

Practical JavaScript Techniques

A large, stylized green letter 'J' that is partially cut off at the bottom right corner of the image.

J



Practical JavaScript Techniques



Imprint

© 2013 Smashing Media GmbH, Freiburg, Germany

ISBN: 978-3-943075-59-5 (Version 1: February 2013)

Cover Design: Ricardo Gimenes.

PR & Press: Stephan Poppe.

eBook Strategy and Editing: Vitaly Friedman.

Technical Editing: Cosima Mielke.

Planning and Quality Control: Vitaly Friedman, Iris Lješnjanić.

Tools: Elja Friedman. Syntax Highlighting: Prism by Lea Verou.

Copywriter: Lucy Leiderman.

Idea & Concept: Smashing Media GmbH.

About This Book

Present across millions of websites and growing in popularity, JavaScript is an essential and practical for all programmers, designers and coding aficionados. *Practical JavaScript Techniques* offers expert instruction, tips and methodologies relevant for all levels of knowledge. Learn interactive CSS and jQuery techniques, how to take advantage of JavaScript's versatile capabilities and even how to build JavaScript-based gaming experiences. Whether you're perfecting Web design or building jQuery plugins, this vital resource is a must-have.

TABLE OF CONTENTS

Imprint

Develop A One-Of-A-Kind CSS/JS-Based Game Portfolio

by Daniel Sternlicht

Five Useful Interactive CSS/jQuery Techniques Deconstructed

by Jon Raasch

Create An Animated Bar Graph With HTML, CSS And jQuery

by Derek Mack

A Beginner's Guide To jQuery-Based JSON API Clients

by Ben Howdle

How To Build A Real-Time Commenting System

by Phil Leggetter

The Developer's Guide To Conflict-Free JavaScript And CSS In WordPress

by Peter Wilson

Optimizing Long Lists Of Yes/No Values With JavaScript

by Lea Verou

Building A Relationship Between CSS & JavaScript

by Tim Wright

About The Authors

Develop A One-Of-A-Kind CSS/JS-Based Game Portfolio

BY DANIEL STERNLICHT 🐉

A portfolio is a must-have for any designer or developer who wants to stake their claim on the Web. It should be as unique as possible, and with a bit of HTML, CSS and JavaScript, you could have a one-of-a-kind portfolio that capably represents you to potential clients. In this chapter, I'll show you how I created my 2-D Web-based game portfolio.



The 2-D Web-based game portfolio of Daniel Sternlicht¹.

Before getting down to business, let's talk about portfolios

Before getting down to business, let's talk about portfolios.

A portfolio is a great tool for Web designers and developers to show off their skills. As with any project, spend some time learning to develop a portfolio and doing a little research on what's going on in the Web design industry, so that the portfolio presents you as an up to date, innovative and inspiring person. All the while, keep in mind that going with the flow isn't necessarily the best way to stand out from the crowd.

One last thing before we dive into the mystery of my Web-based game portfolio. I use jQuery which has made my life much easier by speeding up development and keeping my code clean and simple.

Now, let's get our hands dirty with some code.

The HTML

Let's warm up with a quick overview of some very basic HTML code. It's a bit long, I know, but let's take it step by step.

```
<div id="wrapper">
```

```
  <hgroup id="myInfo">
```

```
<h1>DANIEL STERNLICHT</h1>
  <h2>Web Designer, Front-End Developer</h2>
</hgroup>

<div id="startCave" class="cave"></div>
<div id="startCaveHole" class="caveHole"></div>

<div id="mainRoad" class="road"></div>
<div id="leftFence"></div>
<div id="rightFence"></div>

<div id="daniel"></div>

<div id="aboutRoad" class="road side"></div>
<div id="aboutHouse" class="house">
  <div class="door"></div>
  <div class="lightbox">...</div>
<div id="aboutSign" class="sign">
  <span>About Me</span>
</div>

...

...

<div id="rightTrees" class="trees"></div>
<div id="leftGrass" class="grass"></div>

<div id="endSea" class="sea"></div>
<div id="endBridge" class="bridge"></div>
```



```
<div id="boat" class="isMoored">
  <div class="meSail"></div>
</div>

</div>
```

The HTML is not very complicated, and I could have used an HTML5 canvas² element for this game, but I felt more comfortable using simple HTML DOM elements.

Basically, we have the main **#wrapper** div, which contains the game's elements, most of which are represented as div elements (I chose divs because they are easy to manipulate).

Have a quick look at my game³. Can you detect what makes up the game view?



The game view

We have roads, trees, fences, water, caves, houses and so on.

Back to our HTML. You'll find an element for each of these items, with the relevant class and ID. Which brings us to the CSS.

The CSS

First of all, note that I prepared the HTML to follow the principles of object-oriented CSS⁴ by determining global classes for styling, and not using IDs as styling hooks. For example, I used the class **.road** on each element that should look like a road. The CSS for the **.road** class would be: **.road** {

```
position: absolute;
background: url(images/road.png) repeat;
}
```

Take trees as another example:

```
.trees {
  position: absolute;
  background: url(images/tree.png) repeat 0 0;
}
```

Note that almost all of the elements are absolutely positioned on the game's canvas. Positioning the elements relatively would be impossible for our purposes, especially because we want the game to be as **responsive** as possible (within limits, of course — the minimum width that I deal with is 640 pixels). We can write a general rule giving all of the DOM elements in the game an absolute position: `#wrapper * {`

```
position: absolute;
}
```

This snippet will handle all of the child elements inside the `#wrapper` div, and it frees us from having to repeat code.

One more word about the CSS. The animations in the game are done with **CSS3 transitions and animations**, excluding certain features such

the lightboxes and player “teleporting.” There are two reasons for this.

The first is that one of the purposes of this portfolio is to demonstrate innovation and up-to-date development, and what's more innovative than using the power of CSS3?

The second reason is performance. Upon reading Richard Bradshaw's very interesting article “Using CSS3 Transitions, Transforms and Animation⁵,” I came to the overwhelming conclusion: **use CSS3 when you can.**

A great example of the power of CSS3 animations in my portfolio is the pattern of movement of the water. The CSS looks like this: `.sea {`

```
    left: 0;
    width: 100%;
    height: 800px;
    background: url(images/sea.png) repeat 0 0;
    -webkit-animation: seamove 6s linear infinite; /*
Webkit support */
    -moz-animation: seamove 6s linear infinite; /*
Firefox support */
    animation: seamove 6s linear infinite; /*
Future browsers support */
}
```

And here is the code for the animation itself: `/* Webkit support */`

```
@-webkit-keyframes seamove {  
  0% {  
    background-position: 0 0;  
  }  
  100% {  
    background-position: 65px 0;  
  }  
}
```

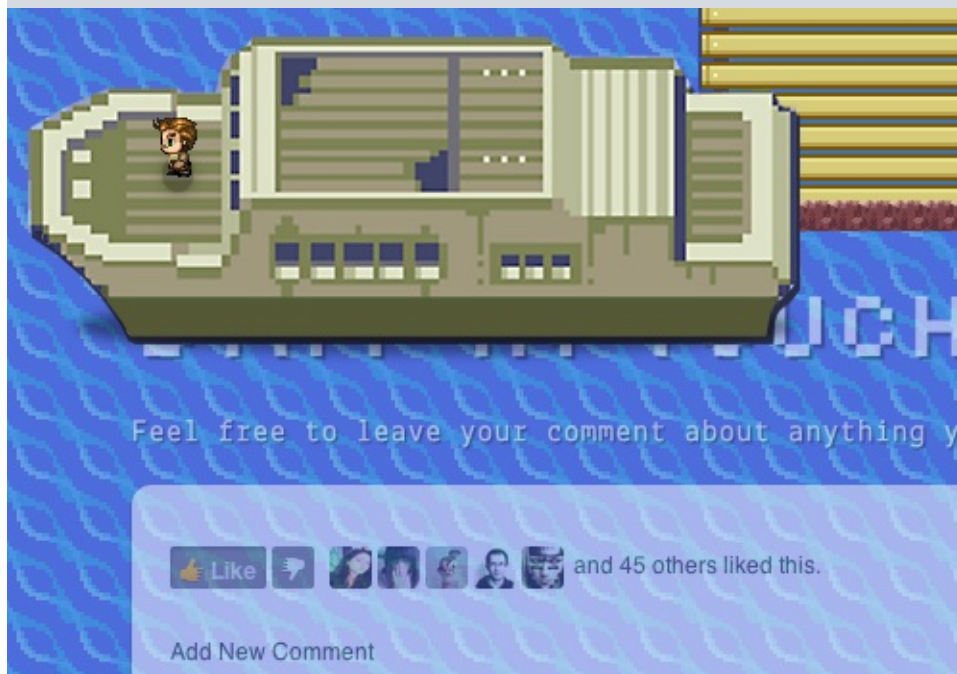
```
@-moz-keyframes seamove {...} /* Firefox support */  
@-keyframes seamove {...} /* Future browsers  
support */
```



The sea PNG is marked out.

