ABOUT THE AUTHOR



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

# 9. Don't Lose Your :focus

---

Patrick Lauke

[24ways.org/200909](http://24ways.org/200909)

For many web designers, accessibility conjures up images of blind users with screenreaders, and the difficulties in making sites accessible to this particular audience. Of course, accessibility covers a wide range of situations that go beyond the extreme example of screenreader users. And while it's true that making a complex site accessible can often be a daunting prospect, there are also many small things that don't take anything more than a bit of judicious planning, are very easy to test (without having to buy expensive assistive technology), and can make all the difference to certain user groups.

In this short article we'll focus on keyboard accessibility and how careless use of CSS can potentially make your sites completely unusable.

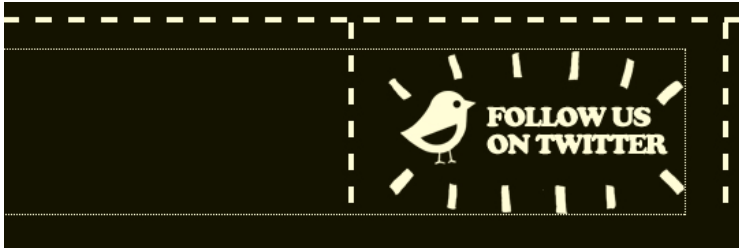
## KEYBOARD ACCESS

Users who for whatever reason can't use a mouse will employ a keyboard (or keyboard-like custom interface) to navigate around web pages. By default, they will use `TAB` and `SHIFT + TAB` to move from one focusable element (links, form controls and area) of a page to the next.

**Note:** in OS X, you'll first need to turn on full keyboard access under *System Preferences > Keyboard and Mouse > Keyboard Shortcuts*. Safari under Windows needs to have the option *Press Tab to highlight each item on a webpage* in *Preferences > Advanced* enabled. Opera is the odd one out, as it has a variety of keyboard navigation options – the most relevant here being **spatial navigation** via `Shift+Down`, `Shift+Up`, `Shift+Left`, and `Shift+Right`).

## BUT I DON'T LIKE YOUR DOTTED LINES...

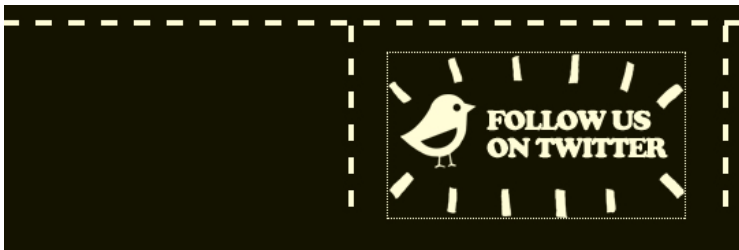
To show users where they are within a page, browsers place an *outline* around the element that currently has focus. The “problem” with these default outlines is that some browsers (Internet Explorer and Firefox) also display them when a user clicks on a focusable element with the mouse. Particularly on sites that make extensive use of image replacement on links with “off left” techniques this can create very unsightly outlines that stretch from the replaced element all the way to the left edge of the browser.



Outline bleeding off to the left (image-replacement example from [carsonified.com](http://carsonified.com))

---

There is a trivial workaround to prevent outlines from “spilling over” by adding a simple `overflow: hidden`, which keeps the outline in check around the clickable portion of the image-replaced element itself.



Outline tamed with `overflow: hidden`

---

But for many designers, even this is not enough. As a final solution, many actively suppress outlines altogether in their stylesheets. Controversially, even Eric Meyer’s popular *reset.css* – an otherwise excellent set of styles that levels the playing field of varying browser defaults – suppresses outlines.

---

```
html, body, div, span, applet, object, iframe ... {  
    ...  
    outline: 0;  
    ...  
}  
/* remember to define focus styles! */  
:focus {  
    outline: 0;  
}
```

Yes, in his explanation (and in the CSS itself) Eric does remind designers to define relevant styles for :focus... but judging by the number of sites that seem to ignore this (and often remove the related comment from the stylesheet altogether), the message doesn't seem to have sunk in.

Anyway... hurrah! No more unsightly dotted lines on our lovely design. But what about keyboard users? Although technically they can still `TAB` from one element to the next, they now get no default cue as to where they are within the page (one notable exception here is Opera, where the outline is displayed regardless of stylesheets)... and if they're Safari users, they won't even get an indication of a link's target in the status bar, like they would if they hovered over it with the mouse.

## ONLY SUPPRESS OUTLINE FOR MOUSE USERS

Is there a way to allow users navigating with the keyboard to retain the standard outline behaviour they've come to expect from their browser, while also ensuring that it doesn't show display for mouse users?

### Outline suppressed, reintroduced on `:focus`, suppressed on `:active`

[regular link](#)



```
a { outline: none; }
a:focus { outline: thin dotted; }
a:active { outline: none; }
```

Testing some convoluted style combinations

---

After playing with various approaches (see [Better CSS outline suppression](#) for more details), the most elegant solution also seemed to be the simplest: don't remove the outline on `:focus`, do it on `:active` instead – after all, `:active` is the dynamic pseudo-class that deals explicitly with the styles that should be applied when a focusable element is clicked or otherwise activated.

```
a:active { outline: none; }
```



The only minor issues with this method: if a user activates a link and then uses the browser's *back* button, the outline becomes visible. Oh, and old versions of Internet Explorer notoriously get confused by the exact meaning of `:focus`, `:hover` and `:active`, so this method fails in [IE6](#) and below. Personally, I can live with both of these.

**Note:** at the last minute before submitting this article, I discovered a fatal flaw in my test. It appears that `outline` still manages to appear in the time between activating a link and the link target loading (which in hindsight is logical – after activation, the link does indeed receive focus). As my test page only used in-page links, this issue never came up before. The slightly less elegant solution is to also suppress the outline on `:hover`.

```
a:hover, a:active { outline: none; }
```

## IN CONCLUSION

Of course, many web designers may argue that they know what's best, even for their keyboard-using audience. Maybe they've removed the default outline and are instead providing some carefully designed `:focus` styles. If they know for sure that these custom styles are indeed a reliable alternative for their users, more power to them... but, at the risk of sounding like Jakob "blue underlined links" Nielsen, I'd still argue that sometimes the default browser behaviours are best left alone.

Complemented, yes (and if you're already defining some fancy styles for `:hover`, by all means feel free to also make them display on `:focus`)... but not suppressed.



**AVAILABLE IN GERMAN**  
[webkrauts.de](http://webkrauts.de)

## ABOUT THE AUTHOR



**Patrick H. Lauke** works as Web Evangelist in the Developer Relations team at Opera Software. He has been engaged in the discourse on standards and accessibility since early 2001 – regularly speaking at conferences and contributing to a variety of web development and accessibility related mailing lists and

initiatives such as the **Web Standards Project** and the **Webkrauts**. For more of his ruminations and weird experiments you can visit **Patrick's personal site**.

# 10. A New Year's Resolution

---

Mike Kus

24ways.org/200910

The end of 2009 is fast approaching. Yet another year has passed in a split second. Our Web Designing careers are one year older and it's time to reflect on the highs and lows of 2009. What was your greatest achievement and what could you have done better? Perhaps, even more importantly, what are your goals for 2010?

Something that I noticed in 2009 is that being a web designer 24/7; it's easy to get consumed by the web. It's easy to get caught up in the blog posts, CSS galleries, web trends and Twitter! Living in this bubble can lead to one's work becoming stale, boring and basically like everyone else's work on the web. No designer wants this.

So, I say on 1st January 2010 let's make it our New Year's resolution to create something different, something special or even ground-breaking! Make it your goal to break the mold of current web design trends and light the way for your fellow web designer comrades!

Of course I wouldn't let you embark on the New Year empty handed. To help you on your way I've compiled a few thoughts and ideas to get your brains ticking!

## **DON'T DESIGN FOR THE WEB, JUST DESIGN**

A key factor in creating something original and fresh for the web is to stop thinking in terms of web design. The first thing we need to do is forget the notion of headers, footers, side bars etc. A website doesn't necessarily need any of these, so even before we've started we've already limited our design possibilities by thinking in these very conventional and generally accepted web terms. The browser window is a 2D canvas like any other and we can do with it what we like.

With this in mind we can approach web design from a fresh perspective. We can take inspiration for web design from editorial design, packaging design, comics, poster design, album artwork, motion design, street signage and anything else you can think of. Web design is way more than the just the web and by taking this more wide angled view of what web design is and can be you'll find there are a thousand more exiting design possibilities.

**Note:** Try leaving the wire framing till after you've gone to town with some initial design concepts. You might find it helps keep your head out of that 'web space' a little bit longer, thus enabling you to think more freely about your

design. Really go crazy with these as you can always pull it back into line later. The key is to think big initially and then work backwards. There's no point restricting your creativity early on because your technical knowledge can foresee problems down the line. You can always sort these problems out later on... let your creative juices flow!



Inspiration can come from anywhere! (Photo: [modomatic](#))

---

## **TRY SOMETHING NEW!**

Progress in web design or in any design discipline is a sort of evolution. Design trends and solutions merge and mutate to create new design trends and hopefully better solutions. This is fine but the real leaps are made when someone has the guts to do something different.

Don't be afraid to challenge the status quo. To create truly original work you have to be prepared to get it wrong and that's hard to do. When you're faced with this challenge just remind yourself that in web design there is rarely a 'best way to do something', or why would we ever do it any other way?

If you do this and get it right the pay off can be immense. Not only will you work stand out from the crowd by a mile, you will have become a trend setter as opposed to a trend follower.

## **TELL A STORY WITH YOUR DESIGN**

Great web design is way more than just the aesthetics, functionality or usability. Great web design goes beyond the pixels on the screen. For your website to make a real impact on it's users it has to connect with them emotionally. So, whether your website is promoting your own company or selling cheese it needs to **move** people. You need to weave a story into your design. It's this story that your users will connect with.

To do this the main ingredients of your design need to be strongly connected. In my head those main ingredients are Copy, Graphic Design, Typography, imagery and colour.

## **Copy**

Strong meaningful copy is the backbone to most great web design work. Pay special attention to strap lines and headlines as these are often the sparks that start the fire. All the other elements can be inspired by this backbone of strong copy.

## **Graphic Design**

Use the copy to influence how you treat the page with your graphic design. Let the design echo the words.

## **Typography**

What really excites me about typography isn't the general text presentation on a page, most half decent web designer have a grasp of this already. What excites me is the potential there is to base a whole design on words and letters. Using the strong copy you already have, one has the opportunity to customise, distort, build and arrange words and letters to create beautiful and powerful compositions that can be the basis for an entire site design.





Get creative with Typography (Photo: Pam Sattler)

---

## Imagery and Colour

With clever use of imagery (photographs or illustrations) and colour you further have the chance to deepen the story you are weaving into your design. The key is to use meaningful imagery, don't to insert generic imagery for the sake of filling space... it's just a wasted opportunity.

Remember, the main elements of your design combined are greater than the sum of their parts. Whatever design decisions you make on a page, make them for a good reason. It's not good enough to try and seduce your users with slick and shiny web pages. For your site to leave a lasting impression on the user you need to make that emotional connection.



Telling the Story (Advertising Agency: Tita, Milano, Italy, Art Director: Emanuele Basso)

## GO ONE STEP FURTHER

So you've almost finished your latest website design. You've fulfilled the brief, you're happy with the result and you're pretty sure your client will be too. It's at this point we should ask ourselves "Can I push this further"? What touches could you add to the site that'll take it beyond what was required and into something exceptional? The truth is, to produce exceptional work we need to do more than is required of us. We need to answer the brief and then some!

Go back through your site and make a note of what enhancements could be made to make the site not just good but outstanding. It might be revisiting a couple of pages that were neglected in the design process, it might be adding some CSS 3 gloss for the users that can benefit from it or it might just be adding some clever little easter eggs to show that you care. These touches will soon add up and make a massive difference to the finished product.

So, go one step further... take it further than you anyone else will. Then your work will stand out for sure.

## PARTING MESSAGE

I love being a designer for many of reasons but the main one being that with every new project we embark on we have the chance to express ourselves. We have the chance to create something special, something that

people will talk about. It's this chance that drives us onwards day after day, year after year. So in 2010 shout louder than you ever have before, take chances, try something new and above all design your socks off!

## ABOUT THE AUTHOR



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

# 11. Incite A Riot

---

Jeremy Keith

24ways.org/200911

Given its relatively limited scope, HTML can be remarkably expressive. With a bit of lateral thinking, we can mark up content such as tag clouds and progress meters, even when we don't have explicit HTML elements for those patterns.

Suppose we want to mark up a short conversation:

*Alice*: "I think Eve is watching."

*Bob*: "This isn't a cryptography tutorial ...we're in the wrong example!"

A note in the *the HTML 4.01 spec* says it's okay to use a definition list:

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

That would give us:

```
<dl>
  <dt>Alice</dt>: <dd>I think Eve is watching.</dd>
  <dt>Bob</dt>: <dd>This isn't a cryptography tutorial
  ...we're in the wrong example!</dd>
</dl>
```

This usage of a definition list is proof that writing W3C specifications and smoking crack are not mutually exclusive activities. “I think Eve is watching” is not a definition of “Alice.” If you (ab)use a definition list in this way, Norm will hunt you down.

The conversation problem was revisited in HTML5. What if dt and dd didn't always mean “definition title” and “definition description”? A new element was forged: dialog. Now the the “d” in dt and dd doesn't stand for “definition”, it stands for “dialog” (or “dialogue” if you can spell):

```
<dialog>
  <dt>Alice</dt>: <dd>I think Eve is watching.</dd>
  <dt>Bob</dt>: <dd>This isn't a cryptography tutorial
  ...we're in the wrong example!</dd>
</dialog>
```

Problem solved ...except that dialog is no longer in the HTML5 spec. Hixie further expanded the meaning of dt and dd so that they could be used inside details (which makes sense—it starts with a “d”) and figure (...um). At

the same time as the content model of details and figure were being updated, the completely-unrelated dialog element was dropped.

Back to the drawing board, or in this case, *the HTML 4.01 specification*. The spec defines the cite element thusly:

Contains a citation or a reference to other sources.

Perfect! There's even an example showing how this can be applied when attributing quotes to people:

```
As <CITE>Harry S. Truman</CITE> said,  
<Q lang="en-us">The buck stops here.</Q>
```

For longer quotes, the blockquote element might be more appropriate. In a conversation, where the order matters, I think an ordered list would make a good containing element for this pattern:

```
<ol>  
  <li><cite>Alice</cite>: <q>I think Eve is  
watching.</q></li>  
  <li><cite>Bob</cite>: <q>This isn't a cryptography  
tutorial ...we're in the wrong example!</q></li>  
</ol>
```

Problem solved ...except that the cite element has been redefined in the HTML5 spec:

The `cite` element represents the title of a work ... A person's name is not the title of a work ... and the element must therefore not be used to mark up people's names.

HTML5 is supposed to be backwards compatible with previous versions of HTML, yet here we have a semantic pattern already defined in HTML 4.01 that is now non-conforming in HTML5. The entire justification for the change boils down to this line of reasoning:

1. Given that: titles of works are often italicised and
2. given that: people's names are not often italicised and
3. given that: most browsers italicise the contents of the `cite` element,
4. therefore: the `cite` element should not be used to mark up people's names.

In other words, the default browser *styling* is now dictating semantic meaning. The tail is wagging the dog.

Not to worry, the HTML5 spec tells us how we can mark up names in conversations without using the `cite` element:

In some cases, the `b` element might be appropriate for names

I believe the colloquial response to this is a combination of the letters W, T and F, followed by a question mark.

The non-normative note continues:



In other cases, if an element is *really* needed, the `span` element can be used.

This is not a joke. We are seriously being told to use semantically meaningless elements to mark up content that is semantically meaningful.

We don't have to take it.

Firstly, any conformance checker—that's the new politically correct term for "validator"—cannot possibly check every instance of the `cite` element to see if it's really the title of a work and not the name of a person. So we can disobey the specification without fear of invalidating our documents.

Secondly, Hixie has repeatedly stated that browser makers have a powerful voice in deciding what goes into the HTML5 spec; if a browser maker refuses to implement a feature, then that feature should come out of the spec because otherwise, the spec is fiction. Well, one of the design principles of HTML5 is the *Priority of Constituencies*:

In case of conflict, consider users over authors over implementors over specifiers over theoretical purity.

That places us—authors—*above* browser makers. If we resolutely refuse to implement part of the HTML5 spec, then the spec becomes fiction.

Join me in a campaign of civil disobedience against the unnecessarily restrictive, backwards-incompatible change to the `cite` element. Start using HTML5 but start using it sensibly. Let's ensure that bad advice remains fictitious.

Tantek has set up a [page on the WHATWG wiki](#) to document usage of the `cite` element for conversations. Please contribute to it.