The repeating **sea.png** image is 65 pixels wide, so to give the sea a waving effect, we should move it by the same number of pixels. Because the background is repeating, it gives us the effect we want.

Another cool example of CSS3 animations happens when the player steps into the boat and sails off the screen.



The boat sails off the screen, revealing the “Contact” section.

If the player gets back onto the road, you'll notice that the boat moves in “reverse,” back to its original position. It sounds complicated, but you have no idea how easy it is with CSS3 transitions. All I did was capture the event with JavaScript to determine whether the user is “on board.” If the user is, then we add the class **.sail** to the boat element, which make it sail off; otherwise, we withhold this class. At the same time, we

add a **.show** class to the **#contact** wrapper, which smoothly reveals the contact form in the water. The CSS of the boat looks like this: **#boat**

```
{  
    position: absolute;  
    bottom: 500px;  
    left: 50%;  
    margin-left: -210px;  
    width: 420px;  
    height: 194px;  
    background: url(images/boat.png) no-repeat center;  
    -webkit-transition: all 5s linear 1.5s;  
    -moz-transition: all 5s linear 1.5s;  
    transition: all 5s linear 1.5s;  
}
```

When we add the class **.sail** to it, all I'm doing is changing its **left** property.

```
#boat.sail {  
    left: -20%;  
}
```

The same goes for the **#contact** wrapper with the class **.show**. Except here, I'm playing with the **opacity** property: **#contact.show** {

```
    opacity: 1;  
}
```


CSS3 transitions do the rest of the work.

The JavaScript

Because we are dealing with a **2-D game**, we might want to base it on a JavaScript game engine, perhaps an existing framework. But the thing about frameworks (excluding jQuery, which I'm using as a base) is that they are usually good for a head start, but they probably won't fit your needs in the long run.

A good example is the lightboxes in my portfolio, which provide information about me and are activated when the user enters a house.



An example of a lightbox in the game. (Large image⁶)

This kind of functionality doesn't exist in a regular JavaScript game engine. You could always improve an existing framework with your own code, but diving into someone else's code sometimes takes longer than writing your own. Moreover, if you rewrite someone else's code, it could become a problem when a new version is released.

After passing over libraries such as Crafty⁷, LimeJS⁸ and Impact⁹, which really are great game engine frameworks, I felt I had no choice but to build my own engine to fit my needs.

Let's quickly review the main methods that I'm running in the game.

To handle the keyboard arrow events, I use the following code:

```
$(window).unbind('keydown').bind('keydown',
function(event) {
    switch (event.keyCode) {
        event.preventDefault();
        case 37: // Move Left
            me.moveX(me.leftPos - 5, 'left');
            break;

        case 39: // Move Right
            me.moveX(me.leftPos + 5, 'right');
```

```

        break;

    case 38: // Move Up
        me.moveY(me.topPos - 5, 'up');
        break;

    case 40: // Move Down
        me.moveY(me.topPos + 5, 'down');
        break;
    }
});

```

As you can see, the code is very simple. When the user presses the up or down arrow, I call the `moveY()` function, and when they press right or left, I call `moveX()`.

A quick peek at one of them reveals all the magic: `moveX`:

```

function(x, dir) {
    var player = this.player;
    var canMove = this.canImove(x, null);
    if(canMove){
        this.leftPos = x;
        player.animate({'left': x + 'px'}, 10);
    }
    if(dir == 'left') {
        this.startMoving('left', 2);
    }
    else {

```

```
        this.startMoving('right', 3);  
    }  
}
```

At each step the player takes, I check with a special method named **canImove()** (i.e. “Can I move?”) to determine whether the character may move over the game canvas. This method include screen boundaries, house positions, road limits and so on, and it gets two variables, including the x and y coordinates of where I want the player to move to. In our example, if I wanted the player to move left, I'd pass to the method their current left position plus 5 pixels. If I wanted them to move right, I'd pass its current position minus 5 pixels.

If the character “can move,” I return **true**, and the character keeps moving; or else, I return **false**, and the character remains in their current position.

Note that in the **moveX()** method, I'm also checking the direction in which the user wants to go, and then I call a method named

```
startMoving(): if(dir == 'left') {  
    this.startMoving('left', 2);  
}  
else {  
    this.startMoving('right', 3);  
}
```

You're probably wondering how the walking effect on the character is achieved. You might have noticed that I'm using CSS sprites. But how do I activate them? It's actually quite simple, with the help of a jQuery plugin called Spritely¹⁰. This amazing plugin enables you to animate CSS sprites simply by calling the method on the relevant element and passing it your properties (such as the number of frames).

Back to our **startMoving()** method: **startMoving**:

```
function(dir, state) {  
    player.addClass(dir);  
    player.sprite({fps: 9, no_of_frames:  
3}).spState(state);  
}
```

I simply add a direction class to the player element (which sets the relevant sprite image), and then call the **sprite()** method from Spritely's API.

Because we are dealing with the Web, I figured that being able to move with the keyboard arrows would not be enough. You always have to think of the user, your client, who might not have time to hang out in your world. That is why I added both a navigation bar and an option to “teleport” the character to a given point in the game — again, using the **canImove()** method to check whether the player may move to this point.

Next we've got the lightboxes. Recall what the HTML looks like for each house: `<div id="aboutHouse" class="house">`

```
<div class="door"></div>
<div class="lightbox">
  <div class="inner about">
    Lightbox content goes here...
  </div>
</div>
</div>
```

Did you notice the `.lightbox` class in the `house` div? We will use it later. What I basically did was define a “hot spot” for each house. When the player gets to one of those hot spots, the JavaScript activates the `lightboxInit(elm)` method, which also gets the relevant house's ID.

This method is very simple: `lightboxInit: function(elm) {`

```
// Get the relevant content
var content = $(elm).find('.lightbox').html();

// Create the lightbox
$('<div id="dark">
</div>').appendTo('body').fadeIn();
$('<div id="lightbox">' + content + '<span
id="closeLB">x</span>
</div>').insertAfter("#dark").delay(1000).fadeIn();
}
```

First, I get the relevant content by finding the `div.lightbox` child of the house element. Then, I create and fade in a blank div, named `dark`,

which gives me the dark background. Finally, I create another div, fill it up with the content (which I had already stored in a variable), and insert it right after the dark background. Clicking the “x” will call another method that fades out the lightbox and removes it from the DOM.

One good practice that I unfortunately learned the hard way is to **keep the code as dynamic as possible**. Write your code in such a way that if you add more content to the portfolio in future, the code will support it.

Conclusion

As you can see, developing a 2-D Web-based game is fun and not too complicated a task at all. But before rushing to develop your own game portfolio, consider that it doesn't suit everyone. If your users don't have any idea what HTML5 is or why IE 5.5 isn't the “best browser ever,” then your effort will be a waste of time, and perhaps this kind of portfolio would alienate them. Which is bad.

Nevertheless, I learned a lot from this development process and I highly recommend, whatever kind of portfolio you choose, that you invest a few days in developing your *own* one of a kind portfolio. 🐼

1. <http://danielsternlicht.com>
2. <http://sixrevisions.com/html/canvas-element/>
3. <http://danielsternlicht.com>
4. <http://coding.smashingmagazine.com/2011/12/12/an-introduction-to-object-oriented-css-oocss/>
5. <http://css3.bradshawenterprises.com/>
6. <http://media.smashingmagazine.com/wp-content/uploads/2012/04/5.png>
7. <http://craftyjs.com/>
8. <http://www.limejs.com/>
9. <http://impactjs.com/>
10. <http://www.spritely.net/>

Five Useful Interactive CSS/jQuery Techniques Deconstructed

BY JON RAASCH 

With the wide variety of CSS3 and JavaScript techniques available today, it's easier than ever to create unique interactive websites that delight visitors and provide a more engaging user experience.

In this chapter, we'll walk through five interactive techniques that you can start using right now.

Besides learning how to accomplish these specific tasks, you'll also master a variety of **useful CSS and jQuery tricks** that you can leverage when creating your own interactive techniques. The solutions presented here are certainly not perfect, so any thoughts, ideas and suggestions on how you would solve these design problems would be very appreciated.

So, let's dive in and start building more exciting websites!

1. Extruded Text Effect



The footer of David DeSandro's website¹ uses extruded text that animates on mouseover. This interactive text effect is a quick and impressive way to add some flare to your website. With only a few lines of CSS3, we can make the text appear to pop out of the page in **three dimensions**.

- View the demo²

First let's set up some text (the code is copied from the original site):

```
<span class="extruded">Extrude Me</span>
```

And some basic styling (the code is copied from the original site):

```
body  
{  
    background-color: #444;  
    text-align: center;  
}
```

```
.extruded {  
  color: #888;  
  font-family: proxima-nova-1, proxima-nova-2,  
'Helvetica Neue', Arial, sans-serif;  
  font-size: 48px;  
  font-weight: bold;  
  text-shadow: #000 3px 3px;  
}
```

Here, we've applied some basic styles and added a **text-shadow**. But this text-shadow doesn't look three-dimensional; to accomplish the extruded effect, we'll need to **add more text-shadows**: **text-shadow: #000 1px 1px, #000 2px 2px, #000 3px 3px;**

This will add three different text-shadows to our text, stacked on top of each other to create the three dimensional appearance we want.

STYLING THE HOVER STATE

Next, let's add a hover state with a bigger text-shadow:

```
.extruded:hover {  
  color: #FFF;  
  text-shadow: #58E 1px 1px, #58E 2px 2px, #58E 3px  
3px, #58E 4px 4px, #58E 5px 5px, #58E 6px 6px;  
}
```

Here, we've added three more text-shadows to **increase the depth** of the extrude effect. But this effect alone is too flat; we want the text to look like it's popping off the page. So, let's reposition the text to make it appear to grow taller from the base of the extruded section: `.extruded`

```
{
    position: relative;
}

.extruded:hover {
    color: #FFF;
    text-shadow: #58E 1px 1px, #58E 2px 2px, #58E 3px
3px, #58E 4px 4px, #58E 5px 5px, #58E 6px 6px;
    left: -6px;
    top: -6px;
}
```

Now in the hover state, the extruded text moves up the same distance as our max `text-shadow` value. We also added `position: relative`, which must be attached to the base state, not just the hover state, or else it will cause problems when we animate it.

ANIMATING THE TRANSITION

Next, we can add a CSS3 transition³ to our text to animate the color change and extrude effect: `.extruded` {

```
-moz-transition: all 0.3s ease-out; /* FF3.7+ */
```

```
        -o-transition: all 0.3s ease-out; /* Opera 10.5
*/
        -webkit-transition: all 0.3s ease-out; /* Saf3.2+,
Chrome */
        transition: all 0.3s ease-out;
    }
}
```

This triggers a smooth animation for our different CSS changes on hover. While it doesn't work in all browsers, it does degrade nicely to the basic hover effect.

Bringing it all together:

```
body {
    background-color: #444;
    text-align: center;
}

.extruded {
    color: #888;
    font-family: proxima-nova-1, proxima-nova-2,
'Helvetica Neue', Arial, sans-serif; /* the original
site doesn't use the @font-face attribute */
    font-size: 48px;
    font-weight: bold;
    text-shadow: #000 1px 1px, #000 2px 2px, #000 3px
3px;
```

```
position: relative;
-moz-transition: all 0.3s ease-out; /* FF3.7+ */
-o-transition: all 0.3s ease-out; /* Opera 10.5
*/
-webkit-transition: all 0.3s ease-out; /* Saf3.2+,
Chrome */
transition: all 0.3s ease-out;
}

.extruded:hover {
color: #FFF;
text-shadow: #58E 1px 1px, #58E 2px 2px, #58E 3px
3px, #58E 4px 4px, #58E 5px 5px, #58E 6px 6px;
left: -6px;
top: -6px;
}
```

SHORTCOMINGS

While applying several CSS3 text-shadows works well when the text is static, it falls a bit short when used alongside the transition animation.

In short, the biggest text-shadow animates just fine, but the other text-shadows aren't applied until the animation completes. This causes a quick correction: the browser stutters with a basic drop-shadow before filling in the rest diagonally.

Fortunately, we can make this drawback relatively unnoticeable, provided that we follow a few style guidelines. Basically, we want to hide the bulk of the extruded portion with the top text. This means that we should generally use this effect with bolder fonts, such as the Proxima Nova family used by David DeSandro. Also, we should be careful to avoid text-shadows that are too big for the font. Tweak your settings with this in mind until the animated extrude looks believable.

Finally, this technique **will not work in IE**, because **text-shadow** is unsupported in all versions of IE (even IE9).

- [Download the complete example](#)⁴
- [View the demo](#)²

2. Animating A Background Image



Galileo and a bully in a dress

You would expect that 400 years after somebody rediscovered that the earth is not the center of the universe everybody would just accept that as a fact and move on. No, not everybody. On Saturday November 6th 2010 the [First Annual Catholic Conference on Geocentrism: Galileo Was Wrong](#) will be held in a small town somewhere in the USA. These people earn their money with writing scientific books about this matter! That is just amazing, literally. I'm jealous, I wish I could make a living just by publishing huge piles of nonsense. Unfortunately I don't have the money to attend this conference, without a doubt it would be the best and most entertaining waste of time ever.

[There is some more](#)

WRITTEN BY VASILIS ON OCTOBER 10TH, 2010 [DIRECT LINK](#)

While we can easily animate text with a few lines of code, animating an image usually requires bigger and slower assets, such as animated GIFs⁶ or Flash or HTML5 video. While complex animations will still depend on these technologies, we can create a compelling illusion of animation using CSS alone.

Love Nonsense⁷ uses a hover effect to alter the color of the images on the website. The trick here is to use a transparent PNG with a background color. The color of the PNG should match the website's background, so that all of the transparent areas in the PNG show up when filled with a background color. Thus, the PNG should contain the **negative space** of the image you want to display (i.e. the shape you want should be transparent, and everything else should be the same color as the background).

Here's an example of the Smashing Magazine logo with negative space:



Notice in the demo⁸ how when the background color is set to orange, it starts to look more like the real thing.

THE CODE

- View the demo⁹

First, let's do some basic mark-up:

```
<div class="post-wrapper">
  <h2 class="post-title">
    This is the title you hover over

    
  </h2>
```



```
    <p>Some more text here.</p>
</div>
```

Here we include a post with a title, our knockout image and a paragraph of text.

```
Next, let's set up some static styles: .post-wrapper {
    position: relative;
    padding-left: 240px;
}

.post-image {
    position: absolute;
    top: 0;
    left: 0;
    background-color: #bbb;
}

.post-title {
    color: #037072;
}
```

Here, we've set up the post's wrapper with **position: relative** and with enough padding on the left side to absolutely position the image to the left of our post. We've also added a background color to our image; so now the positive space in our PNG shows up as light gray.

Next, let's add some hover effects:

```
.post-title:hover {  
    color: #d63c25;  
}  
  
.post-title:hover .post-image {  
    background-color: #f04e36;  
}
```

Now, when we hover over the title or the image, both will change color.

We can take this effect a step further by animating the transition: `.post-image`

```
{  
    -webkit-transition: background-color 400ms ease-in;  
    -moz-transition: background-color 400ms ease-in;  
    transition: background-color 400ms ease-in;  
}  
  
.post-title {  
    -webkit-transition: color 400ms ease-in;  
    -moz-transition: color 400ms ease-in;  
    transition: color 400ms ease-in;  
}
```

Here, we've added a CSS3 transition¹⁰ to both the image and the title, which will make for a smooth color change animation.

Unfortunately, CSS3 transitions are not currently supported in IE9. However, even in unsupported browsers, the color change will still occur—it just won't have a smooth animation.

If complete cross-browser support for the animation is important, you could always provide a jQuery version of the animation for unsupported browsers. Bear in mind, though, that jQuery's `animate()` method¹¹ does not support color animations, so you'll need to use a color plug-in¹².