## 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.

# 12. Self-Testing Pages with JavaScript

---

Ross Bruniges

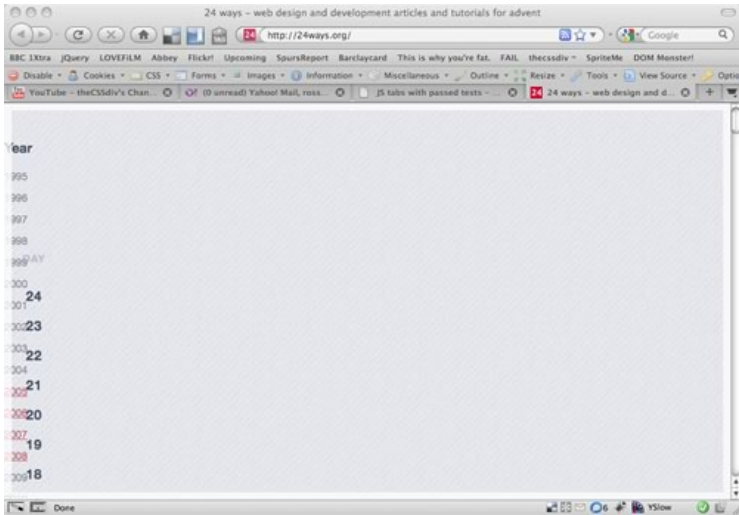
24ways.org/200912

Working at an agency I am involved more and more on projects in which client side code is developed internally then sent out to a separate team for implementation. You provide static HTML, CSS and JavaScript which then get placed into the CMS and brought to life as an actual website. As you can imagine this can sometimes lead to frustrations. However many safeguards you include, handing over your code to someone else is always a difficult thing to do effectively.

In this article I will show you how you can create a JavaScript implementation checker and that will give you more time for drink based activity as your web site and apps are launched quicker and with less unwanted drama!

## AN ALL TOO FREQUENT OCCURRENCE

You've been working on a project for weeks, fixed all your bugs and send it to be implemented. You hear nothing and assume all is going well then a few days before it's meant to launch you get an email from the implementation team informing you of bugs in your code that you need to urgently fix.



The 24ways website with a misspelt ID for the years menu

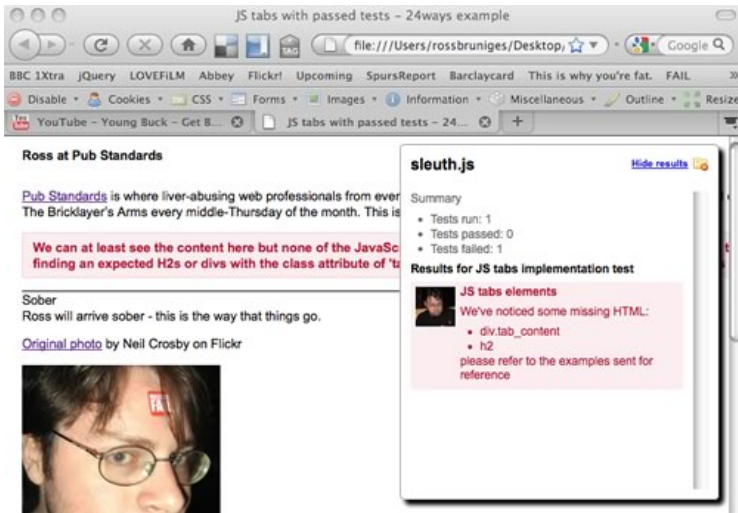
---

Being paranoid you trawl through the preview URL, check they have the latest files, check your code for errors then notice that a required HTML attribute has been omitted from the build and therefore CSS or JavaScript you've hooked onto that particular attribute isn't being applied and that's what is causing the "bug".

---

It takes you seconds drafting an email informing them of this, it takes then seconds putting the required attribute in and low and behold the bug is fixed, everyone is happy but you've lost a good few hours of your life – this time could have been better spent in the pub.

I'm going to show you a way that these kind of errors can be alerted immediately during implementation of your code and ensure that when you are contacted you know that there actually is a bug to fix. You probably already know the things that could be omitted from a build and look like bugs so you'll soon be creating tests to look for these and alert when they are not found on the rendered page. The error is reported directly to those who need to know about it and fix it. Less errant bug reports and less frantic emails ahoy!



A page with an implementation issue and instant feedback on the problem

---

### JAVASCRIPT SELECTOR ENGINES TO THE RESCUE

Whether you're using a library or indeed tapping into the loveliness of the new JavaScript Selector APIs looking for particular HTML elements in JavaScript is fairly trivial now.

For instance this is how you look for a `div` element with the `id` attribute of `year` (the missing attribute from top image) using jQuery (the library I'll be coding my examples in):

```
if ($('#div#year').length) {  
    alert('win');  
}
```

Using this logic you can probably imagine how you can write up a quick method to check for the existence of a particular element and alert when it's not present — but assuming you have a complex page you're going to be repeating yourself a fair bit and we don't want to be doing that.

## TEST SCRIPTS

If you've got a lot of complex HTML patterns that need testing across a number of different pages it makes sense to keep your tests out of production code. Chances are you've already got a load of heavy JavaScript assets, and when it comes to file size saving every little helps.

I don't think that tests should contain code inside of them so keep mine externally as JSON. This also means that you can use the one set of tests in multiple places. We already know that it's a good idea to keep our CSS and JavaScript separate so lets continue along those lines here.

The test script for this example looks like this:

```
{
  "title": "JS tabs implementation test",
  "description": "Check that the correct HTML patterns
has been used",
  "author": "Ross Bruniges",
  "created": "20th July 2009",
  "tests": [
    {
      "name": "JS tabs elements",
      "description": "Checking that correct HTML
elements including class/IDs are used on the page for
the JS to progressively enhance",
      "selector": "div.tabbed_content",
      "message": "We couldn't find VAR on the page -
it's required for our JavaScript to function correctly",
      "check_for": {
        "contains": {
```



```
    "elements": [  
      "div.tab_content", "h2"  
    ],  
    "message": "We've noticed some missing  
HTML:</p><ul><li>VAR</li></ul><p>please refer to the  
examples sent for reference"  
  }  
}  
]  
}
```

The first four lines are just a little bit of meta data so we remember what this test was all about when we look at it again in the future, or indeed if it ever breaks. The tests are the really cool parts and firstly you'll notice that it's an array – we're only going to show one example test here but there is no reason why you can't place in as many as you want. I'll explain what each of the lines in the example test means:

- name – short test name, I use this in pass/fail messaging later
- description – meta data for future reference
- selector – the root HTML element from which your HTML will be searched
- message – what the app will alert if the initial selector isn't found
- check\_for – a wrapper to hold inner tests – those run if the initial selector does match

- `contains` – the type of check, we’re checking that the selector contains specified elements
- `elements` – the HTML elements we are searching for
- `message` – a message for when these don’t match (VAR is substituted when it’s appended to the page with the name of any elements that don’t exist)

It’s very important to pass the function valid JSON (JSONLint is a great tool for this) otherwise you might get a console showing no tests have even been run.

## THE JAVASCRIPT THAT MAKES THIS HELPFUL

Again, this code should never hit a production server so I’ve kept it external. This also means that the only thing that’s needed to be done by the implementation team when they are ready to build is that they delete this code.

```
<script src="sleuth.js" type="text/javascript"></script>
<script type="text/javascript">
  $(document).ready(function() {
    sleuth.test_page.init('js_tabs_test.js');
  });
</script>
```

“View the full [JavaScript:/examples/self-testing-pages-with-javascript/js/tests/test\\_suite.js](https://github.com/24ways/24ways/blob/master/js_tabs_test.js)

The `init` function appends the test console to the page and inserts the CSS file required to style it (you don’t need to use pictures of me when tests pass and fail though I see

no reason why you shouldn't), goes and grabs the JSON file referenced and parses it. The methods to pass (tests\_pass) and fail (haz\_fail) the test I hope are pretty self-explanatory as is the one which creates the test summary once everything has been run (create\_summary).

The two interesting functions are `init_tests` and `confirm_html`.

## INIT\_TESTS

```
init_tests:function(i,obj) {
    var $master_elm = $(obj.selector);
    sleuth.test_page.$logger.append("<div id='test_' + i +
    "' class='message'><p><em>" + obj.name +
    "</em></p></div>");
    var $container = $('#test_' + i);
    if (!$master_elm.length) {
        var err_sum = obj.message.replace(/VAR/gi,
obj.selector);
        sleuth.test_page.haz_failed(err_sum, $container);
        return;
    }
    if (obj.check_for) {
        $.each(obj.check_for,function(key, value){
            sleuth.test_page.assign_checks($master_elm,
$container, key, value);
        });
    } else {
        sleuth.test_page.tests_passed($container);
    }
}
```

```
    return;  
  }  
}
```

The function gets sent the number of the current iteration (used to create a unique id for its test summary) and the current object that contains the data we're testing against as parameters.

We grab a reference to the root element and this is used (pretty much in the example shown right at the start of this article) and its `length` is checked. If the length is positive we know we can continue to the inner tests (if they exist) but if not we fail the test and don't go any further. We append the error to the test console for everyone to see.

If we pass the initial check we send the reference to the root element, message contains and the inner object to a function that in this example sends us on to `confirm_html` (if we had a more complex test suite it would do a lot more).

## CONFIRM\_HTML

```
confirm_html:function(target_selector, error_elm, obj) {  
  var missing_elms = [];  
  $.each(obj.elements, function(i, val) {  
    if (!target_selector.find(val).length) {  
      missing_elms.push(val);  
    }  
  }  
}
```

```
});  
if (missing_elms.length) {  
    var file_list = missing_elms.join('</li><li>');  
    var err_sum = obj.message.replace(/VAR/gi,  
file_list);  
    sleuth.test_page.haz_failed(err_sum, error_elm);  
    return;  
}  
sleuth.test_page.tests_passed(error_elm);  
return;  
}
```

We're again using an array to check for a passed or failed test and checking its length but this time we push in a reference to each missing element we find.

If the test does fail we're providing even more useful feedback by informing what elements have been missed out. All the implementation team need do is look for them in the files we've sent and include them as expected.

No more silly implementation bugs!

Here is an example of a successful implementation.

Here are some examples of failed implementations – one which fails at finding the root node and one that has the correct root node but none of the inner HTML tests pass.

### **IS THIS ALL WE CAN CHECK FOR?**

Certainly not!

JavaScript provides pretty easy ways to check for attributes, included files (if the files being checked for are being referenced correctly and not 404ing) and even applied CSS.

Want to check that those ARIA attributes are being implemented correctly or that all images contain an alt attribute well this simple test suite can be extended to include tests for this – the sky is pretty much up to your imagination.

## ABOUT THE AUTHOR



**Ross Bruniges** is a client-side engineer currently working at **LBi**, a large creative agency on Brick Lane in London. A long-serving **Pub Standards** regular, Ross likes beer, fine dining, **taking pictures of fine dining**, **making videos**, rap music and **twittering** the word beer (normally alongside other words too).

He has a **blog** in much need of a designers touch. Depending on how well his article on 24ways goes he plans to blog more in the future...

# 13. Rock Solid HTML Emails

---

David Greiner

24ways.org/200913

At some stage in your career, it's likely you'll be asked by a client to design a HTML email. Before you rush to explain that all the cool kids are using social media, keep in mind that when done correctly, email is still one of the best ways to promote you and your clients online. In fact, a recent survey showed that every dollar spent on email marketing this year generated more than \$40 in return. That's more than any other marketing channel, including the *cool ones*.

There are a whole host of ingredients that contribute to a good email marketing campaign. Permission, relevance, timeliness and engaging content are all important. Even so, the biggest challenge for designers still remains building an email that renders well across all the popular email clients.



## **SAME SAME, BUT DIFFERENT**

Before getting into the details, there are some uncomfortable facts that those new to HTML email should be aware of. Building an email *is not* like building for the web. While web browsers continue their onward march towards standards, many email clients have stubbornly stayed put. Some have even gone backwards. In 2007, Microsoft switched the Outlook rendering engine from Internet Explorer to Word. Yes, as in *the word processor*. Add to this the quirks of the major web-based email clients like Gmail and Hotmail, sprinkle in a little Lotus Notes and you'll soon realize how different the email game is.

While it's not without its challenges, rest assured it can be done. In my experience the key is to focus on three things. First, you should keep it simple. The more complex your email design, the more likely is it to choke on one of the popular clients with poor standards support. Second, you need to take your coding skills back a good decade. That often means nesting tables, bringing CSS inline and following the coding guidelines I'll outline below. Finally, you need to test your designs regularly. Just because a template looks nice in Hotmail now, doesn't mean it will next week.

## SETTING YOUR LOWEST COMMON DENOMINATOR

To maintain your sanity, it's a good idea to decide exactly which email clients you plan on supporting when building a HTML email. While **general research** is helpful, the email clients your subscribers are using can vary significantly from list to list. If you have the time there are a **number of** tools that can tell you specifically which email clients your subscribers are using. Trust me, if the testing shows almost none of them are using a client like Lotus Notes, save yourself some frustration and ignore it altogether.

Knowing which email clients you're targeting not only makes the building process easier, it can save you lots of time in the testing phase too. For the purpose of this article, I'll be sharing techniques that give the best results across all of the popular clients, including the notorious ones like Gmail, Lotus Notes 6 and Outlook 2007. Just remember that pixel perfection in all email clients is a pipe dream.

Let's get started.

## USE TABLES FOR LAYOUT

Because clients like Gmail and Outlook 2007 have poor support for `float`, `margin` and `padding`, you'll need to use tables as the framework of your email. While nested tables are widely supported, consistent treatment of

width, margin and padding within table cells is not. For the best results, keep the following in mind when coding your table structure.

### **Set the width in each cell, not the table**

When you combine table widths, td widths, td padding and CSS padding into an email, the final result is different in almost every email client. The most reliable way to set the width of your table is to set a width for each cell, not for the table itself.

```
<table cellspacing="0" cellpadding="10" border="0">
  <tr>
    <td width="80"></td>
    <td width="280"></td>
  </tr>
</table>
```

Never assume that if you don't specify a cell width the email client will figure it out. It won't. Also avoid using percentage based widths. Clients like Outlook 2007 don't respect them, especially for nested tables. Stick to pixels. If you want to add padding to each cell, use either the cellpadding attribute of the table or CSS padding for each cell, but never combine the two.

## Err toward nesting

Table nesting is far more reliable than setting left and right margins or padding for table cells. If you can achieve the same effect by table nesting, that will always give you the best result across the buggier email clients.

## Use a container table for body background colors

Many email clients ignore background colors specified in your CSS or the `<body>` tag. To work around this, wrap your entire email with a 100% width table and give that a background color.

```
<table cellpadding="0" cellspacing="0" border="0"
width="100%">
  <tr>
    <td bgcolor="#000000">
      Your email code goes here.
    </td>
  </tr>
</table>
```

You can use the same approach for background images too. Just remember that some email clients don't support them, so always provide a fallback color.

## **Avoid unnecessary whitespace in table cells**

Where possible, avoid whitespace between your `<td>` tags. Some email clients (ahem, Yahoo! and Hotmail) can add additional padding above or below the cell contents in some scenarios, breaking your design for no apparent reason.

## **CSS AND GENERAL FONT FORMATTING**

While some email designers do their best to avoid CSS altogether and rely on the dreaded `<font>` tag, the truth is many CSS properties are well supported by most email clients. See this comprehensive [list of CSS support](#) across the major clients for a good idea of the safe properties and those that should be avoided.

### **Always move your CSS inline**

Gmail is the culprit for this one. By stripping the CSS from the `<head>` and `<body>` of any email, we're left with no choice but to move all CSS inline. The good news is this is something you can almost completely automate. Free services like **Premailer** will move all CSS inline with the click of a button. I recommend leaving this step to the end of your build process so you can utilize all the benefits of CSS.

## Avoid shorthand for fonts and hex notation

A number of email clients reject CSS shorthand for the font property. For example, never set your font styles like this.

```
p {  
  font:bold 1em/1.2em georgia,times,serif;  
}
```

Instead, declare the properties individually like this.

```
p {  
  font-weight: bold;  
  font-size: 1em;  
  line-height: 1.2em;  
  font-family: georgia,times,serif;  
}
```

While we're on the topic of fonts, I recently tested every conceivable variation of @font-face across the major email clients. The results were dismal, so unfortunately it's web-safe fonts in email for the foreseeable future.

When declaring the color property in your CSS, some email clients don't support shorthand hexadecimal colors like color:#f60; instead of color:#ff6600;. Stick to the longhand approach for the best results.

## Paragraphs

Just like table cell spacing, paragraph spacing can be tricky to get a consistent result across the board. I've seen many designers revert to using double `<br />` or DIVs with inline CSS margins to work around these shortfalls, but recent testing showed that paragraph support is now reliable enough to use in most cases (there was a time when Yahoo! didn't support the paragraph tag at all).

The best approach is to set the `margin` inline via CSS for every paragraph in your email, like so:

```
p {
  margin: 0 0 1.6em 0;
}
```

Again, do this via CSS in the head when building your email, then use **Premailer** to bring it inline for each paragraph later.

If part of your design is height-sensitive and calls for pixel perfection, I recommend avoiding paragraphs altogether and setting the text formatting inline in the table cell. You might need to use table nesting or `cellpadding` / CSS to get the desired result. Here's an example:

```
<td width="200" style="font-weight:bold; font-size:1em;
line-height:1.2em;
font-family:georgia,'times',serif;">your height
sensitive text</td>
```

## Links

Some email clients will overwrite your link colors with their defaults, and you can avoid this by taking two steps. First, set a default color for each link inline like so:

```
<a href="http://somesite.com/" style="color:#ff00ff">this is a link</a>
```

Next, add a redundant span inside the a tag.

```
<a href="http://somesite.com/" style="color:#ff00ff"><span style="color:#ff00ff">this is a link</span></a>
```

To some this may be overkill, but if link color is important to your design then a superfluous span is the best way to achieve consistency.

## IMAGES IN HTML EMAILS

The most important thing to remember about images in email is that they won't be visible by default for many subscribers. If you start your design with that assumption, it forces you to keep things simple and ensure no important content is suppressed by image blocking.

With this in mind, here are the essentials to remember when using images in HTML email:



## **Avoid spacer images**

While the combination of spacer images and nested tables was popular on the web ten years ago, image blocking in many email clients has ruled it out as a reliable technique today. Most clients replace images with an empty placeholder in the same dimensions, others strip the image altogether. Given image blocking is on by default in most email clients, this can lead to a poor first impression for many of your subscribers. Stick to fixed cell widths to keep your formatting in place with or without images.

## **Always include the dimensions of your image**

If you forget to set the dimensions for each image, a number of clients will invent their own sizes when images are blocked and break your layout. Also, ensure that any images are correctly sized before adding them to your email. Some email clients will ignore the dimensions specified in code and rely on the true dimensions of your image.

## **Avoid PNGs**

Lotus Notes 6 and 7 don't support 8-bit or 24-bit PNG images, so stick with the GIF or JPG formats for all images, even if it means some additional file size.

## **Provide fallback colors for background images**

Outlook 2007 has no support for background images (aside from **this hack** to get full page background images working). If you want to use a background image in your design, always provide a background color the email client can fall back on. This solves both the image blocking and Outlook 2007 problem simultaneously.

## **Don't forget alt text**

Lack of standards support means email clients have long destroyed the chances of a semantic and accessible HTML email. Even still, providing alt text is important from an image blocking perspective. Even with images suppressed by default, many email clients will display the provided alt text instead. Just remember that some email clients like Outlook 2007, Hotmail and Apple Mail don't support alt text at all when images are blocked.

## **Use the display hack for Hotmail**

For some inexplicable reason, Windows Live Hotmail adds a few pixels of additional padding below images. A workaround is to set the display property like so.

```
img {display:block;}
```

This removes the padding in Hotmail and still gives you the predicable result in other email clients.

## Don't use floats

Both Outlook 2007 and earlier versions of Notes offer no support for the float property. Instead, use the align attribute of the `img` tag to float images in your email.

```

```

If you're seeing strange image behavior in Yahoo! Mail, adding `align="top"` to your images can often solve this problem.

## VIDEO IN EMAIL

With no support for JavaScript or the `object` tag, video in email (if you can call it that) has long been limited to animated gifs. However, some recent research I did into the HTML5 video tag in email showed some promising results.

Turns out HTML5 video *does* work in many email clients right now, including Apple Mail, Entourage 2008, MobileMe and the iPhone. The real benefit of this approach is that if the video isn't supported, you can provide reliable fallback content such as an animated GIF or a clickable image linking to the video in the browser.

Of course, the question of whether you should add video to email is another issue altogether. If you lean toward the "yes" side **check out the technique** with code samples.

## WHAT ABOUT MOBILE EMAIL?

The mobile email landscape was a huge mess until recently. With the advent of the iPhone, Android and big improvements from Palm and RIM, it's becoming less important to think of mobile as a different email platform altogether.

That said, there are a few key pointers to keep in mind when coding your emails to get a decent result for your more mobile subscribers.

### **Keep the width less than 600 pixels**

Because of email client preview panes, this rule was important long before mobile email clients came of age. In truth, the iPhone and Pre have a viewport of 320 pixels, the Droid 480 pixels and the Blackberry models hover around 360 pixels. Sticking to a maximum of 600 pixels wide ensures your design should still be readable when scaled down for each device. This width also gives good results in desktop and web-based preview panes.

### **Be aware of automatic text resizing**

In what is almost always a good feature, email clients using webkit (such as the iPhone, Pre and Android) can automatically adjust font sizes to increase readability. If

testing shows this feature is doing more harm than good to your design, you can always disable it with the following CSS rule:

```
-webkit-text-size-adjust: none;
```

## **DON'T FORGET TO TEST**

While standards support in email clients hasn't made much progress in the last few years, there has been continual change (for better or worse) in some email clients. Web-based providers like Yahoo!, Hotmail and Gmail are notorious for this. On countless occasions I've seen a proven design suddenly stop working without explanation.

For this reason alone it's important to retest your email designs on a regular basis. I find a quick test every month or so does the trick, especially in the web-based clients. The good news is that after designing and testing a few HTML email campaigns, you will find that order will emerge from the chaos. Many of these pitfalls will become quite predictable and your inbox-friendly designs will take shape with them in mind.

## **LOOKING AHEAD**

Designing HTML email can be a tough pill for new designers and standardistas to swallow, especially given the fickle and retrospective nature of email clients today.

With HTML5 just around the corner we are entering a new, uncertain phase. Will email client developers take the opportunity to repent on past mistakes and bring email clients into the present? The aim of groups such as the **Email Standards Project** is to make much of the above advice as redundant as the long-forgotten `<blink>` and `<marquee>` tags, however, only time will tell if this is to become a reality.

Although not the most compliant (or fashionable) medium, the results speak for themselves – email is, and will continue to be one of the most successful and targeted marketing channels available to you. As a designer with HTML email design skills in your arsenal, you have the opportunity to not only broaden your service offering, but gain a unique appreciation of how vital standards are.

## **NEXT STEPS**

Ready to get started? There are a number of HTML email design galleries to provide ideas and inspiration for your own designs.

- <http://www.campaignmonitor.com/gallery/>
- <http://htmlemailgallery.com/>
- <http://inboxaward.com/>

Enjoy!

## ABOUT THE AUTHOR



**David Greiner** is the co-founder of Campaign Monitor, email marketing software for web designers. He has been working with HTML email for more than a decade and started a number of initiatives to improve web standards support in email including the Email Standards Project and more recently the Fix Outlook campaign. You can follow him on [Twitter](#).

# 14. Going Nuts with CSS Transitions

---

Natalie Downe

24ways.org/200914

I'm going to show you how CSS 3 transforms and WebKit transitions can add zing to the way you present images on your site.

## LAYING THE FOUNDATIONS

First we are going to make our images look like mini polaroids with captions. Here's the markup:

```
<div class="polaroid pull-right">
  
  <p class="caption">Found this little cutie on a walk
in New Zealand!</p>
</div>
```

You'll notice we're using a somewhat presentational class of `pull-right` here. This means the logic is kept separate from the code that applies the polaroid effect. The `polaroid` class has no positioning, which allows it to be used generically anywhere that the effect is required. The `pull` classes set a float and add appropriate margins—they can be used for things like `blockquote`s as well.



```
.polaroid {  
  width: 150px;  
  padding: 10px 10px 20px 10px;  
  border: 1px solid #BFBFBF;  
  background-color: white;  
  -webkit-box-shadow: 2px 2px 3px rgba(135, 139, 144,  
0.4);  
  -moz-box-shadow: 2px 2px 3px rgba(135, 139, 144, 0.4);  
  box-shadow: 2px 2px 3px rgba(135, 139, 144, 0.4);  
}
```

The actual polaroid effect itself is simply applied using padding, a border and a background colour. We also apply a nice subtle box shadow, using a property that is supported by modern WebKit browsers and Firefox 3.5+. We include the `box-shadow` property last to ensure that future browsers that support the eventual CSS3 specified version natively will use that implementation over the legacy browser specific version.

The `box-shadow` property takes four values: three lengths and a colour. The first is the horizontal offset of the shadow—positive values place the shadow on the right, while negative values place it to the left. The second is the vertical offset, positive meaning below. If both of these are set to 0, the shadow is positioned equally on all four sides. The last length value sets the blur radius—the larger the number, the blurrier the shadow (therefore the darker you need to make the colour to have an effect).

The colour value can be given in any format recognised by CSS. Here, we're using `rgba` as explained by Drew behind the first door of this year's calendar.

## ROTATION

For browsers that understand it (currently our old favourites WebKit and FF3.5+) we can add some visual flair by rotating the image, using the `transform` CSS 3 property.

```
-webkit-transform: rotate(9deg);  
-moz-transform: rotate(9deg);  
transform: rotate(9deg);
```

Rotations can be specified in degrees, **radians** (`rads`) or **grads**. WebKit also supports turns unfortunately Firefox doesn't just yet.

For our example, we want any polaroid images on the left hand side to be rotated in the opposite direction, using a negative degree value:

```
.pull-left.polaroid {  
  -webkit-transform: rotate(-9deg);  
  -moz-transform: rotate(-9deg);  
  transform: rotate(-9deg);  
}
```

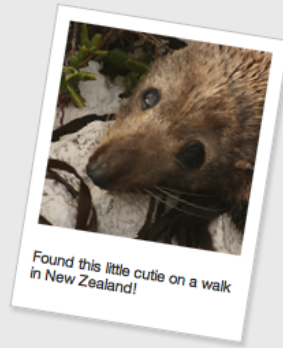
Multiple class selectors don't work in IE6 but as luck would have it, the `transform` property doesn't work in any current IE version either. The above code is a good

example of progressive enrichment: browsers that don't support box-shadow or transform will still see the image and basic polaroid effect.

## Example one

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



## ANIMATION

WebKit is unique amongst browser rendering engines in that it allows animation to be specified in pure CSS.

Although this may never actually make it in to the CSS 3 specification, it degrades nicely and more importantly is an awful lot of fun!

Let's go nuts.

In the next demo, the image is contained within a link and mousing over that link causes the polaroid to animate from being angled to being straight.

Here's our new markup:

```
<a href="http://www.flickr.com/photos/nataliedowne/2340993237/" class="polaroid">
  
  White water rafting in Queenstown
</a>
```

And here are the relevant lines of CSS:

```
a.polaroid {
  /* ... */
  -webkit-transform: rotate(10deg);
  -webkit-transition: -webkit-transform 0.5s ease-in;
}
a.polaroid:hover,
a.polaroid:focus,
a.polaroid:active {
  /* ... */
  -webkit-transform: rotate(0deg);
}
```

The `@-webkit-transition@` property is the magic wand that sets up the animation. It takes three values: the property to be animated, the duration of the animation and a 'timing function' (which affects the animation's acceleration, for a smoother effect).

-webkit-transition only takes affect when the specified property changes. In pure CSS, this is done using **dynamic pseudo-classes**. You can also change the properties using JavaScript, but that's a story for another time.

## THROWING POLAROIDS AT A TABLE

Imagine there are lots of differently sized polaroid photos scattered on a table. That's the effect we are aiming for with our next demo.



As an aside: we are using absolute positioning to arrange the images inside a flexible width container (with a minimum and maximum width specified in pixels). As some are positioned from the left and some from the right when you resize the browser they shuffle underneath each other. This is an effect used on the [UX London site](#).

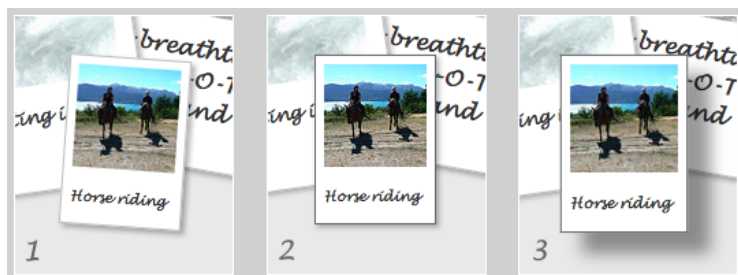
This demo uses a darker colour shadow with more transparency than before. The grey shadow in the previous example worked fine, but it was against a solid background. Since the images are now overlapping each other, the more opaque shadow looked fake.

```
-webkit-box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);  
-moz-box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);  
box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);
```

On hover, as well as our previous trick of animating the image rotation back to straight, we are also making the shadow darker and setting the z-index to be higher than the other images so that it appears on top.

## AND FINALLY...

Finally, for a bit more fun, we're going to simulate the images coming towards you and lifting off the page. We'll achieve this by making them grow larger and by offsetting the shadow & making it longer.



Screenshot 1 shows the default state, while 2 shows our previous hover effect. Screenshot 3 is the effect we are aiming for, **illustrated by demo 4**.

```
a.polaroid {
  /* ... */
  z-index: 2;
  -webkit-box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);
  -moz-box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);
  box-shadow: 2px 2px 4px rgba(0,0, 0, 0.3);
  -webkit-transform: rotate(10deg);
  -moz-transform: rotate(10deg);
  transform: rotate(10deg);
  -webkit-transition: all 0.5s ease-in;
}
a.polaroid:hover,
a.polaroid:focus,
a.polaroid:active {
  z-index: 999;
  border-color: #6A6A6A;
  -webkit-box-shadow: 15px 15px 20px rgba(0,0, 0, 0.4);
  -moz-box-shadow: 15px 15px 20px rgba(0,0, 0, 0.4);
  box-shadow: 15px 15px 20px rgba(0,0, 0, 0.4);
  -webkit-transform: rotate(0deg) scale(1.05);
  -moz-transform: rotate(0deg) scale(1.05);
  transform: rotate(0deg) scale(1.05);
}
```

You'll notice we are now giving the transform property another transform function: `scale`, which takes increases the size by the specified factor. **Other things you can do with transform** include skewing, translating or you can go mad creating your own transforms with a matrix.

The box-shadow has both its offset and blur radius increased dramatically, and is darkened using the alpha channel of the `rgba` colour.

And because we want the effects to all animate smoothly, we pass a value of `all` to the `-webkit-transition` property, ensuring that any changed property on that link will be animated.

Demo 5 is the finished example, bringing everything nicely together.

CSS transitions and transforms are a great example of progressive enrichment, which means improving the experience for a portion of the audience without negatively affecting other users. They are also a lot of fun to play with!

## FURTHER READING

- `-moz-transform` – the mozilla developer center has a comprehensive explanation of transform that also applies to `-webkit-transform` and `transform`.



- **CSS: Animation Using CSS Transforms** – this is a good, more indepth tutorial on animations.
- **CSS Animation** – the Safari blog explains the usage of `-webkit-transform`.
- **Dinky pocketbooks with transform** – another use for transforms, create your own printable pocketbook.
- A while back, **Simon** wrote a little bookmarklet to spin the entire page... warning: this will spin the entire page.

## ABOUT THE AUTHOR



**Natalie Downe** is an excitable client-side web developer at **Clearleft** in Brighton, a perfectionist by nature and comes with the expertise and breadth of knowledge of a web agency background. Although front-end development and usability

engineering are her first loves, Natalie still has fun dabbling with Python and poking the odd API. Natalie is also an experienced usability consultant and project manager.

# 15. CSS Animations

---

Tim Van Damme

24ways.org/200915

*Friend:* “You should learn how to write CSS!”

*Me:* “...”

*Friend:* “CSS; Cascading Style Sheets. If you’re serious about web design, that’s the next thing you should learn.”

*Me:* “What’s wrong with `<font>` tags?”

That was 8 years ago. Thanks to the hard work of Jeffrey, Andy, Andy, Cameron, Colly, Dan and many others, learning how to decently markup a website and write lightweight stylesheets was surprisingly easy. They made it so easy even a complete idiot (OH HAI) was able to quickly master it.

And then... nothing. For a long time, it seemed like there wasn’t happening anything in the land of CSS, time stood still. Once you knew the basics, there wasn’t anything new to keep up with. It looked like a great band split, but people just kept re-releasing their music in various “Best Of!” or “Remastered!” albums.

Fast forward a couple of years to late 2006. On the official WebKit blog *Surfin' Safari*, there's an article about something called **CSS animations**. Great new stuff to play with, but only supported by **nightly builds** (read: very, very beta) of WebKit. In the following months, they release other goodies, like **CSS gradients**, **CSS reflections**, **CSS masks**, and even more **CSS animation sexiness**. Whoa, looks like the band got back together, found their second youth, and went into overdrive! The problem was that if you wanted to listen to their new albums, you had to own some kind of new high-tech player no one on earth (besides some early adopters) owned.

Back in the time machine. It is now late 2009, close to Christmas. *Things have changed*. Browsers supporting these new toys are widely available **left and right**. Even non-techies are using these advanced browsers to surf the web on a daily basis!

Epic win? Almost, but at least this gives us enough reason to start learning how we could use all this new CSS voodoo. On Monday, Natalie Downe showed you a good tutorial on **Going Nuts with CSS Transitions**. Today, I'm taking it one step further...

## HOWTO: A BASIC SPINNER

No matter how fast internet tubes or servers are, we'll always need spinners to indicate something's happening behind the scenes. Up until now, people would go to **some site**, pick one of the available templates, customize their foreground and background colors, and download a beautiful GIF image.

There are some downsides to this though:

- It's only `_semi_`-transparent: If you change your mind and pick a slightly different background color, you need to go back to the site, set all the parameters again, and replace your current image. There isn't even a way to pick an image or gradient as background.
- Limited number of frames, probable to keep the file-size as small as possible (don't forget this thing needs to be loaded *before* whatever process is finished in the background), and you don't have that 24 frames per second smoothness.
- This is just too fucking easy. As a front-end code geek, there must be a "cooler" way to do this!