Putting all the CSS together:

```
.post-wrapper {  
    position: relative;  
    padding-left: 240px;  
}  
  
.post-image {  
    position: absolute;  
    top: 0;  
    left: 0;  
    background-color: #bbb;
```

```
-webkit-transition: background-color 400ms ease-in;
-moz-transition: background-color 400ms ease-in;
transition: background-color 400ms ease-in;
}

.post-title {
  color: #037072;
  -webkit-transition: color 400ms ease-in;
  -moz-transition: color 400ms ease-in;
  transition: color 400ms ease-in;
}

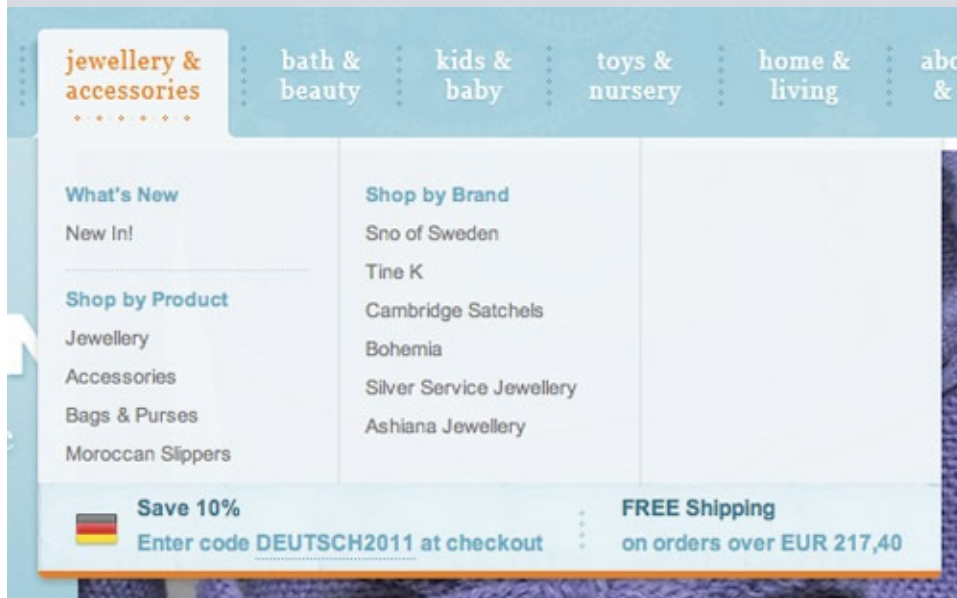
/* add the hover states */

.post-title:hover {
  color: #d63c25;
}

.post-title:hover .post-image {
  background-color: #f04e36;
}
```

- Download the complete example¹³
- View the demo⁹

3. Mega Dropdown



One common design problem with dropdown menus is that they often contain a lot of items. Instead of presenting all of its items in a long single column, Bohemia Design¹⁵ uses a **multi-column dropdown**. This approach not only looks great, but provides an opportunity to group the links and highlight the most important ones.

Let's recreate this menu using CSS and jQuery.

- View the demo¹⁶

BUILDING THE TABS

Ideally, we would start with a lean and simple mark-up...

```
<nav>
  <li><a href="#">Tab 1</a></li>
  <li><a href="#">Tab 2</a></li>
  <li><a href="#">Tab 3</a></li>
  <li><a href="#">Tab 4</a></li>
  <li><a href="#">Tab 5</a></li>
</nav>
```

...and use `nav li a`, `nav > li` or `nav li` to style the list items in the navigation. The child selector doesn't work in IE6 and `nav li` would cause problems since there are additional LIs nested in the content area of the dropdown. If you absolutely need the site to work for IE6 users as well (and that's what you sometimes will have to do), you'll need to have markup similar to the original mark-up in this example: `<ul id="main-nav">`

```
<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 1</a>
</li>

<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 2</a>
</li>
```

```
<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 3</a>
</li>

<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 4</a>
</li>

<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 5</a>
</li>
</ul>
```

Next, let's style these five tabs:

```
#main-nav {
  width: 800px;
  height: 50px;
  position: relative;
  list-style: none;
  padding: 0;
}

#main-nav .main-nav-item {
  display: inline;
}

#main-nav .main-nav-tab {
```

```
float: left;
width: 140px;
height: 30px;
padding: 10px;
line-height: 30px;
text-align: center;
color: #FFF;
text-decoration: none;
font-size: 18px;
}
```

Although a lot of the CSS is specific to our example, there are a few important styles to note.

First, we've defined a height and width for our overall tab area and matched the total height and width of all five tabs, so that we can position the dropdown correctly. Next, we've defined **position: relative** for the tab wrapper, which will allow us to position the dropdown absolutely.

Then, we added **list-style: none** to the list wrapper, and **display: inline** to each list item, to eliminate any list styling.

Finally, we floated all of the tab links to the left.

BUILDING THE DROPDOWN

Now, let's build the dropdown mark-up in one of our tab wrappers:

```
<li class="main-nav-item">
  <a href="#" class="main-nav-tab">Tab 1</a>

  <div class="main-nav-dd">
    <div class="main-nav-dd-column">
      Column content here
    </div>
  </div>

  <div class="main-nav-dd">
    <div class="main-nav-dd-column">
      Column content here
    </div>
  </div>

  <div class="main-nav-dd">
    <div class="main-nav-dd-column">
      Column content here
    </div>
  </div>
</li>
```

Next, let's style this dropdown:

```
#main-nav .main-nav-dd {
```

```
position: absolute;
top: 50px;
left: 0;
margin: 0;
padding: 0;
background-color: #FFF;
border-bottom: 4px solid #f60;
}

#main-nav .main-nav-dd-column {
width: 130px;
padding: 15px 20px 8px;
display: table-cell;
border-left: 1px solid #ddd;
*float: left;
*border-left: 0;
}

#main-nav .main-nav-dd-column:first-child {
border-left: 0;
}
```

Here, we've positioned the dropdown absolutely, directly beneath the first tab.

Let's set **display: table-cell** on all of the column wrappers, so that they display next to each other. But **table-cell** is not supported in IE6

or 7, so we've used an attribute hack¹⁷ as an alternative for IE6 and 7. This hack places an asterisk (*) before each of the attributes that are specific to IE6 and 7.

Thus, we've defined a backup for unsupported IEs, which is simply **float: left**. This works almost as well as **display: table-cell**, except that the floated elements don't match each other's height, so the borders between columns don't line up. To avoid this minor issue, we simply remove the **border-left** using the same asterisk hack.

Finally, we remove the left border from the first column for all browsers. Although the **:first-child** pseudo-class doesn't work properly in IE6, fortunately it doesn't make a difference, because we've already hidden the borders in these browsers.

ADDING THE INTERACTION

We've built the mark-up and styles for our dropdown, but we still need to make the menu interactive. Let's use jQuery to add a class to show and hide the dropdown: `$(function() {`

```
    var $mainNav = $('#main-nav');

    $mainNav.children('.main-nav-
item').hover(function(ev) {
        // show the dropdown
```

```
        $(this).addClass('main-nav-item-active');
    }, function(ev) {
        // hide the dropdown
        $(this).removeClass('main-nav-item-active');
    });
});
```

Here, we've attached a hover listener¹⁸ to each list item, which adds and removes the class **main-nav-item-active**. Attach this to the list item rather than the tab itself, or else the dropdown will disappear when the user mouses off the tab and into the dropdown area.

Now we can use this class hook to hide and show the dropdown with

```
CSS: #main-nav .main-nav-dd {
    display: none;
}

#main-nav .main-nav-item-active .main-nav-dd {
    display: block;
}
```

Let's use the **active** class to style the active tab: #main-nav .main-nav-item-active .main-nav-tab {

```
    background-color: #FFF;
    color: #f60;
    -webkit-border-topleft-radius: 5px;
    -webkit-border-topright-radius: 5px;
```

```
-moz-border-radius-topleft: 5px;
-moz-border-radius-topright: 5px;
border-top-left-radius: 5px;
border-top-right-radius: 5px;
}
```

Here, we've changed the background and text colors and rounded the top corners (in supported browsers).

POSITIONING THE DROPDOWN

Now the basic mouse interaction has been built and the dropdown displays on mouseover. Unfortunately, it is still not positioned correctly under each tab, so let's add some more code to our hover events:

```
$(function() {
    var $mainNav = $('#main-nav');

    $mainNav.children('.main-nav-
item').hover(function(ev) {
        var $this = $(this),
            $dd = $this.find('.main-nav-dd');

        // get the left position of this tab
        var leftPos = $this.find('.main-nav-
tab').position().left;
```

```

        // position the dropdown

        $dd.css('left', leftPos);

        // show the dropdown
        $this.addClass('main-nav-item-active');
    }, function(ev) {

        // hide the dropdown
        $(this).removeClass('main-nav-item-active');
    });
});

```

Here, we use jQuery's `position()` method¹⁹ to get the left offset from the current tab. We then use this value to position the dropdown directly beneath the appropriate tab.

However, with the tabs on the right side, the dropdown menu will end up poking out of the tab area. Besides looking bad, this could lead to **overflow issues**, with portions of the dropdown falling outside of the browser window.

Let's fix the positioning with some JavaScript: `$(function() {`
`var $mainNav = $('#main-nav'),`
`navWidth = $mainNav.width();`
`$mainNav.children('.main-nav-`

```

item').hover(function(ev) {
    var $this = $(this),
        $dd = $this.find('.main-nav-dd');

    // get the left position of this tab
    var leftPos = $this.find('.main-nav-tab').position().left;

    // get the width of the dropdown
    var ddWidth = $dd.width(),
        leftMax = navWidth - ddWidth;

    // position the dropdown
    $dd.css('left', Math.min(leftPos, leftMax) );

    // show the dropdown
    $this.addClass('main-nav-item-active');
}, function(ev) {

    // hide the dropdown
    $(this).removeClass('main-nav-item-active');
});
});

```

Here, we start by finding the overall width of the tab area. Because recalculating the width for each tab is not necessary, we can define it outside of our hover listener.

Next, we find the width of the dropdown and determine the maximum left value, which is the overall tab width minus the width of the dropdown.

Finally, instead of always positioning the dropdown directly beneath the tab, we use the `Math.min()` method²⁰ to pick the lowest between the tab offset and the maximum left value.

Thus, we confine the dropdown to the area beneath the tabs and avoid any content issues.

OTHER APPROACHES

While this script is fully functional, we could still improve the **user experience**. Currently, when the user mouses away from the dropdown, the menu hides immediately. You could build a delay using `setTimeout()` to ensure that the dropdown remains visible when the user mouses away and then quickly mouses back. This creates a better experience, because it avoids hiding the dropdown during accidental movements.

If you'd rather avoid `setTimeout()`, you could also look into the `hoverIntent` jQuery plug-in²¹, which makes fine-tuned control over mouse actions much easier.

Besides improving the user experience, you could also **avoid jQuery altogether** in all browsers except IE6.

Instead of using jQuery's `hover()` listener, we could use the CSS pseudo-class `:hover` to hide and show the dropdown.

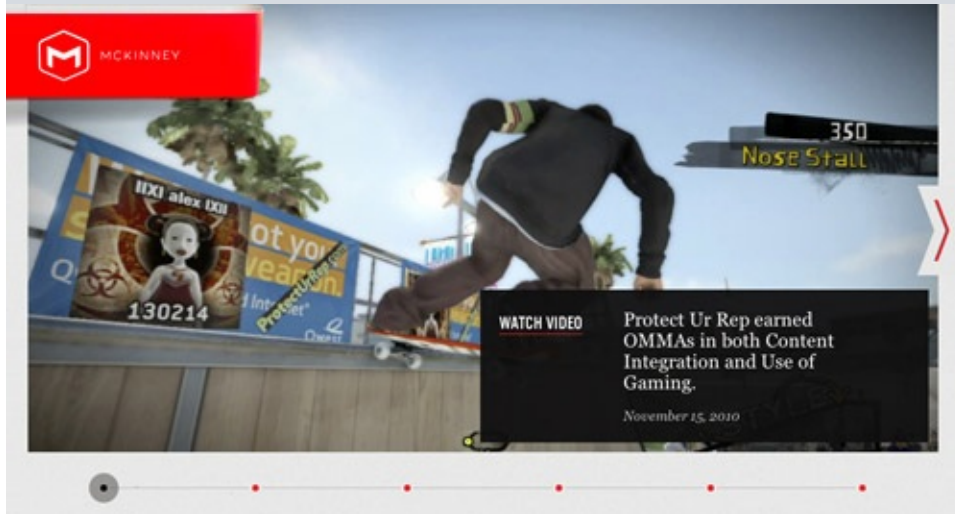
One downside with the CSS-only solution is that you can't build a delay for the `:hover` pseudo-class.

Also, you will have to position the dropdown manually under each tab to avoid the overflow issues. Alternatively, if you aren't concerned with overflow issues, you could attach `position: relative` to each list item and avoid setting any positions manually.

Finally, if you're supporting IE6, make sure to include the script above as a backup for IE6 (but don't include it for other browsers).

- Download the complete example²²
- View the demo¹⁶

4. Animated Slideshow Navigation



There are a lot of JavaScript slideshow techniques, but the animated navigation on McKinney²⁴ is a fresh, subtle approach.

BASIC JQUERY SLIDESHOW

- View the demo²⁵

Let's build something similar. We'll start with some mark-up for a basic slideshow: `<div id="slideshow">`

```
<div id="slideshow-reel">
  <div class="slide">
    <h1>Slide 1</h1>
```

```
</div>

<div class="slide">
  <h1>Slide 2</h1>
</div>

<div class="slide">
  <h1>Slide 3</h1>
</div>

<div class="slide">
  <h1>Slide 4</h1>
</div>

<div class="slide">
  <h1>Slide 5</h1>
</div>

<div class="slide">
  <h1>Slide 6</h1>
</div>
</div>
</div>
```

Here we've set up six slides, which can be filled with any content we need. Let's set up some CSS to display the slides as a horizontal reel:

```
#slideshow {
  width: 900px;
  height: 500px;
```

```
        overflow: hidden;
        position: relative;
    }

    #slideshow-reel {
        width: 5400px;
        height: 450px;
        position: absolute;
        top: 0;
        left: 0;
    }

    #slideshow-reel .slide {
        width: 900px;
        height: 450px;
        float: left;
        background-color: gray;
    }
```

Here, we've defined the dimensions of the slideshow, along with **overflow: hidden** to hide the other slides in the reel. We've also defined the dimensions of the reel: with six slides at 900 pixels each, it is 5400 pixels wide. (You could also just set this to a really high number, like 10000 pixels.) Then, we absolutely positioned the reel inside the slideshow (which has **position: relative**). Finally, we defined the dimensions for all of the individual slides and floated them to the left to fill up our reel.

BASIC SLIDESHOW ANIMATION

Now, let's add some jQuery to animate this slideshow: `$(function()`

```
{
    function changeSlide( newSlide ) {
        // change the currSlide value
        currSlide = newSlide;

        // make sure the currSlide value is not too
low or high
        if ( currSlide > maxSlide ) currSlide = 0;
        else if ( currSlide < 0 ) currSlide =
maxSlide;

        // animate the slide reel
        $slideReel.animate({
            left : currSlide * -900
        }, 400, 'swing', function() {
            // set new timeout if active
            if ( activeSlideshow ) slideTimeout =
setTimeout(nextSlide, 1200);
        });
    }

    function nextSlide() {
        changeSlide( currSlide + 1 );
    }

    // define some variables / DOM references
```

```
var activeSlideshow = true,
    currSlide = 0,
    slideTimeout,
    $slideshow = $('#slideshow'),
    $slideReel = $slideshow.find('#slideshow-reel'),
    maxSlide = $slideReel.children().length - 1;

// start the animation
slideTimeout = setTimeout(nextSlide, 1200);
});
```

Here, we've started by creating the function **changeSlide()**, which animates the slide reel. This function accepts an index for the next slide to show, and it checks to make sure that the value isn't too high or low to be in the reel.

Next, it animates the slide reel to the appropriate position, and then finishes by setting a new timeout²⁶ to trigger the next iteration.

Finally, we've built the function **nextSlide()**, which simply triggers **changeSlide()** to show the next slide in the reel. This simple function is just a shortcut to be used with **setTimeout()**.

THE LEFT AND RIGHT NAVIGATION

Next, let's set up the left and right arrows in the slideshow, starting with the mark-up:

```
<a href="#" id="slideshow-prev"></a>  
<a href="#" id="slideshow-next"></a>
```

For simplicity's sake, we've added the mark-up to the HTML source. Appending it to the jQuery is often a better approach, to ensure that the controls appear only when they are usable.

Let's style these arrows with CSS:

```
#slideshow-prev, #slideshow-next {  
    display: block;  
    position: absolute;  
    top: 190px;  
    width: 0;  
    height: 0;  
    border-style: solid;  
    border-width: 28px 21px;  
    border-color: transparent;  
    outline: none;  
}  
  
#slideshow-prev:hover, #slideshow-next:hover {  
    opacity: .5;  
    filter: alpha(opacity=50);  
    -ms-
```

```

filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=50)";
}

#slideshow-prev {
    left: 0;
    border-right-color: #fff;
}

#slideshow-next {
    right: 0;
    border-left-color: #fff;
}

```

We've positioned the arrows absolutely within the slideshow frame and added an opacity change on hover. In our example, we've used a CSS triangle trick²⁷ to style the arrows with straight CSS, but feel free to use an image if you want richer graphics.