What do we need to make a spinner with CSS animations? One image, and one element on our webpage we can hook on to. Yes, that's it. I created a **simple transparent PNG** that looks it might be a spinner, and for the element on the page, I wrote this piece of genius HTML:

```
<p id="spinner">Please wait while we do what we do  
best.</p>
```

Looks semantic enough to me! Here's the basic HTML I'm using to position the element in the center of the screen, and make the text inside it disappear:

```
#spinner {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  margin: -100px 0 0 -100px;  
  height: 200px;  
  width: 200px;  
  text-indent: 250px;  
  white-space: nowrap;  
  overflow: hidden;  
}
```

Cool, but now we don't see anything. Let's pull rabbit number one out of the hat: `-webkit-mask-image` (accompanied by the previously mentioned transparent PNG image):

```
#spinner {  
  ...  
  -webkit-mask-image: url(..img/spinner.png);  
}
```

By now you should be feeling like a magician already. Oh, wait, we still have a blank screen, looks like we left something in the hat (tip: not rabbit droppings):

```
#spinner {  
  ...  
  -webkit-mask-image: url(../img/spinner.png);  
  background-color: #000;  
}
```

Nice! What we've done right here is telling the element to clip onto the PNG. It's a lot like clipping layers in Photoshop. So, spinners, they move, right? Into the hat again, and look what we pull out this time: CSS animations!

```
#spinner {  
  ...  
  -webkit-mask-image: url(../img/spinner.png);  
  background-color: #000;  
  -webkit-animation-name: spinnerRotate;  
  -webkit-animation-duration: 2s;  
  -webkit-animation-iteration-count: infinite;  
  -webkit-animation-timing-function: linear;  
}
```

Some explanation:

- `-webkit-animation-name`: Name of the animation we'll be defining later.
- `-webkit-animation-duration`: The timespan of the animation.
- `-webkit-animation-iteration-count`: Repeat once, a defined number of times or infinitely?

- `-webkit-animation-timing-function`: Linear is the one you'll be using mostly. Other options are ease-in, ease-out, ease-in-out...

Let's define `spinnerRotate`:

```
@-webkit-keyframes spinnerRotate {  
  from {  
    -webkit-transform: rotate(0deg);  
  }  
  to {  
    -webkit-transform: rotate(360deg);  
  }  
}
```

**En Anglais: Rotate #spinner starting at 0 degrees, ending at 360 degrees, over a timespan of 2 seconds, at a constant speed, and keep repeating this animation forever.**

That's it! See it in action on [the demo page](#).

*Note: these examples only work when you're using a WebKit-based browser like Safari, Mobile Safari or Google Chrome. I'm confident though that Mozilla and Opera will try their very best catching up with all this new CSS goodness soon.*

When looking at this example, you see the possibilities are endless. Another advantage is you can change the look of it entirely by only changing a couple of lines of CSS, instead of re-creating and re-downloading the image from some website smelling like web 2.0 gone bad. I made



another demo that shows how great it is to be able to change background and foreground colors (even on the fly!).

So there you have it, a smoothly animated, fully transparent and completely customizable spinner. Cool? I think so. (Ladies?)

But you can do a lot more with CSS animations than just create pretty spinners. Since I was fooling around with it anyway, I decided to test how far you can push this, space is the final limit, right?

## CONCLUSION

CSS has never been more exciting than it is right now. I'm even prepared to say CSS is "cool" again, both for the more experienced front-end developers as for the new designers discovering CSS every day now.

But...

Remember when Javascript became popular? Remember when Flash became popular? Every time we're been given new toys, some people aren't ashamed to use it in a way you can barely call constructive. I'm thinking of Geocities websites, loaded with glowing blocks of text, moving images, bad color usage... In the wise words of *Stan Lee*:

“With great power there must also come great responsibility!” A sprinkle of CSS animations is better than a bucket load. Apply with care.

## 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.

# 16. Designing For The Switch

---

Mark Boulton

[24ways.org/200916](http://24ways.org/200916)

For a long time on the web, we've been typographically spoilt. Yes, you heard me correctly. Think about it: our computers come with web fonts already installed; fonts that have been designed specifically to work well online and at small size; and fonts that we can be sure other people have too.

Yes, we've been spoilt. We don't need to think about using Verdana, Arial, Georgia or Cambria.

Yet, for a long time now, designers have felt we needed more. We want to choose whatever typeface we feel necessary for our designs. We did bad things along the way in pursuit of this goal such as images for text. Smart people dreamt up tools to help us such as sIFR, or Cufón. Only fairly recently, @font-face is supported in most

browsers. The floodgates are opening. It really is the dawn of a new typographic era on the web. And we must tread carefully.

## THE NEW TYPESETTERS

Many years ago, before the advent of desktop publishing, if you wanted words set in a particular typeface, you had to go to a Typesetter. A Typesetter, or Compositor, as they were sometimes called, was a person whose job it was to take the written word (in the form of a document or manuscript) and 'set' the type in the desired typeface. The designer would choose what typeface they wanted – and all the ligatures, underlines, italics and whatnot – and then scribble all over the manuscript so the typesetter could set the correct type.

Then along came Desktop Publishing and every Tom, Dick and Harry could choose type on their computer and an entire link in the typographic chain was removed within just a few years. Well, that's progress I guess. That was until six months ago when Typesetting was reborn on the web in the guise of a font service: **Typekit**.

Typekit – and services like Typekit such as **Typotheque**, **Kernest** and the upcoming **Fontdeck** – are typesetting services for the web. You supply them with your content,

in the form of a webpage, and they provide you with some JavaScript to render that webpage in the typeface you've specified simply by adding the font name in your CSS file.

Thanks to services like these, font foundries are now talking to create licensing structures to allow us to embed fonts into our web pages legally – which has always been a sticking point in the past. So, finally, us designers can get what we want: whatever typeface we want on the web.

Yes, but... there are hurdles. One of which is the subject of this article.

## **THE DIFFERENCES BETWEEN WEB FONTS AND OTHER FONTS**

Web fonts are different to normal fonts. They differ in a whole bunch of ways, from loose letter spacing to larger x-heights. But perhaps the most notable practical difference is file size. Let's take a look at one of Typekit's latest additions from the **FontFont** library, **Meta**.

Meta Roman weighs in at 42 KB. This is a fairly typical file size for a single weight of a good font. Now, let's have a look at Verdana. Verdana is 186 KB. For one weight. The four weight family for Verdana weighs in at 686 KB. Four weights for half a megabyte!?! Why so huge?

Well, Verdana has a lot of information packed into its 186 KB. It has the largest hinting data table of any typeface (the information carried by a font that tells it how to align itself to the pixels on your screen). As it has been shipped with Microsoft products since 1996, it has had time to grow to support many, many languages. Along with its cousin, Georgia (283 KB), Verdana was a new breed of typeface. And it's grown fat.

If really serious web typography takes off – and by that I mean typefaces specifically designed for the screen – then we're going to see more fonts increase in file size as the font files include more data. So, if you're embedding a font weighing in at 100 KB, what happens?

## THE FLASH OF UNSTYLED TEXT

We all remember the Flash of Unstyled Content bug on Internet Explorer, right? That annoying bug that caused a momentary flash of unstyled HTML page. Well, the same thing can happen with embedding fonts using `@font-face`. An effect called **The Flash of Unstyled Text** (FOUT), first coined by **Paul Irish**. Personally, I prefer to call it the Flash of UnTypeset Text (still FOUT), as the text **is** styled, just not with what you want.

If you embed a typeface in your CSS, then the browser will download that typeface. Typically, browsers differ in the way they handle this procedure.

Firefox and Opera will render the text using the next font in your font stack until the first (embedded) font is loaded. It will then switch to the embedded font.

Webkit takes the approach that you asked for that font so it will wait until it's completely loaded before showing it you.

In Opera and Firefox, you get a FOUT. In Webkit, you don't. You wait.

Hang on there. Didn't I say that good web fonts weigh in considerably more than 'normal' fonts? And whilst the browser is downloading the font, the user gets what to look at? Some pictures, background colours and whatever else isn't HTML? I believe Webkit's handling of font embedding – as deliberate as it is – is damaging to the practice of font embedding. Why? Well, we can design to a switch in typeface (as jarring as that is for the user), but we can't design to blank space.

Let's have a closer look at how we can design to FOUT.

## **MORE CONSIDERED FONT STACKS**

We all know that font stacks in CSS are there for when a user doesn't have a font; the browser will jump to the next one in the stack. Adding embedded fonts into the font stack means that because of FOUT (in gecko and Opera),

the user can see a switch, and depending on their connection that switch could happen well into any reading that the user may be doing.

The practicalities of this are that a user could be reading and be towards the end of a line when the paragraph they are reading changes shape. The word they were digesting suddenly changes to three lines down. It's the online equivalent of someone turning the page for you when you least expect it. So, how can we think about our font stacks slightly differently so we can minimise the switch?

Two years ago, Richard Rutter wrote on this very site about **increasing our font stacks**. By increasing the font stacks (by using his **handy matrix**) we can begin to experiment with different typefaces. However, when we embed a typeface, we must look very carefully at the typefaces in the font stack and the relationship between them. Because, previously, the user would not see a switch from one typeface to another, they'd just get either one **or** the other. Not both. With FOUT, the user sees *two typefaces*.

By carefully looking at the characteristics of the typefaces you choose, you can minimise the typographic 'distance' between the type down the stack. In doing so, you minimise the jarring effect of the switch.

Let's take a look at an example of how to go about this.



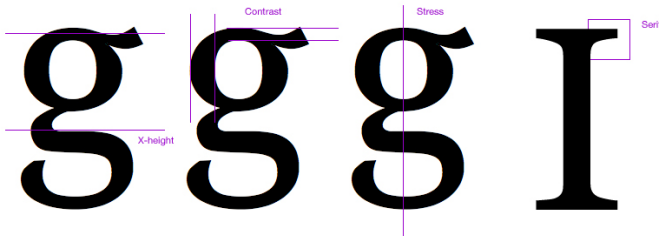
## MICRO TYPOGRAPHY TO BUILD BETTER FONT STACKS

Let's say I want to use a recent edition to Typekit – Meta Serif Book – as my embedded font. My font stack would start like this:

```
font-family: 'Meta Serif Bold';
```

Where do you go from here? Well, first, familiarise yourself with Richard's Font Matrix so you get an idea of what fonts are available for different people. Then start by looking closely at the characters of the embedded font and then compare them to different fonts from the matrix.

When I do this, I'm looking to match type characteristics such as x-height, contrast (the thickness and thinness of strokes), the stress (the angle of contrast) and the shape of the serifs (if the typeface has any).



Using just these simple comparative metrics means you can get to a 'best fit' reasonably quickly. And remember, you're not after an ideal match. You're after a match that means the switch is less painful for the reader, but also a typeface that carries similar characteristics so your design doesn't change too much.

Building upon my choice of embedded font, I can quickly build up a stack by comparing letters.



This then creates my 'best fit' stack.

**Too slow, chicken merango**

Meta Serif

**Too slow, chicken merango**

Lucida Bright

**Too slow, chicken merango**

Cambria

**Too slow, chicken merango**

Georgia

This translates to the CSS as:

```
font-family: 'Meta Serif Bold', 'Lucida Bright',  
Cambria, Georgia, serif
```

Following this process, and ending up with considered font stacks, means that we can design to the Flash of UnTypeset Content and ensure that our readers don't get a diminished experience.

## 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.

# 17. The Web Is Your CMS

---

Christian Heilmann

[24ways.org/200917](http://24ways.org/200917)

It is amazing what you can do these days with the services offered on the web. Flickr stores terabytes of photos for us and converts them automatically to all kind of sizes, finds people in them and even allows us to edit them online. YouTube does almost the same complete job with videos, LinkedIn allows us to maintain our CV, Delicious our bookmarks and so on.

We don't have to do these tasks ourselves any more, as all of these systems also come with ways to use the data in the form of Application Programming Interfaces, or APIs for short. APIs give us raw data when we send requests telling the system what we want to get back.

The problem is that every API has a different idea of what is a simple way of accessing this data and in which format to give it back.

## MAKING IT EASIER TO ACCESS APIS

What we need is a way to abstract the pains of different data formats and authentication formats away from the developer – and this is the purpose of the Yahoo Query Language, or YQL for short.

Libraries like jQuery and YUI make it easy and reliable to use JavaScript in browsers (yes, even IE6) and YQL allows us to access web services and even the data embedded in web documents in a simple fashion – SQL style.

## SELECT \* FROM THE WEB AND FILTER IT THE WAY I WANT

YQL is a web service that takes a few inputs itself:

- A query that tells it what to get, update or access
- An output format – XML, JSON, JSON-P or JSON-P-X
- A callback function (if you defined JSON-P or JSON-P-X)

You can try it out yourself – check out [this link](#) to get back Flickr photos for the search term

'santa'\*%20from%20flickr.photos.search%20where%20text%3D%22santa%22 in XML format. The YQL query for this is

```
select * from flickr.photos.search where text="santa"
```

The easiest way to take your first steps with YQL is to **look at the console**. There you get sample queries, access to all the data sources available to you and you can easily put together complex queries. In this article, however, let's use PHP to put together a web page that pulls in Flickr photos, blog posts, Videos from YouTube and latest bookmarks from Delicious.

**Check out the demo and get the source code on GitHub.**

```
<?php
    /* YouTube RSS */
    $query = 'select description from rss(5) where
url="http://gdata.youtube.com/feeds/base/users/
chrisheilmann/
uploads?alt=rss&v=2&orderby=published&client=ytapi-youtube-profile";';
    /* Flickr search by user id */
    $query .= 'select farm,id,owner,secret,server,title
from flickr.photos.search where user_id="11414938@N00";';
    /* Delicious RSS */
    $query .= 'select title,link from rss where
url="http://feeds.delicious.com/v2/rss/
codepo8?count=10";';
    /* Blog RSS */
    $query .= 'select title,link from rss where
url="http://feeds.feedburner.com/wait-till-i/gwZf"';
    /* The YQL web service root with JSON as the output */
    $root = 'http://query.yahooapis.com/v1/public/
yql?format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys';
    /* Assemble the query */
    $query = "select * from query.multi where
queries='\".$query.\"'";
    $url = $root . '&q=' . urlencode($query);
```

```

/* Do the curl call (access the data just like a
browser would) */
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
$output = curl_exec($ch);
curl_close($ch);
$data = json_decode($output);
$results = $data->query->results->results;
/* YouTube output */
$youtube = '<ul id="youtube">';
foreach($results[0]->item as $r){
$cleanHTML = undoYouTubeMarkupCrimes($r->description);
$youtube .= '<li>'.$cleanHTML.'</li>';
}
$youtube .= '</ul>';
/* Flickr output */
$flickr = '<ul id="flickr">';
foreach($results[1]->photo as $r){
$flickr .= '<li>'.
    '<a href="http://www.flickr.com/photos/
codepo8/'. $r->id.'/">'.
    'title.'"></a></li>';
}
$flickr .= '</ul>';
/* Delicious output */
$delicious = '<ul id="delicious">';
foreach($results[2]->item as $r){
$delicious .= '<li><a

```



```

href="'. $r->link. '">' . $r->title. '</a></li>';
    }
    $delicious .= '</ul>';
    /* Blog output */
    $blog = '<ul id="blog">';
    foreach($results[3]->item as $r){
    $blog .= '<li><a
href="'. $r->link. '">' . $r->title. '</a></li>';
    }
    $blog .= '</ul>';
    function undoYouTubeMarkupCrimes($str){
    $cleaner = preg_replace('/555px/', '100%', $str);
    $cleaner = preg_replace('/width="[^"]+"/', '', $cleaner);
    $cleaner = preg_replace('/<tbody>/', '<colgroup><col
width="20%"><col width="50%"><col
width="30%"></colgroup><tbody>', $cleaner);
    return $cleaner;
    }
    ?>

```

What we are doing here is create a few different YQL statements and queue them together with the `query.multi` table. Each of these can be run inside YQL itself. Check out the **YouTube**, **Flickr**, **Delicious** and **Blog** example in the console if you don't believe me. The benefit of using this table is that we don't make individual requests for each query but we get all the data in one single request – which means a much better performing solution as the YQL server farm is faster on the web than our servers.

We point the query to the YQL web service end point and get the resulting data using cURL. All that we need to do then is to convert the returned data to HTML lists that can be printed out inside an HTML template.