Finally, let's build the required interaction into our JavaScript:

```

$(function() {
    function changeSlide( newSlide ) {
        // cancel any timeout
        clearTimeout( slideTimeout );

        // change the currSlide value
        currSlide = newSlide;
    }
}

```



```

        // make sure the currSlide value is not too
low or high
        if ( currSlide > maxSlide ) currSlide = 0;
        else if ( currSlide < 0 ) currSlide =
maxSlide;

        // animate the slide reel
        $slideReel.animate({
            left : currSlide * -900
        }, 400, 'swing', function() {
            // hide / show the arrows depending on
which frame it's on
            if ( currSlide == 0 )
$slidePrevNav.hide();
            else $slidePrevNav.show();

            if ( currSlide == maxSlide )
$slideNextNav.hide();
            else $slideNextNav.show();

            // set new timeout if active
            if ( activeSlideshow ) slideTimeout =
setTimeout(nextSlide, 1200);
        });

        // animate the navigation indicator
        $activeNavItem.animate({
            left : currSlide * 150
        }, 400, 'swing');

```

```

}

function nextSlide() {
    changeSlide( currSlide + 1 );
}

// define some variables / DOM references
var activeSlideshow = true,
    currSlide = 0,
    slideTimeout,
    $slideshow = $('#slideshow'),
    $slideReel = $slideshow.find('#slideshow-reel'),
    maxSlide = $slideReel.children().length - 1,
    $slidePrevNav = $slideshow.find('#slideshow-
prev'),
    $slideNextNav = $slideshow.find('#slideshow-
next');

// set navigation click events

// left arrow
$slidePrevNav.click(function(ev) {
    ev.preventDefault();

    activeSlideshow = false;

    changeSlide( currSlide - 1 );
});

// right arrow

```

```
$slideNextNav.click(function(ev) {  
    ev.preventDefault();  
  
    activeSlideshow = false;  
  
    changeSlide( currSlide + 1 );  
});  
  
// start the animation  
slideTimeout = setTimeout(nextSlide, 1200);  
});
```

Here, we've added quite a bit of new interaction. First, look at the bottom of this script, where we've added click event listeners to both of our navigational items.

In these functions, we have first set **activeSlideshow** to **false**, which disables the automatic animation of the reel. This provides a better user experience by allowing the user to control the reel manually. Then, we trigger either the previous or next slide using **changeSlide()**. Next, in the **changeSlide()** function, we've added a **clearTimeout()**²⁸. This works in conjunction with the **activeSlideshow** value, cancelling any hanging iteration from a **setTimeout**.

Finally, in the callback of the **animate()** function, we've added some code to hide and show the arrow navigation. This hides the left arrow

when the slideshow is showing the left-most slide, and vice versa.

ANIMATING THE BOTTOM NAVIGATION

The basic slideshow works with the previous and next arrows. Let's take it to the next level by adding the animated navigation. Please note that I am using a more complex markup because it avoids the use of images and is ultimately simpler. It would have to use three background images otherwise — one for the center sections and one for each cap to allow the clickable areas to be larger). However, you could clean up the bottom navigation with a background-image.

Here is the jQuery code for animation: `$(function() {`
 `function changeSlide(newSlide) {`
 `// cancel any timeout`
 `clearTimeout(slideTimeout);`

 `// change the currSlide value`
 `currSlide = newSlide;`

 `// make sure the currSlide value is not too`
`low or high`
 `if (currSlide > maxSlide) currSlide = 0;`
 `else if (currSlide < 0) currSlide =`
`maxSlide;`

```

        // animate the slide reel
        $slideReel.animate({
            left : currSlide * -900
        }, 400, 'swing', function() {
            // hide / show the arrows depending on
            which frame it's on
            if ( currSlide == 0 )
                $slidePrevNav.hide();
            else $slidePrevNav.show();

            if ( currSlide == maxSlide )
                $slideNextNav.hide();
            else $slideNextNav.show();

            // set new timeout if active
            if ( activeSlideshow ) slideTimeout =
                setTimeout(nextSlide, 1200);
        });

        // animate the navigation indicator
        $activeNavItem.animate({
            left : currSlide * 150
        }, 400, 'swing');
    }

    function nextSlide() {
        changeSlide( currSlide + 1 );
    }

    // define some variables / DOM references

```

```
var activeSlideshow = true,
currSlide = 0,
slideTimeout,
$slideshow = $('#slideshow'),
$slideReel = $slideshow.find('#slideshow-reel'),
maxSlide = $slideReel.children().length - 1,
$slidePrevNav = $slideshow.find('#slideshow-
prev'),
$slideNextNav = $slideshow.find('#slideshow-
next'),
$activeNavItem = $slideshow.find('#active-nav-
item');

// set navigation click events

// left arrow
$slidePrevNav.click(function(ev) {
    ev.preventDefault();

    activeSlideshow = false;

    changeSlide( currSlide - 1 );
});

// right arrow
$slideNextNav.click(function(ev) {
    ev.preventDefault();

    activeSlideshow = false;
```

```
        changeSlide( currSlide + 1 );
    });

    // main navigation
    $slideshow.find('#slideshow-nav a.nav-
item').click(function(ev) {
        ev.preventDefault();

        activeSlideshow = false;

        changeSlide( $(this).index() );
    });

    // start the animation
    slideTimeout = setTimeout(nextSlide, 1200);
});
```

We've added a couple of things to our script.

First, we've included a second animation in **changeSlide()**, this time to animate the active indicator in the navigation. This **animate()** is basically the same as the one we built for the reel, the main difference being that we want to move it only **150px** per slide.

Finally, we added a click event listener to the items in the bottom navigation. Similar to the arrow navigation, we start by disabling the

automatic animation, setting **activeSlideshow** to **false**. Next, we trigger **changeSlide()**, passing in the index of whichever slide was clicked, which is easy to determine using jQuery's **index()**²⁹ method.

Now the slideshow navigation animation is complete and ready to impress your visitors.

- Download the complete example³⁰
- View the demo²⁵

5. Animated Icons



CSS-Tricks³² has a simple but elegant effect in its footer when the user mouses over various buttons. Besides the color changing and an icon being added, the effect is animated in browsers that support transition,

making the icon appear to **slide into place**.

- View the demo³³

Let's create a similar effect, starting with some mark-up: ``

`<h3>Panel Title</h3>`

`<p>Additional information about the panel goes in a paragraph here</p>`

``

One thing to note about this mark-up is that it has block elements nested in an `<a>` element, which makes sense semantically, but it's valid only if you're using the HTML5 doc type.

STYLING THE BUTTONS

Let's set up some basic CSS to style the block in its natural (non-hovered) state: `.hover-panel {`

`background-color: #E6E2DF;`

`color: #B2AAA4;`

`float: left;`

`height: 130px;`

```
width: 262px;
margin: 0 10px 10px 0;
padding: 15px;
}

.hover-panel h3 {
  font-family: tandelle-1, tandelle-2, Impact, Sans-
serif, sans;
  font-size: 38px;
  line-height: 1;
  margin: 0 0 10px;
  text-transform: uppercase;
}

.hover-panel p {
  font-size: 12px;
  width: 65%;
}
```

Now let's add a static hover effect to change some of the colors and add a drop shadow: `.hover-panel:hover` {

```
background-color: #237ABE;
}

.hover-panel:hover h3 {
  color: #FFF;
  text-shadow: rgba(0, 0, 0, 0.398438) 0px 0px 4px;
}
```

```
.hover-panel:hover p {  
    color: #FFF;  
}
```

Finally, let's add a background image that pops into place on hover:

```
.hover-panel {  
    background-image: url(hover-panel-icon.png);  
    background-position: 292px 10px;  
    background-repeat: no-repeat;  
}  
  
.hover-panel:hover {  
    background-position: 180px 10px;  
}
```

Here, we've added a few important styles to accomplish the hover effect. First, we've attached the background image to our **.hover-panel**. This is normally positioned outside of the button, but on mouseover, it is placed correctly. Also, note that we've placed it off to the right side of the panel, so that when we apply the transition animation, the icon will slide in from the right.

ANIMATING THE TRANSITION

Finally, let's add the transition:

```
.hover-panel {  
    -moz-transition: all 0.2s ease; /* FF3.7+ */  
    -o-transition: all 0.2s ease; /* Opera 10.5 */  
    -webkit-transition: all 0.2s ease; /* Saf3.2+,  
Chrome */  
    transition: all 0.2s ease;  
}
```

The transition effect triggers the animation of the background image. Because we've flagged it to apply to **all** attributes, the transition will also be applied to the **background-color** change that we applied above.

Although this works in most modern browsers, it will not work in IE9. But even in unsupported browsers, the user will see the color change and icon; they just won't see the animation effect.

On most websites, this enhancement wouldn't be necessary for all users. But if support is a priority, look into this jQuery back-up³⁴.