## MIXING, MATCHING AND USING HTML AS A DATA SOURCE

This was a simple example of what YQL can do for you. Where it gets really powerful however is by mixing and matching different APIs. YQL is also a good tool to get information from HTML documents. By using the `html` table you can load the content of an HTML document (which gets fixed automatically by HTMLTidy) and use XPATH to filter down results to what you need. Take the following example which takes headlines from the `news.bbc.co.uk` homepage and runs the results through Yahoo's Term Extractor API to give you a list of currently hot topics.

```
select * from search.termextract where context in (
  select content from html where
  url="http://news.bbc.co.uk" and
  xpath="//table[@width=800]//a"
)
```

Try it out in the [console](#) or see the [results here](#). In English, this means:

1. Go to `http://news.bbc.co.uk` and get me the HTML

2. Run it through HTML Tidy to clean it up.
3. Get me only the links inside the table with an attribute of width and the value 800
4. Get only the content of the link and for each of the links

1. Take the content and send it as context to the **Yahoo Term Extractor API**

If we choose JSON-P as the output format we can use the outcome directly in JavaScript (see [this demo](#) or see its [source](#)):

```
<ul id="hottopics"></ul>
<script type="text/javascript">
function hottopics(o){
  var res = o.query.results.Result,
      all = res.length,
      topics = {},
      out = [],
      html = '',
      i=0;
  /* create hash from topics to prevent repetition */
  for(i=0;i<all;i++){
    topics[res[i]] = res[i];
  };
  for(i in topics){
    out.push(i);
  };
  html = '<li>' + out.join('</li><li>') + '</li>';
  document.getElementById('hottopics').innerHTML = html;
};
</script>
```

```
<script type="text/javascript"
src="http://query.yahooapis.com/v1/public/
yql?q=select%20content%20from%20search.termextract%20where
%20context%20in%20(select%20content%20from%20html%20where%20url%3D%22h
```

Using JSON, we can also use PHP which means the demo works for everybody – not only those with JavaScript enabled (see this demo or see its source):

```
<ul id="hottopics"><li>
<?php
$url = 'http://query.yahooapis.com/v1/public/
yql?q=select%20content' .

'%20from%20search.termextract%20where%20context%20in' .

'%20(select%20content%20from%20html%20where%20url%3D%22' .

'http%3A%2F%2Fnews.bbc.co.uk%22%20and%20xpath%3D%22%2F%2F' .
    'table%5B%40width%3D800%5D%2F%2Fa%22)&format=json';
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
$output = curl_exec($ch);
curl_close($ch);
$data = json_decode($output);
$topics = array_unique($data->query->results->Result);
echo join('</li><li>', $topics);
?>
</li></ul>
```

## SUMMARY

This article could only scratch the surface of YQL. You have not only read access to the web but you can also write to web services. For example you can update Twitter, post to your WordPress blog or shorten a URL with bit.ly. Using Open Tables you can add any web service to the YQL interface and you can even run server-side JavaScript which is for example useful to return Flickr photos as HTML or get the HTML content from a document that needs POST data.

The web of data is already here, and using YQL you don't have to be a web services expert to use it and be part of it.



**AVAILABLE IN GERMAN**  
[webkrauts.de](http://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**.

# 18. A Pet Project is For Life, Not Just for Christmas

---

Elliot Jay Stocks

24ways.org/200918

I'm excited: as December rolls on, I'm winding down from client work and indulging in a big pet project I've been dreaming up for quite some time, with the aim of releasing it early next year. I've always been a bit of a sucker for pet projects and currently have a few in the works: the big one, two collaborations with friends, and my continuing (and completely un-web-related) attempt at music. But when I think about the other designers and developers out there whose work I admire, one thing becomes obvious: they've *all* got pet projects! Look around the web and you'll see that anyone worth their salt has some sort of side project on the go. If you don't have yours yet, now's the time!

## HAVE A PET PROJECT TO COLLABORATE WITH YOUR FRIENDS

It's not uncommon to find me staring at my screen, looking at beautiful websites my friends have made, grinning inanely because I feel so honoured to know such talented individuals. But one thing really frustrates me: I hardly ever get to work with these people! Sure, there are times when it's possible to do so, but due to various project situations, it's a rarity.

So, in order to work with my friends, I've found the best way is to instigate the collaboration *outside* of client work; in other words, have a pet project together! Free from the hard realities of budgets, time restraints, and client demands, you and your friends can come up with something purely for your own pleasures. If you've been looking for an excuse to work with other designers or developers whose work you love, the pet project *is* that excuse. They don't necessarily have to be friends, either: if the respect is mutual, it can be a great way of breaking the ice and getting to know someone.





Figure 1: A forthcoming secret love-child from myself and Tim Van Damme

---

## **HAVE A PET PROJECT TO ESCAPE FROM YOUR DAY JOB**

We all like to moan about our clients and bosses, don't we? But if leaving your job or firing your evil client just isn't an option, why not escape from all that and pour your creative energies into something you genuinely enjoy?

It's not just about reacting to negativity, either: a pet project is a great way to give yourself a bit of variety. As web designers, our day-to-day work forces us to work within a set of web-related constraints and sometimes it can be demoralising to spend so many hours fixing IE bugs. The perfect antidote? Go and do some print design! If it's not possible in your day job or client work, the pet project is the perfect place to exercise your other creative muscles. Yes, print design (or your chosen alternative) has its own constraints, but if they're different to those you experience on a daily basis, it'll be a welcome relief and you'll return to your regular work feeling refreshed.



Figure 2: Ligature, Loop & Stem, from Scott Boms & Luke Dorny

## HAVE A PET PROJECT TO FULFILL YOUR OWN NEEDS

Many pet projects come into being because the designers and/or developers behind them are looking for a tool to accomplish a task and find that it doesn't exist, thus

prompting them to create their own solution. In fact, the very app I'm using to write this article — **Ommwriter**, from **Herraiz Soto & Co** — was originally a tool they'd created for their internal staff, before releasing it to the public so that it could be enjoyed by others.

Just last week, **Tina Roth Eisenberg** launched **Teux Deux**, a pet project she'd designed to meet her own requirements for a to-do list, having found that no existing apps fulfilled her needs. Oh, and it was a collaboration with her studio mate **Cameron**. Remember what I was saying about working with your friends?

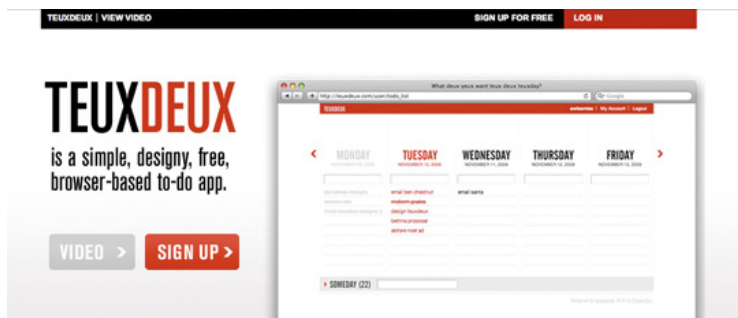


Figure 3: Teux Deux, the GTD pet project that launched just last week

## HAVE A PET PROJECT TO HELP PEOPLE OUT

Ommwriter and Teux Deux are free for anyone to use. Let's just think about that for a moment: the creators have invested their time and effort in the project, and then

given it away to be used by others. That's very cool and something we're used to seeing a lot of in the web community (how lucky we are)! People love free stuff and giving away the fruits of your labour will earn you major kudos. Of course, there's nothing wrong with making some money, either — more on that in a second.



Figure 4: Dan Rubin's extremely helpful Make Photoshop Faster

---

## HAVE A PET PROJECT TO RAISE YOUR PROFILE

So, giving away free stuff earns you kudos. And kudos usually helps you raise your profile in the industry. We all like a bit of shameless fame, don't we? But seriously, if you want to become well known, make something cool. It could be free (to buy you the love and respect of the community) or it could be purchasable (if you've made

something that's cool enough to deserve hard-earned cash), but ultimately it needs to be something that people will love.



Figure 5: Type designer **Jos Buivenga** has shot to fame thanks to his beautiful typefaces and 'freemium' business model

If you're a developer with no design skills, team up with a good designer so that the design community appreciate its aesthetic. If you're a designer with no development skills, team up with a good developer so that it *works*. Oh, and not that I'd recommend you ever do this for selfish

reasons, but collaborating with someone you admire — whose work is well-respected by the community — will also help raise your profile.

## HAVE A PET PROJECT TO MAKE MONEY

In spite of our best hippy-esque intentions to give away free stuff to the masses, there's also nothing wrong with making a bit of money from your pet project. In fact, if your project involves you having to make a considerable financial investment, it's probably a good idea to try and recoup those costs in some way.

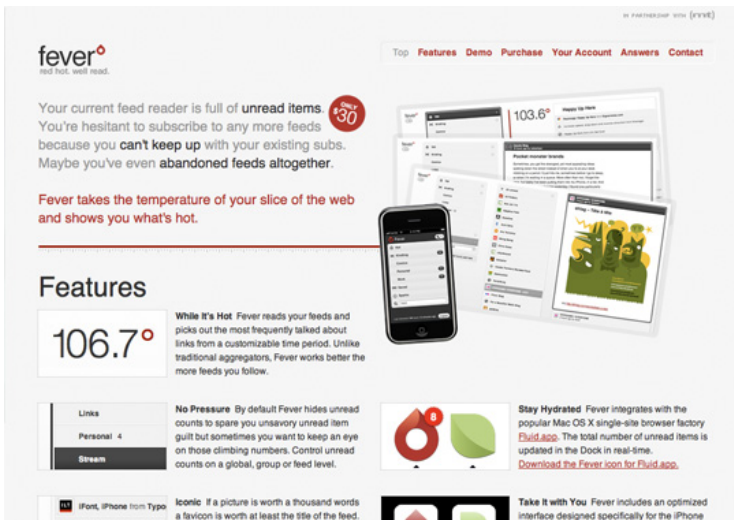


Figure 6: The success of Shaun Inman's various pet projects — Mint, Fever, Horror Vacui, etc. — have allowed him to give up client work entirely.

A very common way to do that in both the online and offline worlds is to get some sort of advertising. For a slightly different approach, try contacting a company who are relevant to your audience and ask them if they'd be interesting in sponsoring your project, which would usually just mean having their brand associated with yours in some way. This is still a form of advertising but tends to allow for a more tasteful implementation, so it's worth pursuing.

Advertising is a great way to cover your own costs and keep things free for your audience, but when costs are considerably higher (like if you're producing a magazine with high production values, for instance), there's nothing wrong with charging people for your product. But, as I mentioned above, you've got to be positive that it's worth paying for!

## **HAVE A PET PROJECT JUST FOR FUN**

Sometimes there's a very good reason for having a pet project — and sometimes even a viable business reason — but actually you don't need any reason at all. Wanting to have fun is just as worthy a motivation, and if you're not going to have fun doing it, then what's the point? Assuming that almost all pet projects are designed, developed, written, printed, marketed and supported in our free time, why not do something enjoyable?



orem ipsum dolor sit amet, consectetur adipiscing elit. Donec mauris lorem, luctus nec imperdiet a, porttitor eget erat. Quisque suscipit congue neque id adipiscing. Vivamus ornare nisl id lacus egestas quis varius dolor vehicula. Cras ac lacus in massa sagittis sollicitudin quis in lorem. Aliquam et diam ac massa convallis tincidunt. Aenean convallis consequat tincidunt. Vestibulum lobortis, nibh at sodales eleifend, nunc lorem sodales arcu, sit amet sodales diam felis ac nisl. In hac habitasse platea dictumst. In arcu ante, adipiscing nec pharetra ut, cursus vitae lorem. Nam eget mauris eget neque sagittis vulputate. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Donec diam enim, pretium iaculis elementum id, faucibus non lacus.

Figure 7: Jessica Hische's beautiful Daily Drop Cap

---

## IN CONCLUSION

The fact that you're reading *24 ways* shows that you have a passion for the web, and that's something I'm happy to see in abundance throughout our community. Passion is a term that's thrown about all over the place, but it really is evident in the work that people do. It's perhaps *most*



evident, however, in the pet projects that people create. Don't forget that the very site you're reading this article on is... a pet project.

If you've yet to do so, make it a new year's resolution for 2010 to have your own pet project so that you can collaborate with your friends, escape from your day job, fulfil your own needs, help people out, raise your profile, make money, and – above all – have fun.

## 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

# 19. Spruce It Up

---

Jonathan Snook

24ways.org/200919

The landscape of web typography is changing quickly these days. We've gone from the wild west days of sIFR to Cufón to finally seeing font embedding seeing wide spread adoption by browser developers (and soon web designers) with `@font-face`. For those who've felt limited by the typographic possibilities before, this has been a good year.

As Mark Boulton has so eloquently elucidated, `@font-face` embedding doesn't come without its drawbacks. Font files can be quite large and FOUT—that nasty flash of unstyled text—can be a distraction for users.

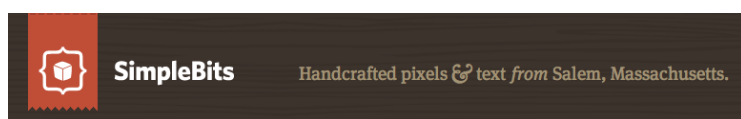
## **DATA URIS**

We can battle FOUT by using Data URIs. A Data URI allows the font to be encoded right into the CSS file. When the font comes with the CSS, the flash of unstyled text is mitigated. No extra HTTP requests are required.

Don't be a grinch, though. Sending hundreds of kilobytes down the pipe still isn't great. Sometimes, all we want to do is spruce up our site with a little typographic sugar.

## BE SELECTIVE

Dan Cederholm's **SimpleBits** is an attractive site.

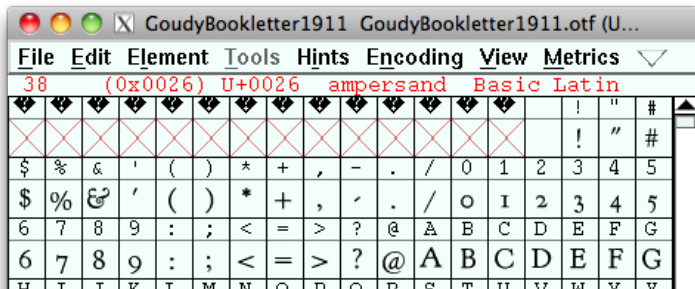


Take a look at the ampersand within the header of his site. It's the lovely (and free) **Goudy Bookletter 1911** available from **The League of Movable Type**. The Opentype format is a respectable 28KB. Nothing too crazy but hold on here. Mr. Cederholm is only using the ampersand! Ouch. That's a lot of bandwidth just for one character.

Can we optimize a font like we can an image? Yes. Image optimization essentially works by removing unnecessary image data such as colour data, hidden comments or using compression algorithms. How do you remove unnecessary information from a font? Subsetting.

If you're the adventurous type, grab a copy of **FontForge**, which is an open source font editing tool. You can open the font, view and edit any of the glyphs and then re-generate the font. The interface is a little clunky but you'll

be able to select any character you don't want and then cut the glyphs. Re-generate your font and you've now got a smaller file.



There are certainly more optimizations that can also be made such as removing hinting and kerning information. Keep in mind that removing this information may affect how well the type renders.

At this time of year, though, I'm sure you're quite busy. Save yourself some time and head on over to the [Font Squirrel Font Generator](#).

## @font-face Kit Generator

+ Add Fonts

Goudy Bookletter 1911 Regular	OTF	303 glyphs	39 KB	✖
-------------------------------	-----	------------	-------	---

**Agreement:**  Yes, the fonts I'm uploading are legally eligible for web embedding.  
Font Squirrel offers this service in good faith. Please honor the EULAs of your fonts.

The Font Generator is extremely handy and allows for a number of optimizations and cross-platform options to be generated instantly. Select the font from your local system—make sure that you are only using properly licensed fonts!

In this particular case, we only want the ampersand. Click on Subset Fonts which will open up a new menu. Unselect any preselected sets and enter the ampersand into the Single Characters text box.

Generate your font and what are you left with? 3KB.

**Subset to**  
**Western Language**  
**Characters:**

Lowercase  
 Uppercase  
 Numbers  
 Punctuation  
 Currency  
 Typographics  
 Math Symbols  
 Alt Punctuation  
 Lower Accents  
 Upper Accents  
 Diacriticals

**Subset to**  
**Unicode Tables:**

Basic Latin  
 Latin-1 Sup  
 Currency Symbols  
 Punctuation  
 Latin Extended-A  
 Latin Extended-B  
 Latin Extended +  
 Greek & Coptic  
 Cyrillic

**Single Characters:** &

**Subset Preview:** &

[Download Your Kit](#)

The Font Generator even generates a base64 encoded data URI stylesheet to be imported easily into your project.

Check out the [Demo page](#). (This demo won't work in Internet Explorer as we're only demonstrating the Data URI font embedding and not using the EOT file format that IE requires.)

## **NO UNNECESSARY ADDITIVES**

If you peeked under the hood of that demo, did you notice something interesting? There's no `<span>` around the ampersand. The great thing about this is that we can take advantage of the font stack's natural ability to switch to a fallback font when a character isn't available.

Just like that, we've managed to spruce up our page with a little typographic sugar without having to put on too much weight.

## ABOUT THE AUTHOR



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

Photo: Patrick H. Lauke



## 20. Cleaner Code with CSS3 Selectors

---

Rachel Andrew

[24ways.org/200920](http://24ways.org/200920)

The parts of CSS3 that seem to grab the most column inches on blogs and in articles are the shiny bits. Rounded corners, text shadow and new ways to achieve CSS layouts are all exciting and bring with them all kinds of possibilities for web design. However what really gets me, as a developer, excited is a bit more mundane.

In this article I'm going to take a look at some of the ways our front *and* back-end code will be simplified by CSS3, by looking at the ways we achieve certain visual effects now in comparison to how we will achieve them in a glorious, CSS3-supported future. I'm also going to demonstrate how we can use these selectors now with a little help from JavaScript – which can work out very useful if you find yourself in a situation where you can't change markup that is being output by some server-side code.

## THE WONDER OF NTH-CHILD

So why does `nth-child` get me so excited? Here is a really common situation, the designer would like the tables in the application to look like this:

Name	Cards sent	Cards received	Cards written but not sent
Ann	40	28	4
Joe	2	27	29
Paul	5	35	2
Louise	65	65	0

Setting every other table row to a different colour is a common way to enhance readability of long rows. The tried and tested way to implement this is by adding a class to every other row. If you are writing the markup for your table by hand this is a bit of a nuisance, and if you stick a row in the middle you have to change the rows the class is applied to. If your markup is generated by your content management system then you need to get the server-side code to add that class – if you have access to that code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Striping every other row - using classes</title>
<style type="text/css">
    body {
```

```

padding: 40px;
margin: 0;
font: 0.9em Arial, Helvetica, sans-serif;
}
table {
border-collapse: collapse;
border: 1px solid #124412;
width: 600px;
}
th {
border: 1px solid #124412;
background-color: #334f33;
color: #fff;
padding: 0.4em;
text-align: left;
}
td {
padding: 0.4em;
}
tr.odd td {
background-color: #86B486;
}
</style>
</head>
<body>
<table>
<tr>
<th>Name</th>
<th>Cards sent</th>
<th>Cards received</th>
<th>Cards written but not sent</th>
</tr>
<tr>
<td>Ann</td>

```

```
<td>40</td>
<td>28</td>
<td>4</td>
</tr>
<tr class="odd">
<td>Joe</td>
<td>2</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>Paul</td>
<td>5</td>
<td>35</td>
<td>2</td>
</tr>
<tr class="odd">
<td>Louise</td>
<td>65</td>
<td>65</td>
<td>0</td>
</tr>
</table>
</body>
</html>
```

## View Example 1

This situation is something I deal with on almost every project, and apart from being an extra thing to do, it just isn't ideal having the server-side code squirt classes into the markup for purely presentational reasons. This is where the `nth-child` pseudo-class selector comes in. The

server-side code creates a valid HTML table for the data, and the CSS then selects the odd rows with the following selector:

```
tr:nth-child(odd) td {  
    background-color: #86B486;  
}
```

### View Example 2

The odd and even keywords are very handy in this situation – however you can also use a multiplier here. 2n would be equivalent to the keyword ‘odd’ 3n would select every third row and so on.

### Browser support

Sadly, nth-child has pretty poor browser support. It is not supported in Internet Explorer 8 and has somewhat buggy support in some other browsers. Firefox 3.5 does have support. In some situations however, you might want to consider using JavaScript to add this support to browsers that don’t have it. This can be very useful if you are dealing with a Content Management System where you have no ability to change the server-side code to add classes into the markup.

I’m going to use jQuery in these examples as it is very simple to use the same CSS selector used in the CSS to target elements with jQuery – however you could use any

library or write your own function to do the same job. In the CSS I have added the original class selector to the `nth-child` selector:

```
tr:nth-child(odd) td, tr.odd td {
    background-color: #86B486;
}
```

Then I am adding some jQuery to add a class to the markup once the document has loaded – using the very same `nth-child` selector that works for browsers that support it.

```
<script src="http://code.jquery.com/
jquery-latest.js"></script>
<script>
    $(document).ready(function(){
        $("tr:nth-child(odd)").addClass("odd");
    });
</script>
```

### View Example 3

We could just add a background colour to the element using jQuery, however I prefer not to mix that information into the JavaScript as if we change the colour on our table rows I would need to remember to change it both in the CSS and in the JavaScript.

## DOING SOMETHING DIFFERENT WITH THE LAST ELEMENT

So here's another thing that we often deal with. You have a list of items all floated left with a right hand margin on each element constrained within a fixed width layout. If each element has the right margin applied the margin on the final element will cause the set to become too wide forcing that last item down to the next row as shown in the below example where I have used a grey border to indicate the fixed width.



Currently we have two ways to deal with this. We can put a negative right margin on the list, the same width as the space between the elements. This means that the extra margin on the final element fills that space and the item doesn't drop down.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>The last item is different</title>
<style type="text/css">
    body {
```

```

padding: 40px;
margin: 0;
font: 0.9em Arial, Helvetica, sans-serif;
}
div#wrapper {
width: 740px;
float: left;
border: 5px solid #ccc;
}
ul.gallery {
margin: 0 -10px 0 0;
padding: 0;
list-style: none;
}
ul.gallery li {
float: left;
width: 240px;
margin: 0 10px 10px 0;
}
</style>
</head>
<body>
<div id="wrapper">
<ul class="gallery">
<li></li>
<li></li>
<li></li>
</ul>
</div>
</body>
</html>

```

**View Example 4**



The other solution will be to put a class on the final element and in the CSS remove the margin for this class.

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

This second solution may not be easy if the content is generated from server-side code that you don't have access to change.

It could all be so different. In CSS3 we have marvellously common-sense selectors such as `last-child`, meaning that we can simply add rules for the last list item.

```
ul.gallery li:last-child {  
  margin-right: 0;  
}
```

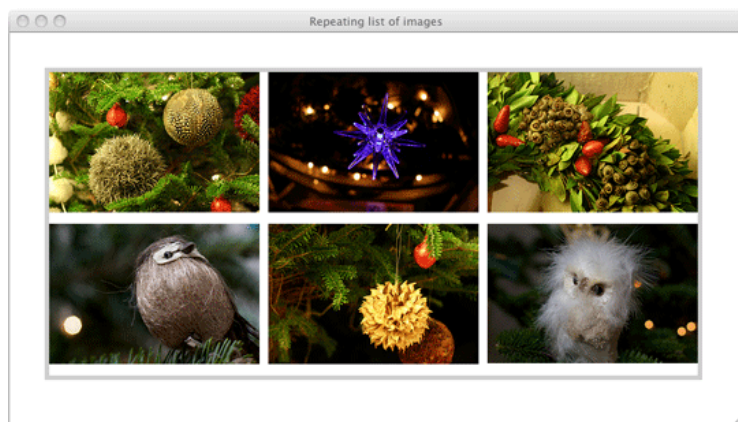
### View Example 5

This removed the margin on the `li` which is the `last-child` of the `ul` with a class of `gallery`. No messing about sticking classes on the last item, or pushing the width of the item out with a negative margin.

If this list of items repeated ad infinitum then you could also use `nth-child` for this task. Creating a rule that makes every 3rd element margin-less.

```
ul.gallery li:nth-child(3n) {  
  margin-right: 0;  
}
```

## View Example 6



A similar example is where the designer has added borders to the bottom of each element – but the last item does not have a border or is in some other way different. Again, only a class added to the last element will save you here if you cannot rely on using the `last-child` selector.

### Browser support for last-child

The situation for `last-child` is similar to that of `nth-child`, in that there is no support in Internet Explorer 8. However, once again it is very simple to replicate the functionality using jQuery. Adding our `.last` class to the last list item.

```
$("#ul.gallery li:last-child").addClass("last");
```

We could also use the `nth-child` selector to add the `.last` class to every third list item.

```
$(“ul.gallery li:nth-child(3n)”).addClass(“last”);
```

[View Example 7](#)

## FUN WITH FORMS

Styling forms can be a bit of a trial, made difficult by the fact that any CSS applied to the `input` element will effect text fields, submit buttons, checkboxes and radio buttons. As developers we are left adding classes to our form fields to differentiate them. In most builds all of my text fields have a simple class of `text` whereas I wouldn't dream of adding a class of `para` to every paragraph element in a document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Styling form fields</title>
<style type="text/css">
    body {
        padding: 40px;
        margin: 0;
        font: 0.9em Arial, Helvetica, sans-serif;
    }
    form div {
        clear: left;
```

```

    padding: 0 0 0.8em 0;
}
form label {
    float: left;
    width: 120px;
}
form .text, form textarea {
    border: 1px solid #333;
    padding: 0.2em;
    width: 400px;
}
form .button {
    border: 1px solid #333;
    background-color: #eee;
    color: #000;
    padding: 0.1em;
}
</style>
</head>
<body>
    <h1>Send your Christmas list to Santa</h1>
    <form method="post" action="" id="christmas-list">
        <div><label for="fName">Name</label>
        <input type="text" name="fName" id="fName"
class="text" /></div>
        <div><label for="fEmail">Email address</label>
        <input type="text" name="fEmail" id="fEmail"
class="text" /></div>
        <div><label for="fList">Your list</label>
        <textarea name="fList" id="fList" rows="10"
cols="30"></textarea></div>
        <div><input type="submit" name="btnSubmit"
id="btnSubmit" value="Submit" class="button" ></div>

```

```
</form>  
</body>  
</html>
```

## View Example 8

Attribute selectors provide a way of targeting elements by looking at the attributes of those elements. Unlike the other examples in this article which are CSS3 selectors, the attribute selector is actually a CSS2.1 selector – it just doesn't get much use because of lack of support in Internet Explorer 6. Using attribute selectors we can write rules for text inputs and form buttons without needing to add any classes to the markup. For example after removing the `text` and `button` classes from my `text` and `submit` button input elements I can use the following rules to target them:

```
form input[type="text"] {  
    border: 1px solid #333;  
    padding: 0.2em;  
    width: 400px;  
}  
form input[type="submit"]{  
    border: 1px solid #333;  
    background-color: #eee;  
    color: #000;  
    padding: 0.1em;  
}
```

## View Example 9

Another problem that I encounter with forms is where I am using CSS to position my labels and form elements by floating the labels. This works fine as long as I want all of my labels to be floated, however sometimes we get a set of radio buttons or a checkbox, and I don't want the label field to be floated. As you can see in the below example the label for the checkbox is squashed up into the space used for the other labels, yet it makes more sense for the checkbox to display after the text.



A screenshot of a form with a vertical line on the left and a horizontal line on the top. The text "Add me to Santa's mailing list" is on the left, and a checkbox is to its right. Below the text is a "Submit" button. The checkbox is positioned such that it overlaps with the text, making it difficult to see.

I could use a class on this label element however CSS3 lets me to target the label attribute directly by looking at the value of the for attribute.

```
label[for="fOptIn"] {  
    float: none;  
    width: auto;  
}
```



A screenshot of a form with a vertical line on the left and a horizontal line on the top. The text "Add me to Santa's mailing list" is on the left, and a checkbox is to its right. Below the text is a "Submit" button. The checkbox is clearly visible and not overlapping with the text.

Being able to precisely target attributes in this way is incredibly useful, and once IE6 is no longer an issue this will really help to clean up our markup and save us from having to create all kinds of special cases when generating this markup on the server-side.

## Browser support

The news for attribute selectors is actually pretty good with Internet Explorer 7+, Firefox 2+ and all other modern browsers all having support. As I have already mentioned this is a CSS2.1 selector and so we really should expect to be able to use it as we head into 2010! Internet Explorer 7 has slightly buggy support and will fail on the label example shown above however I discovered a workaround in the [Sitepoint CSS reference comments](#). Adding the selector `label[htmlFor="f0ptIn"]` to the correct selector will create a match for IE7.

IE6 does not support these selector but, once again, you can use jQuery to plug the holes in IE6 support. The following jQuery will add the `text` and `button` classes to your fields and also add a `checks` class to the label for the checkbox, which you can use to remove the float and width for this element.

```
$('#form input[type="submit"]').addClass("button");  
$('#form input[type="text"]').addClass("text");  
$('#label[for="f0ptIn"]').addClass("checks");
```

## View Example 10

The selectors I've used in this article are easy to overlook as we do have ways to achieve these things currently. As developers – especially when we have frameworks and existing code that cope with these situations – it is easy to carry on as we always have done.

I think that the time has come to start to clean up our front and backend code and replace our reliance on classes with these more advanced selectors. With the help of a little JavaScript almost all users will still get the full effect and, where we are dealing with purely visual effects, there is definitely a case to be made for not worrying about the very small percentage of people with old browsers and no JavaScript. They will still receive a readable website, it may just be missing some of the finesse offered to the modern browsing experience.



## ABOUT THE AUTHOR



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

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

# 21. Make Out Like a Bandit

---

Jina Bolton

24ways.org/200921

If you are anything like me, you are a professional juggler. No, we don't juggle bowling pins or anything like that (or *do* you? Hey, that's pretty rad!). I'm talking about the work that we juggle daily. In my case, I'm a full-time designer, a half-time graduate student, a sometimes author and conference speaker, and an all-the-time social networker. Only two of these "positions" have actually put any money in my pocket (and, well, the second one takes a lot of money *out*). Still, this is all part of the work that I do. Your work situation is probably similar. We are workaholics.

So if we work so much in our daily lives, shouldn't we be making out like bandits? Umm, honestly, I'm not hitting on you, silly. I'm talking about our *success*. We work and work and work. Shouldn't we be filthy, stinking rich? Well... okay, that's not quite what I mean either. I'm not necessarily talking about money (though that could

potentially be a part of it). I'm talking about success — as in feeling a true sense of accomplishment and feeling happy about what we do and why we do it.

It's important to feel accomplished and a general happiness in our work. To make out like a bandit (or have an incredible amount of success), you can either get lucky or work hard for it. And if you're going to work hard for it, you might as well make it all meaningful and worthwhile. This is what I strive for in my own work and my life, and the following points I'm sharing with you are the steps I am taking to work toward this.

I know the price of success: dedication, hard work & an unremitting devotion to the things you want to see happen. — *Frank Lloyd Wright*

## **LEARN. PARTICIPATE. DO.**

The best way to get good at something is to keep doing whatever it is you're doing that you want to be good at. For example, a sushi-enthusiast might take a sushi-making class because she wants to learn to make sushi for herself. It totally makes sense while the teacher demonstrates all the procedures, materials, and methods needed to make good, beautiful sushi. Later, the student goes home and tries to make sushi on her own, she gets totally confused and lost. Okay, I'm not even going to hide it, I'm talking

about myself (this happened to me). As much as I love sushi, I couldn't even begin to make good sushi because I've never really practiced.

Take advantage of learning opportunities where possible. Whether you're learning CSS, Actionscript, or visual design, the best way to grasp how to do things is to participate, practice, do. Apply what you learn in your work. Participation is so vital to your success. If you have problems, let people know, and ask. But definitely practice on your own. And as cliché as it may sound, believe in yourself because if you don't think you can do it, no one else will think you can either.

## **MAINTAIN MOMENTUM**

With whatever it is you're doing, if you find yourself "on a roll", you should take advantage of that momentum and keep moving. Sure, you'll definitely want to take breaks here or there, but remember that momentum can be very difficult to obtain again once you've lost it. Get it done!

## **DEAL WITH PEOPLE**

Whether you love or hate people, the fact is, you gotta deal with them – even the difficult ones. If you're in a management position, then you know pretty well that most people don't like being told what to do (even if that's their job). Find ways to get people excited about what

they're doing. Make people feel that they (and what they do) are needed — people respond better if they're valued, not commanded. Even if you're not in a management position, this still applies to the way you work with your coworkers, clients, vendors, etc.

Resolve any conflicts right away. Conflicts will inevitably happen. Move on to how you can improve the situation, and do it as quickly as possible. Don't spend too much time focusing on whose screw up it is — nobody feels good in this situation. Also, try to keep people informed on whatever it is you need or what it is you're doing. If you're waiting on something from someone, and it's been a while, don't be afraid to say something (tactfully). Sometimes people are forgetful — or just slacking. Hey, it happens!

## **HELP YOURSELF BY HELPING OTHERS**

What are some of the small, simple things you can do when you're working that will help the people you work with (and in most cases, will end up helping yourself)? For example: if you're a designer, perhaps taking a couple minutes now to organize and name your Photoshop layers will end up saving time later (since it will be easier to find things). This is going to help both you and your team. Or, developers: taking some time to write some documentation (even if it's as simple as a comment in the code, or a well-written commit message) could potentially save valuable time for both you and your team later.

Maybe you have to take a little time to sit down with a coworker and explain why something works the way it does. This helps them out tremendously – and will most likely lead to them respecting you a little more. This is a benefit.

If you make little things like this a habit, people *will* notice. People will enjoy working with you. People will trust you and rely on you. Sure, it might seem beneficial at any given moment to be “in it for yourself” (and therefore only helping yourself), but that won’t last very long. Helping others (whether it be a small or large feat) will cause a positive impact in the long run – and that is what will be more valuable to you and your career.

## **DO WORK THAT IS MEANINGFUL**

One of the best ways to feel successful about what you do is to feel good and happy about it. And a great way to feel good and happy about what you’re doing is to actually do good. This could be purpose-driven work that focuses on **sustainability** and **environmentalism**, or work that helps support **causes** and **charity**. Perhaps the work simply **inspires people**. Or maybe the work is just something you are very passionate about. Whatever the work may be, try working on projects that are meaningful to you. You’ll do well simply by being more motivated and interested. And it’s a double-win if the project is meaningful to others as well.

I feel very fortunate to work at a place like **Crush + Lovely**, where we have found quite frequently that the projects that inspire people, focus on global and social good, and create some sort of positive impact are the very projects that bring us more paid projects. But more importantly, we are happy and excited to do it. You might not work at a company that takes on those types of projects. But perhaps you have your own personal endeavors that create this excitement for you. Elliot Jay Stocks wrote about **having pet projects**. Do you take on side projects? What are those projects?

Over the last couple years, I've seen some really fantastic side projects come out that are great examples of meaningful work. These projects reflect the passions and goals of the respective designers and developers involved, and therefore become quite successful (because the people involved simply love what they are doing while they're doing it). Some of these projects include:

- **Typedia** is a shared encyclopedia of typefaces which serves as a resource to classify, categorize, and connect typefaces. It was founded by **Jason Santa Maria**, a graphic designer with a love and passion for typography. He created it as a solution to a problem he faced as a designer: finding the right typeface.

- **Huffduffer** was created by **Jeremy Keith**, a web developer who wanted to create a podcast of inspirational talks — but after he found that this could be tedious, he decided to create a tool to automate this.
- **Level & Tap** was created by passionate photographer and web developer, **Tom Watson**. It began as a photography print store for Tom's best personal photography. Over time, more photographers were added to the site and the site has grown to become quite a great collection of beautiful photography.
- **Heat Eat Review** is a review blog created by information architect and user experience designer, **Abi Jones**. As a foodie, she is able to use this passion for this blog, as it focuses on reviewing TV Dinners, Frozen Meals, and Microwavable Foods.
- **Art in My Coffee**, a favorite personal project of my own, is a photo blog of coffee art I created, after I found that my friends and I were frequently posting coffee art photos to Flickr, Twitter, and other websites. After the blog became more popular, I teamed up with **Meagan Fisher** on the project, who has just as much a passion for coffee art, if not more.

## **SO, WHAT'S IMPORTANT TO YOU?**

This is the very, very important question here. What really matters to you most? Beyond just working on meaningful projects you are passionate about, is the work you're



doing the right work for you, so that you can live a good lifestyle? Scott Boms wrote an excellent article, **Burnout**, in which he shares his own experience in battling stress and exhaustion, and what he learned from it. You should definitely read the article in its entirety, but a couple of his points that are particularly excellent are:

- **Make time for numero uno**, in which you make time for the things in life that make you happy
- **Examine your values, goals, and measures of success**, in which you work toward the things you are passionate about, your own personal development, and focusing on the things that matter.

A solid work-life balance can be a challenging struggle to obtain. Of course, you can cheat this by finding ways to combine the things you love with the things you do (so then it doesn't even feel like you're *working* — oh, you sneaky little bandit!). However, there are other factors to consider beyond your general love for the work you're doing. Take proper care of yourself physically, mentally, and socially.

## **SO, ARE YOU MAKING OUT LIKE A BANDIT?**

Do you feel accomplished and generally happy with your work? If not, perhaps that is something to focus on for the next year. Consider your work (both in your job as well as

any side projects you may take on) and how it benefits you – present and future. Take any steps necessary to get you to where you need to be. If you are miserable, fix it!

Finally, it's important to be thankful for the things that matter to you and make you happy. Pass it along everyday. Thank people. It's a simple thing, really. Saying "thank you" can and will have enormous impact on the people around you. Oh. And, I apologize if the title of this article led you to thinking it would teach you how to be an amazing kisser. That's a different article entirely for *24 ways to impress your friends!*

## ABOUT THE AUTHOR



Jina Bolton is a Senior Product Designer at Salesforce UX, where she helps design and develop systems for enterprise software. She also loves Sass; she leads **Team Sass Design**, an open source task force that redesigned the Sass brand and website. Jina also organizes the San Francisco Sass Meet Up, **The Mixin**. She coauthored two books, *Fancy Form Design* and *The Art & Science of CSS*. Previously, she has worked with rad companies including Apple, Engine Yard, and Crush + Lovely.

Photo: Nick Howland

## 22. Real Fonts and Rendering: The New Elephant in the Room

---

Jeffrey Zeldman

24ways.org/200922

My friend, the content strategist Kristina Halvorson, likes to call content “the elephant in the room” of web design. She means it’s the huge problem that no one on the web development team or client side is willing to acknowledge, face squarely, and plan for.

A typical web project will pass through many helpful phases of research, and numerous beneficial user experience design iterations, while the content—which in most cases is supposed to be the site’s primary focus—gets handled haphazardly at the end. Hence, elephant in the room, and hence also artist Kevin Cornell’s recent use of *elephantine imagery* to illustrate *A List Apart* articles on the subject. But I digress.

Without discounting the primacy of the content problem, we web design folk have now birthed ourselves a second lumbering mammoth, thanks to our interest in “real fonts

on the web“ (the unfortunate name we’ve chosen for the recent practice of serving web-licensed fonts via CSS’s decade-old `@font-face` declaration—as if Georgia, Verdana, and Times were somehow unreal).

For the fact is, even `bulletproof` and `mo’ bulletproofer` `@font-face` CSS syntax aren’t *really* bulletproof if we care about looks and legibility across browsers and platforms.

## **HYENAS IN THE BREAKFAST NOOK**

The problem isn’t just that foundries have yet to agree on a standard font format that protects their intellectual property. And that, even when they do, it will be a while before all browsers support that standard—leaving aside the inevitable politics that impede all standardization efforts. Those are problems, but they’re not the elephant. Call them the coyotes in the room, and they’re slowly being tamed.

Nor is the problem that workable, scalable business models (of which `Typekit`’s is the most visible and, so far, the most successful) are still being shaken out and tested. The quality and ease of use of such services, their stability on heavily visited sites (via massively backed-up server clusters), and the fairness and sustainability of their pricing will determine how licensing and serving “real fonts” works in the short and long term for the majority of designer/developers.

Nor is our primary problem that developers with no design background may serve ugly or illegible fonts that take forever to load, or fonts that take a long time to download and then display as ordinary system fonts (as happens on, say, [about.validator.nu](http://about.validator.nu)). Ugliness and poor optimization on the web are nothing new. That support for @font-face in Webkit and Mozilla browsers (and for TrueType fonts converted to Embedded OpenType in Internet Explorer) adds deadly weapons to the non-designer's toolkit is not the technology's fault. JavaScript and other essential web technologies are equally susceptible to abuse.

## **BEAUTY IS IN THE EYE OF THE RENDERING ENGINE**

No, the real elephant in the room—the thing few web developers and no “web font” enthusiasts are talking about—has to do with legibility (or lack thereof) and aesthetics (or lack thereof) across browsers and platforms. Put simply, even fonts optimized for web use (which is a whole thing: ask a type designer) will not look good in every browser and OS. That's because every browser treats hinting differently, as does every OS, and every OS version.

Firefox does its own thing in both Windows and Mac OS, and Microsoft is all over the place because of its need to support multiple generations of Windows and ClearType

and all kinds of hardware simultaneously. Thus “real type” on a single web page can look markedly different, and sometimes very bad, on different computers at the same company. If that web page is your company’s, your opinion of “web fonts” may suffer, and rightfully. (The advantage of Apple’s closed model, which not everyone likes, is that it allows the company to guarantee the quality and consistency of user experience.)

As near as my font designer friends and I can make out, Apple’s Webkit in Safari and iPhone ignores hinting and creates its own, which Apple thinks is better, and which many web designers think of as “what real type looks like.” The forked version of Webkit in Chrome, Android, and Palm Pre also creates its own hinting, which is close to iPhone’s—close enough that Apple, Palm, and Google could propose it as a standard for use in all browsers and platforms. Whether Firefox would embrace a theoretical Apple and Google standard is open to conjecture, and I somehow have difficulty imagining Microsoft buying in—even though they know the web is more and more mobile, and that means more and more of their customers are viewing web content in some version of Webkit.

## THE END OF SIMPLE

There are ways around this ugly type ugliness, but they involve complicated scripting and sniffing—the very nightmares from which web standards and the simplicity

of @font-face were supposed to save us. I don't know that even mighty Typekit has figured out every needed variation yet (although, working with foundries, they probably will).

For type foundries, the complexity and expense of rethinking classic typefaces to survive in these hostile environments may further delay widespread adoption of web fonts and the resolution of licensing and formatting issues. The complexity may also force designers (even those who prefer to own) to rely on a hosted rental model simply to outsource and stay current with the detection and programming required.

Forgive my tears. I stand in a potter's field of ideas like "Keep it simple," by a grave whose headstone reads "Write once, publish everywhere."



## 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

## 23. Ignorance Is Bliss

---

Andrew Clarke

24ways.org/200923

This is a true story.

### **MEET MIKE**

Mike's a smart guy. He knows a great browser when he sees one. He uses Firefox on his Windows PC at work and Safari on his Mac at home. Mike asked us to design a Web site for his business. So we did.

We wanted to make the best Web site for Mike that we could, so we used all of the CSS tools that are available today. That meant using RGBA colour to layer elements, border-radius to add subtle rounded corners and (possibly most experimental of all new CSS), generated gradients.

It's Hardboiled

WHY HARDBOILED? Think you're hard enough?

HARDBOILED ARCHIVES Stories submitted by readers!

HARDBOILED AUTHORS Ready for some action!


CLASSIC HARDBOILED Load up in our bookstore!

## "It's Hardboiled"

Find a story

### Hardest boiled

#### Contract Killer by Andy Clarke



Sometimes you let the machine get to you, sometimes you don't. Sometimes the only way to get through the day is to blow with a stray anger that fills everyone within earshot that they should stay the hell out of your way. Today was one of those days.

I went as far as the front door and fired up a smoke, blowing deep. The cold winter air on my face and the feeling of smoke filling my lungs made me feel human again.

That was as far as I could get. The reflection that stared back at me in the window told me that I looked like hell. I crossed back to it's face, took another deep drag and flicked the butt outside.

From behind me Letty said, "M...", but she knew better than anyone not to say any more. There was no need. I knew what had to be done.


My gun went a light little bell under my belt because Letty wasn't the only one who had remembered. What I had to do had been a fire in my gut. It was a fire that had been burning me up from the inside and there wasn't a Goddamn thing I could do about it. I twisted my face into hard looks and...

Get a story to tell? Get a story? Don't hold back or be a pussy. Let it out and share your hardboiled with the world.

Submit your hardboiled story


### Latest Hardboiled

#### The Ultimate Package by Simon Collison




I shook the rain from my coat and threw it towards the ceiling as I walked into the room. Nobody said a word, but I could feel their eyes fixed on me. After seconds like this the lesson, Phil remembered hard and said slowly...

#### Pumpkin Soup by Owen Gregory



From the first moment I saw her I knew the damn was trouble. She was a beauty, a goddess who could have any man who could walk. Behind those deep brown eyes I knew a fire was burning, that only I could put out...


#### The Big Break by Gregory Wood



It was one of those nights when the city came down and conformed for me. The wind howled and the cold rain stung my face as I stumbled. Blind drunk and angled towards the cliff edge...

### Hardboiled classics


#### The Mike Hammer Omnibus by Mickey Spillane



"There's a kind of power about Mickey Spillane that no other writer can imitate" — New York Times

Add to basket

#### The Mike Hammer Omnibus Volume 2 by Mickey Spillane



"The sixth century's bestselling novelist" — Guardian

Add to basket

"It's Hardboiled"

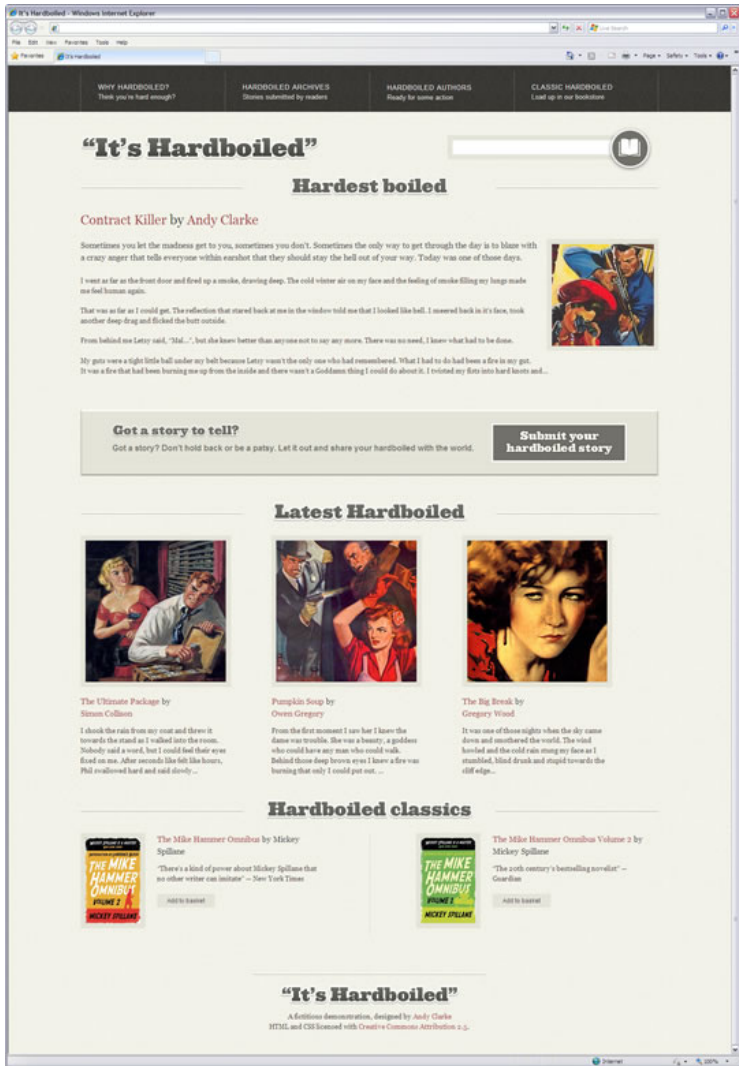
A 50th Anniversary, designed by Andy Clarke  
HTML and CSS licensed with Creative Commons Attribution 4.0.

The home page Mike sees in Safari on his Mac

Mike loves what he sees.

## **MEET SAM**

Sam works with Mike. She uses Internet Explorer 7 because it came on the Windows laptop that the company bought her when she joined.



The home page Sam sees in Internet Explorer 7 on her PC

Sam loves the new Web site too.

How could both of them be happy when they experienced the Web site differently?

### **The new WYSIWYG**

When I first presented my designs to Mike and Sam, I showed them a Web page made with HTML and CSS in their respective browsers and not a picture of a Web page. By showing neither a static image of my design, I set *none* of the false expectations that, by definition, a static Photoshop or Fireworks visual would have established.

Mike saw rounded corners and subtle shadows in Firefox and Safari. Sam saw something equally as nice, just a little different, in Internet Explorer. Both were very happy because they saw something that they liked.

Neither knew, or needed to know, about the subtle differences between browsers. Their users don't need to know either.

That's because in the real world, people using the Web don't find a Web site that they like, then open up another browser to check that it looks they same. They simply buy what they came to buy, read what what they came to read, do what they came to do, then get on with their lives in blissful ignorance of what they might be seeing in another browser.

Often when I talk or write about using progressive CSS, people ask me, “How do you convince clients to let you work that way? What’s your secret?” Secret? I tell them what they need to know, on a need-to-know basis.

## EPILOGUE

Sam has a new iPhone that Mike bought for her as a reward for achieving her sales targets. She loves her iPhone and was surprised at just how fast and good-looking the company Web site appears on that. So she asked,

“Andy, I didn’t know you optimised our site for mobile. I don’t remember seeing an invoice for that.”

I smiled.

“That one was on the house.”



## 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. Make Your Mockup in Markup

---

Meagan Fisher

24ways.org/200924

We aren't designing copies of web pages, we're designing web pages.

*Andy Clarke, via Quotes on Design*

## The old way

I used to think the best place to design a website was in an image editor. I'd create a pixel-perfect PSD filled with generic content, send it off to the client, go through several rounds of revisions, and eventually create the markup.

Does this process sound familiar? You're not alone. In a very scientific and official survey I conducted, close to 90% of respondents said they design in Photoshop before the browser.

## That process is whack, yo!

Recently, thanks in large part to the influence of **design hero Dan Cederholm**, I've come to the conclusion that a website's design should begin where it's going to live: in the browser.

## DIE PHOTOSHOP, DIE

Some of you may be wondering, "what's so bad about using Photoshop for the bulk of my design?" Well, any seasoned designer will tell you that working in Photoshop is akin to working in a minefield: you never know when it's going to blow up in your face.



The application Adobe Photoshop CS4 has unexpectedly ruined your day.

---

Photoshop's propensity to crash at crucial moments is a **running joke** in the industry, as is its **barely usable interface**. And don't even get me started on the hot, steaming pile of crap that is **text rendering**.

One morning, when Gregor Samsa  
woke from troubled dreams, he found  
himself transformed in his bed into a  
horrible vermin.

One morning, when Gregor Samsa  
woke from troubled dreams, he found  
himself transformed in his bed into a  
horrible vermin.

Text rendered in Photoshop (left) versus Safari (right).

---

Crashing and text rendering issues suck, but we've learned to live with them. The real issue with using Photoshop for mockups is the expectations you're setting for a client. When you send the client a static image of the design, you're not giving them the whole picture — they can't see how a **fluid grid** would function, how the design will look in a variety of browsers, basic interactions like :hover effects, or JavaScript behaviors. For more on the disadvantages to showing clients designs as images rather than websites, check out Andy Clarke's **Time to stop showing clients static design visuals**.

### **A necessary evil?**

In the past we've put up with Photoshop because it was vital to achieving our beloved rounded corners, drop shadows, outer glows, and gradients. However, with the recent adaptation of CSS3 in major browsers, and the slow, joyous death of IE6, browsers can render mockups that are just as beautiful as those created in an image editor. With the power of RGBA, text-shadow, box-shadow, border-radius, transparent PNGs, and @font-

face combined, you can create a prototype that radiates shiny awesomeness right in the browser. If you can see this epic article through to the end, I'll show you step by step how to create a gorgeous mockup using mostly markup.

## GET STARTED BY GETTING NAKED

Content precedes design. Design in the absence of content is not design, it's decoration.

*Jeffrey Zeldman*

In the beginning, don't even think about style. Instead, start with the foundation: the content. Lay the groundwork for your markup order, and ensure that your design will be useable with styles and images turned off. This is great for prioritizing the content, and puts you on the right path for accessibility and search engine optimization. Not a bad place to start, amirite?

# Planes & Trains

- [Home](#)
- [About Mike](#)
- [Photographs](#)
- [Notebook](#)
- [Collection](#)
- [Subscribe](#)
- [Contact](#)

---

## What is this?

This is where Mike Fisher writes about his experiences working on the railroad.

An example of unstyled content, in all its naked glory. [View it large](#).

---

## FLUSH OUT THE LAYOUT

The next step is to structure the content in a usable way. With CSS, making basic layout changes is as easy as switching up a `float`, so experiment to see what structure suits the content best.

## Planes & Trains

- [Home](#)
- [About Mike](#)
- [Photographs](#)
- [Notebook](#)
- [Collection](#)
- [Subscribe](#)
- [Contact](#)

### From the Notebook

Thoughts on being a railroad employee and aficionado. Use the search field below to find a specific article, or [browse the archives](#).

Search:

#### ONEONTA'S ROUNDHOUSE

Posted [December 6, 2009](#)

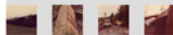
A roundhouse is a building used by railroads for servicing locomotives. Roundhouses are large, circular or semicircular structures that were traditionally located surrounding or adjacent to turntables. The defining feature of the traditional roundhouse was the turntable, which facilitates access when the building is used for repair facilities or for storage of steam locomotives. [Continue Reading...](#)

[21 comments](#) | [Post a comment](#)

### What is this?

This is where Mike Fisher writes about his experiences working on the railroad. He is a collector of railroad memorabilia, guitars, and any other antiques of interests

### Recent Photographs



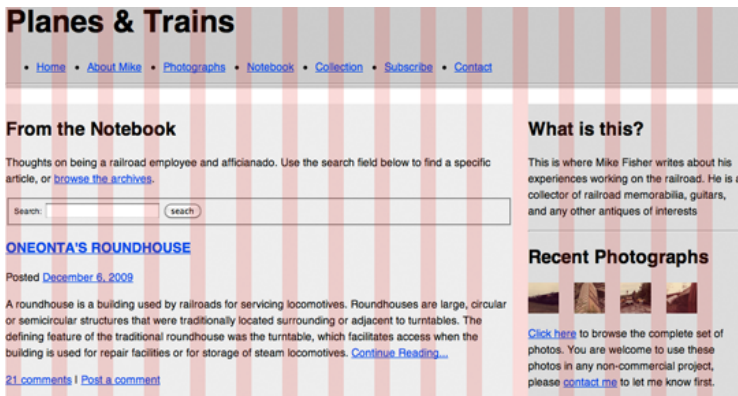
[Click here](#) to browse the complete set of photos. You are welcome to use these photos in any non-commercial project, please [contact me](#) to let me know first.

The mockup with basic layout work done.

---

## Got your grids covered

There are a variety of tools that allow you to layer a grid over your browser window. For Mac users I recommend using **Slammer**, and PC users can check out one of the bookmarklets that are available, such as **960 Gridder**.



The mockup with a grid applied using **Slammer**.

---

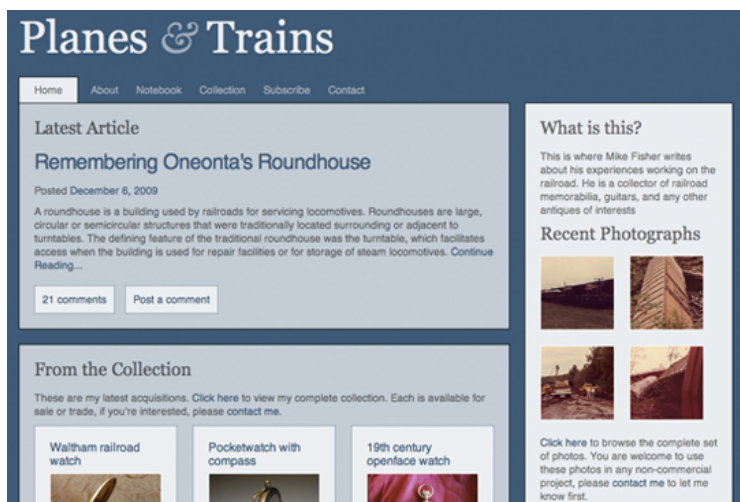
Once you've found a layout that works well for the content, pass it along to the client for review. This keeps them involved in the design process, and gives them an idea of how the site will be structured when it's live.

## START YOUR STYLING

Now for the fun part: begin applying the presentation layer. Let usability considerations drive your decisions about color and typography; use highlighted colors and contrasting typefaces on elements you wish to emphasize.

### RGBA? More like RGBay!

Introducing color is easy with RGBA. I like to start with the page's main color, then use white at varying opacities to emphasize content sections.



In the example mockup the body background is set to `rgba(203, 111, 21)`, the content containers are set to `rgba(255, 255, 255, 0.7)`, and a few elements are highlighted with `rgba(255, 255, 255, 0.1)`. If you're not sure how RGBA works, check out Drew McLellan's super helpful [24ways](#) article.



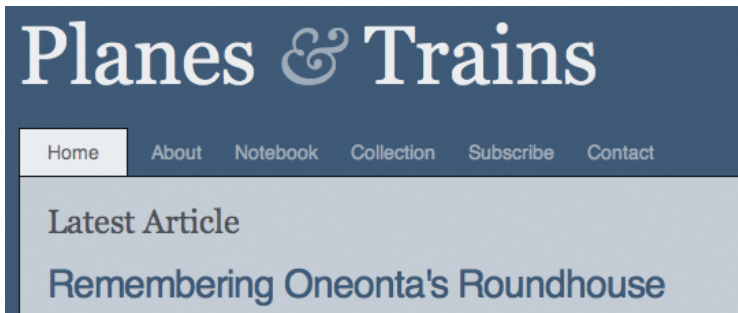
## Laying down type

Just like with color, you can use typography to evoke a feeling and direct a user's attention. Have contrasting typefaces (like serif headlines and sans-serif body text) to group the content into meaningful sections.

In a recent **A List Apart** article, the Master of Web Typography™ Jason Santa Maria offers excellent advice on how to choose your typefaces:

Write down a general description of the qualities of the message you are trying to convey, and then look for typefaces that embody those qualities.

Sounds pretty straightforward. I wanted to give my design a classic feel with a hint of nostalgia, so I used Georgia for the headlines, and incorporated the ornate ampersand from Baskerville into the header.



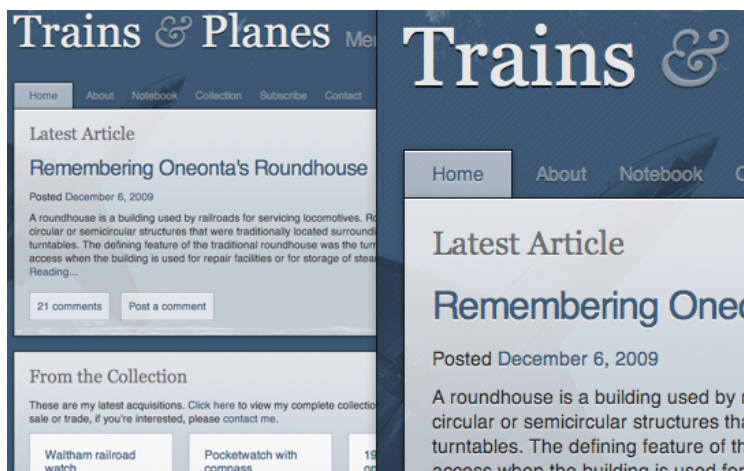
A closeup on the site's header.

## LET'S GET SEXY

The design doesn't look *too* bad as it is, but it's still pretty flat. We can do better, and after mixing in some CSS3 and a couple of PNGs, it's going to get downright steamy in here.

### Give it some glow

Objects in the natural world reflect light, so to make your design feel tangible and organic, give it some glow. In the example design I achieved this by creating two white to transparent gradients of varying opacities. Both have a solid white border across their top, which gives edges a double border effect and makes them look sharper. Using CSS3's `text-shadow` on headlines and `box-shadow` on content modules is another quick way to add depth.

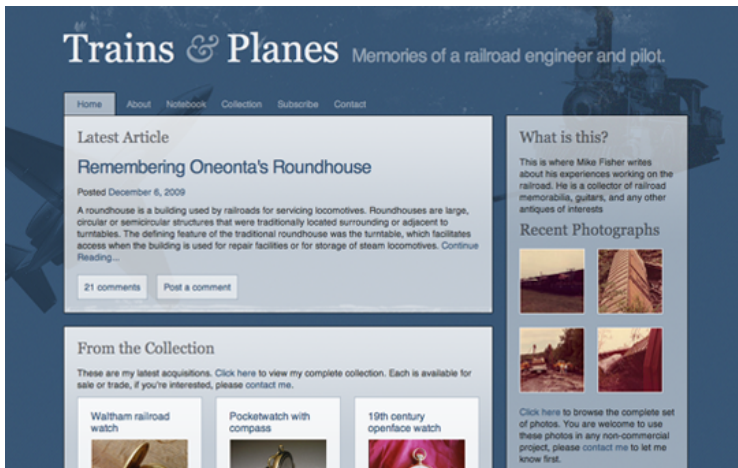


A wide and closeup view of the design with gradients, text-shadow and box-shadow added. For information on how to implement text-shadow and box-shadow using RGBA, check out the article I wrote on it last week.

---

## 37 pieces of flair

Okay, maybe you don't need that much **flair**, but it couldn't hurt to add a little; it's the details that will set your design apart. Work in imagery and texture, using PNGs with an alpha channel so you can layer images and still tweak the color later on.



The design with grungy textures, a noisy diagonal stripe pattern, and some old transportation images layered behind the text. Because the colors are rendered using RGBA, these images bleed through the content, giving the design a layered feel. **Best viewed large.**

---

## SEND IT OFF

Hey, look at that. You've got a detailed, well structured mockup for the client to review. Best of all, your markup is complete too. If the client approves the design at this stage, your template is practically finished. Bust out the party hats!

## NOT SO FAST, BUSTER!

So I don't know about you, but I've never gotten a design past the client's keen eye for criticism on the first go. Let's review some hypothetical feedback (none of which is too outlandish, in my experience), and see how we'd make the requested changes in the browser.

### Updating the typography

My ex-girlfriend loved Georgia, so I never want to see it again. Can we get rid of it? I want to use a font that's chunky and loud, just like my stupid ex-girlfriend.

*Fakey McClient*

Yikes! Thankfully with CSS, removing Georgia is as easy as running a find and replace on the stylesheet. In my revised mockup, I used @font-face and **League Gothic** on the headlines to give the typography the, um, unique feel the client is looking for.



The same mockup, using @font-face on the headlines. If you're unfamiliar with implementing @font-face, check out [Nice Web Type's](#) helpful article.

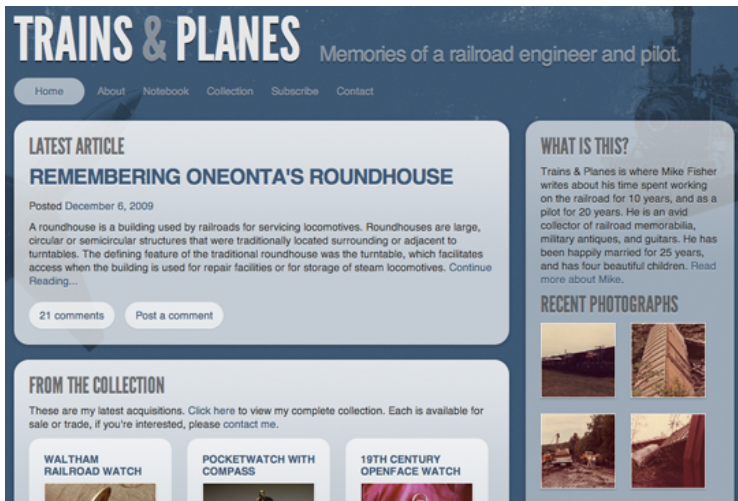
---

## Adding rounded corners

I'm not sure if I'll like it, but I want to see what it'd look like with rounded corners. My cousin, a Web 2.0 marketing guru, says they're trendy right now.

*Fakey McClient*

Switching to rounded corners is a nightmare if you're doing your mockup in Photoshop, since it means recreating most of the shapes and UI elements in the design. Thankfully, with CSS `border-radius` comes to our rescue! By applying this gem of a style to a handful of classes, you'll be rounded out in no time.



The mockup with rounded corners, created using border-radius. If you're not sure how to implement border-radius, check out [CSS3.info's](#) quick how-to.

---

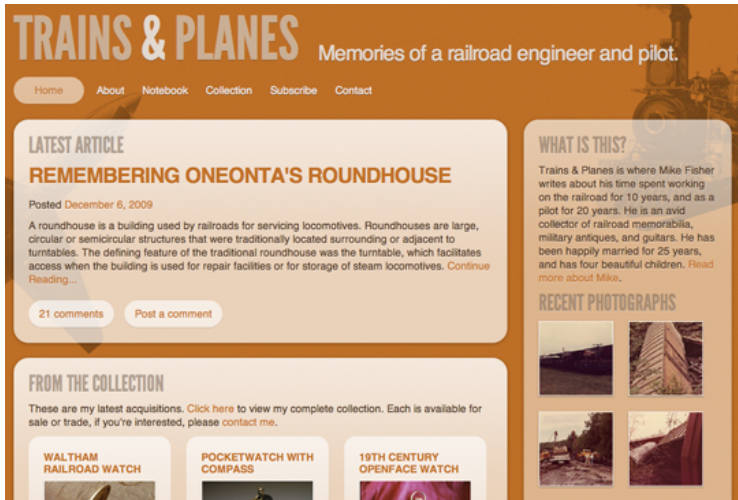
## Making changes to the color

The design is too dark, it's depressing! They call it 'the blues' for a reason, dummy. Can you try using a brighter color? I want orange, like Zeldman uses.

*Fakey McClient*

Making color changes is another groan-inducing task when working in Photoshop. Finding and updating every background layer, every drop shadow, and every link can take forever in a complex PSD. However, if you've done

your mockup in markup with RGBA and semi-transparent PNGs, making changes to your color is as easy as updating the body background and a few font colors.



The mockup with an orange color scheme. Best viewed large.

## AHEM, WHAT ABOUT INTERNET EXPLORER?

Gee, thanks for reminding me, buzzkill. Several of the CSS features I've suggested you use, such as RGBA, text-shadow and box-shadow, and border-radius, are not supported in Internet Explorer. I know, it makes me sad too. However, this doesn't mean you can't try these techniques out in your markup based mockups. The point

here is to get your mockups done as efficiently as possible, and to keep the emphasis on markup from the very beginning.

Once the design is approved, you and the client have to decide if you can live with the design looking different in different browsers. Is it so bad if some users get to see drop shadows and some don't? Or if the rounded corners are missing for a portion of your audience? The design won't be broken for IE people, they're just missing out on a few visual treats that other users will see.

The idea of rewarding users who choose modern browsers is not a new concept; Dan covers it thoroughly in *Handcrafted CSS*, and it's been written about in the past by [Aaron Gustafson](#) and [Andy Clarke](#) on several occasions. I believe we shouldn't have to design for the lowest common denominator (cough, IE6 users, cough); instead we should create designs that are beautiful in modern browsers, but still degrade nicely for the other guy. However, some clients just aren't that progressive, and in that case you can always use background images for drop shadows and rounded corners, as you have in the past.



## CLOSING THOUGHTS

With the advent of CSS3, browsers are just as capable of giving us beautiful, detailed mockups as Photoshop, and in half the time. I'm not the only one to make an argument for this revised process; in his article **Time to stop showing clients static design visuals**, and in his presentation **Walls Come Tumbling Down**, Andy Clarke makes a fantastic case for creating your mockups in markup.

So I guess my challenge to you for 2010 is to get out of Photoshop and into the code. Even if the arguments for designing in the browser aren't enough to make you change your process permanently, it's worthwhile to give it a try. Look at the New Year as a time to experiment; applying constraints and evaluating old processes can do wonders for improving your efficiency and creativity.

## ABOUT THE AUTHOR



**Meagan Fisher** is passionate about owls, coffee, and web design. In her ongoing mission to make the web a better place, she's partnered with some of the best designers in the industry, such as **SimpleBits**, **Happy Cog**, and **Crush + Lovely**. When she's not creating interfaces, she's speaking, **tweeting**, writing on **Owltastic**, or posting coffee art photography to **Art in my Coffee**.