Finally, let's bring all of the styles together: **.hover-panel** {
 background-color: #E6E2DF;
 background-image: url(hover-panel-icon.png);
 background-position: 292px 10px;

```
background-repeat: no-repeat;
color: #B2AAA4;
display: block;
float: left;
height: 130px;
width: 262px;
margin: 0 10px 10px 0;
padding: 15px;
    -moz-transition: all 0.2s ease; /* FF3.7+ */
    -o-transition: all 0.2s ease; /* Opera 10.5 */
    -webkit-transition: all 0.2s ease; /* Saf3.2+,
Chrome */
    transition: all 0.2s ease;
}

.hover-panel h3 {
    font-family: tandelle-1, tandelle-2, Impact, Sans-
serif, sans;
    font-size: 38px;
    line-height: 1;
    margin: 0 0 10px;
    text-transform: uppercase;
}

.hover-panel p {
    font-size: 12px;
    width: 65%;
}

.hover-panel:hover {
```

```
background-color: #237ABE;
background-position: 180px 10px;
}

.hover-panel:hover h3 {
  color: #FFF;
  text-shadow: rgba(0, 0, 0, 0.398438) 0px 0px 4px;
}

.hover-panel:hover p {
  color: #FFF;
}
```

- Download the complete example³⁵
- View the demo³³

Final Thoughts

In this chapter, we've walked through a variety of interactive techniques that can add a bit of style and creativity to your website. Used correctly, techniques like these enhance websites, creating a more engaging and memorable user experience. But be subtle with the interactivity, ensuring that the bells and whistles do not get in the way of the website's primary function, which is **providing meaningful content**. 🐼

—

1. <http://desandro.com/>
2. <http://jonraasch.com/demo/extrude-on-hover>
3. <http://net.tutsplus.com/tutorials/html-css-techniques/css-fundamentals-css-3-transitions/>
4. <https://gist.github.com/957199>
5. <http://jonraasch.com/demo/extrude-on-hover>
6. <http://www.gifmuseum.com/>
7. <http://lovenonsense.com/>
8. <http://jonraasch.com/demo/animated-image>
9. <http://jonraasch.com/demo/animated-image>
10. <http://net.tutsplus.com/tutorials/html-css-techniques/css-fundamentals-css-3-transitions/>

11. <http://api.jquery.com/animate/>
12. <https://github.com/jquery/jquery-color>
13. <https://gist.github.com/957217>
14. <http://jonraasch.com/demo/animated-image>
15. <http://www.bohemiadesign.co.uk/>
16. <http://jonraasch.com/demo/huge-dropdown-menu>
17. <http://paulirish.com/2009/browser-specific-css-hacks/>
18. <http://bavotasan.com/tutorials/a-simple-mouseover-hover-effect-with-jquery/>
19. <http://api.jquery.com/position/>
20. http://www.tutorialspoint.com/javascript/math_min.htm
21. <http://cherne.net/brian/resources/jquery.hoverIntent.html>

22. <https://gist.github.com/957213>
23. <http://jonraasch.com/demo/huge-dropdown-menu>
24. <http://mckinney.com/>
25. <http://jonraasch.com/demo/fancy-slideshow-navigation>
26. <http://www.elated.com/articles/javascript-timers-with-settimeout-and-setinterval/>
27. <http://www.dinnermint.org/blog/css/creating-triangles-in-css/>
28. <http://programming.top54u.com/post/JavaScript-clearTimeout.aspx>
29. <http://api.jquery.com/index/>
30. <https://gist.github.com/957203>
31. <http://jonraasch.com/demo/fancy-slideshow-navigation>
32. <http://css-tricks.com>
33. <http://jonraasch.com/demo/animated-icons>

34. <http://jonraasch.com/blog/graceful-degradation-with-css3#transitions>

35. <https://gist.github.com/957236>

36. <http://jonraasch.com/demo/animated-icons>

Create An Animated Bar Graph With HTML, CSS And jQuery

BY DEREK MACK 

People in boardrooms across the world love a good graph. They go nuts for PowerPoint, bullet points and phrases like “run it up the flagpole,” “blue-sky thinking” and “low-hanging fruit,” and everything is always “moving forward.” Backwards is not an option for people who facilitate paradigm shifts in the zeitgeist¹. Graphs of financial projections, quarterly sales figures and market saturation are a middle-manager's dream.



How can we as Web designers get in on all of this hot graph action? There are actually quite a few ways to display graphs on the Web. We could simply create an image and nail it to a Web page. But that's not very accessible or interesting. We could use Flash, which is quite good

for displaying graphs—but again, not very accessible. Besides, designers, developers and deities² are falling out of love with Flash. Technologies such as HTML5 can do many of the same things without the need for a plug-in. The new HTML5 `<canvas>` element could even be adapted to the task. Plenty of charting tools are online that we might use. But what if we wanted something a little more tailored?

There are pros and cons to the wide range of resources available to us, but this tutorial will not explore them all. Instead, we'll create our graph using a progressively enhanced³ sprinkling of CSS3 and jQuery. Because we can.

What Are We Making?

We're making this⁴. And more! Here are some possibilities on how you can extend the techniques explored in this tutorial: A progress bar that indicates how long until the end of all humanity in the event of a zombie plague; A graph indicating the decline in safe outdoor activities during a zombie plague; A frighteningly similar graph indicating the decline in manners during a zombie plague; The increase of people who were unaware of the zombie plague because they were sharing with all of their now-deceased friends on Facebook what they did on FarmVille⁵.

Or you could create a graph or quota bar that simply illustrates something useful and less full of dread and zombies. So, let's get on with it.

WHAT YOU'LL NEED

- A text or HTML editor. Take your pick⁶; many are out there.
- jQuery⁷. Practice safe scripting and get the latest one. Keep the jQuery website open so that you can look up the documentation as you go.
- Possibly an image editor, such as Paint⁸, to mock up what your graph might look like.
- A modern and decent Web browser to preview changes.

That should do it. Please note that this tutorial is not designed as an introduction to either HTML, CSS, jQuery or zombies. Some intermediate knowledge of these three technologies and the undead is assumed.

The Mark-Up

You can create the underlying HTML for a graph in a number of ways. In this tutorial, we'll start with a **table**, because it will make the most sense visually if JavaScript or CSS is not applied. That's a big checkmark in the column for accessibility.

Quick! You've just been given some alarming figures. The population of tanned zombies is projected to spiral out of control in the next few years. The carbon tigers and blue monkeys are under immediate threat. Then the tanned zombies will probably come for us. But you're just a designer. What could you possibly do to help?

I know! You could make a Web page that illustrates our imminent demise with nice, calming, smoothly animated graphics!

To begin, let's put this data into a table, with columns for each year, and rows for the different species.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=1024">
    <title>Example 01: No CSS</title>
  </head>
```

```

<body>
  <div id="wrapper">
    <div class="chart">
      <h3>Population of endangered species from
2012 to 2016</h3>
      <table id="data-table" border="1"
cellpadding="10" cellspacing="0"
summary="The effects of the zombie
outbreak on the populations
of endangered species from 2012 to 2016">
        <caption>Population in
thousands</caption>
        <thead>
          <tr>
            <td>&nbsp;</td>
            <th scope="col">2012</th>
            <th scope="col">2013</th>
            <th scope="col">2014</th>
            <th scope="col">2015</th>
            <th scope="col">2016</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">Carbon Tiger</th>
            <td>4080</td>
            <td>6080</td>
            <td>6240</td>
            <td>3520</td>
            <td>2240</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>

```

```

        </tr>
        <tr>
            <th scope="row">Blue Monkey</th>
            <td>5680</td>
            <td>6880</td>
            <td>6240</td>
            <td>5120</td>
            <td>2640</td>
        </tr>
        <tr>
            <th scope="row">Tanned
Zombie</th>
            <td>1040</td>
            <td>1760</td>
            <td>2880</td>
            <td>4720</td>
            <td>7520</td>
        </tr>
    </tbody>
</table>
</div>
</div>
</body>
</html>

```

View the example below to see how it looks naked, with no CSS or JavaScript applied. The accessibility of this table will enable people using screen readers to understand the data and the underlying message, which is “Run for your life! The zombies are coming!”

	2012	2013	2014	2015	2016
Carbon Tiger	4080	6080	6240	3520	2240
Carbon Monkey	5680	6880	6240	5120	2240
Carbon Zomb	1040	1760	2880	4720	

Example 1: No CSS >

The easy part is now out of the way. Now, let's tap into the power of CSS and JavaScript (via jQuery) to really illustrate what the numbers are telling us. Technically, our aim is to create a graph that works in all modern browsers, from IE 8 on.

Did I say all modern browsers? IE 8 is lucky: it gets to hang out with the cool kids. Browsers that support CSS3 will get a few extra sprinkles.

“By Your Powers Combined...”

If you wish to summon Captain Planet⁹, you may¹⁰ have¹¹ to¹² look¹³ elsewhere¹⁴. If you want to learn how to combine CSS and jQuery to create a graph that illustrates our impending doom at the hands of a

growing army of zombies who prefer bronzer over brains, then read on.

The first thing to do is style our table with some basic CSS. This is a nice safety net for people who haven't enabled JavaScript in their browser.



2012	2013	2014
4080	6080	6240
5680	6880	6240
		3520
		5120
		4720
		2880

Example 2: CSS Only >

Getting Started In jQuery

We'll use jQuery to create our graph on the fly, separate from the original data table. To do this, we need to get the data out of the table and store it in a more usable format. Then, we can add to our document new elements that use this data in order to construct our graph.

Let's get started by creating our main `createGraph()` function. I've abbreviated some of the inner workings of this function so that you get a

clearer picture of the structure. Don't forget: you can always refer to the source code that comes with this tutorial.

Here's our basic structure:

```
// Wait for the DOM to load everything, just to be
safe
$(document).ready(function() {

    // Create our graph from the data table and specify
    a container to put the graph in
    createGraph('#data-table', '.chart');

    // Here be graphs
    function createGraph(data, container) {
        // Declare some common variables and container
        elements
        ...

        // Create the table data object
        var tableData = {
            ...
        }

        // Useful variables to access table data
        ...

        // Construct the graph
```

```

...

// Set the individual heights of bars
function displayGraph(bars) {
    ...
}

// Reset the graph's settings and prepare for
display
function resetGraph() {
    ...
    displayGraph(bars);
}

// Helper functions
...

// Finally, display the graph via reset function
resetGraph();
}
});

```

We pass two parameters to this function: The **data**, in the form of a **table** element; A **container** element, where we'd like to place our graph in the document.

Next up, we'll declare some variables to manage our data and container

elements, plus some timer variables for animation. Here's the code: //

Declare some common variables and container elements

```
var bars = [];  
var figureContainer = $('<div id="figure"></div>');  
var graphContainer = $('<div class="graph"></div>');  
var barContainer = $('<div class="bars"></div>');  
var data = $(data);  
var container = $(container);  
var chartData;  
var chartYMax;  
var columnGroups;
```

// Timer variables

```
var barTimer;  
var graphTimer;
```

Nothing too exciting here, but these will be very useful later.

Getting The Data

Besides simply displaying the data, a good bar chart should have a nice big title, clearly labelled axes and a color-coded legend. We'll need to strip the data out of the table and format it in a way that is more meaningful in a graph. To do that, we'll create a JavaScript object that stores our data in handy little functions. Let's give birth to our

```
tableData{}
```

 object: // Create table data object

```
var tableData = {
```

```
// Get numerical data from table cells
chartData: function() {
    ...
},
// Get heading data from table caption
chartHeading: function() {
    ...
},
// Get legend data from table body
chartLegend: function() {
    ...
},
// Get highest value for y-axis scale
chartYMax: function() {
    ...
},
// Get y-axis data from table cells
yLegend: function() {
    ...
},
// Get x-axis data from table header
xLegend: function() {
    ...
},
// Sort data into groups based on number of columns
columnGroups: function() {
    ...
}
}
```

We have several functions here, and they are explained in the code's comments. Most of them are quite similar, so we don't need to go through each one. Instead, let's pick apart one of them, **columnGroups**:

```
// Sort data into groups based on number of columns
columnGroups: function() {
    var columnGroups = [];
    // Get number of columns from first row of table
    body
    var columns = data.find('tbody tr:eq(0)
td').length;
    for (var i = 0; i < columns; i++) {
        columnGroups[i] = [];
        data.find('tbody tr').each(function() {
columnGroups[i].push($(this).find('td').eq(i).text());
        });
    }
    return columnGroups;
}
```

Here's how it breaks down:

- Create the **columnGroups[]** array to store the data; Get the number of columns by counting the table cells (**td**) in the first row; For each column, find the number of rows in the table body (**tbody**), and create another array to store the table cell data; Then loop through each row and grab the data from each table cell (via the jQuery **text()** function), and then add it to the table cell data array.

Once our object is full of juicy data, we can start creating the elements that make up our graph.

Using The Data

Using the jQuery `$.each` function, we can now loop through our data at any point and create the elements that make up our graph. One of the trickier bits involves inserting the bars that represent each species inside the yearly columns.

Here's the code:

```
// Loop through column groups, adding bars as we go
$.each(columnGroups, function(i) {
    // Create bar group container
    var barGroup = $('<div class="bar-group"></div>');
    // Add bars inside each column
    for (var j = 0, k = columnGroups[i].length; j < k; j++) {
        // Create bar object to store properties (label,
        height, code, etc.) and add it to array
        // Set the height later in displayGraph() to
        allow for left-to-right sequential display
        var barObj = {};
```

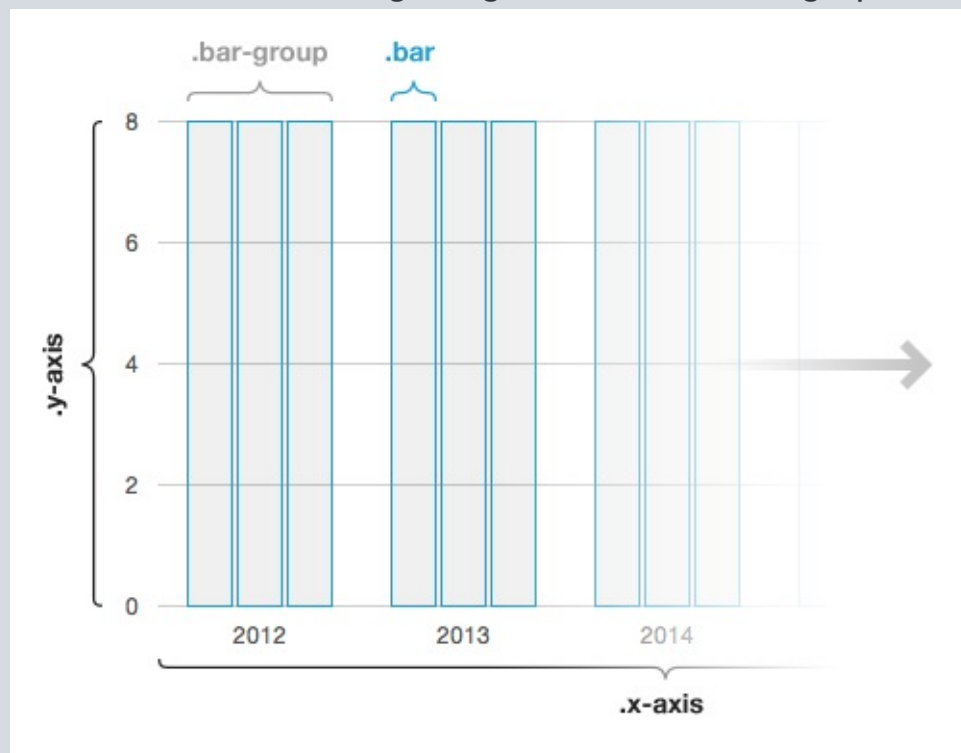


```

        barObj.label = this[j];
        barObj.height = Math.floor(barObj.label /
chartYMax * 100) + '%';
        barObj.bar = $('<div class="bar fig" + j + '">
<span>' + barObj.label + '</span></div>')
            .appendTo(barGroup);
        bars.push(barObj);
    }
    // Add bar groups to graph
    barGroup.appendTo(barContainer);
});

```

Excluding the headings, our table has five columns with three rows. For our graph, this means that for each column we create, three bars will appear in that column. The following image shows how our graph will be



constructed:

Breaking it down:

- For each column, create a container **div**; Loop inside each column to get the row and cell data; Create a bar object (**barObj{}**) to store the properties for each bar, such as its label, height and mark-up; Add the mark-up property to the column, applying a CSS class of **'.fig' + j** to color code each bar in the column, wrapping the label in a **span**; Add the object to our **bars[]** array so that we can access the data later; Piece it all together by adding the columns to a container element.

Bonus points if you noticed that we didn't set the height of the bars. This is so that we have more control later on over how the bars are displayed.

Now that we have our bars, let's work on labelling our graph. Because the code to display the labels is quite similar, talking you through all of it won't be necessary. Here's how we display the y-axis: // Add y-axis to graph

```
var yLegend    = tableData.yLegend();
var yAxisList  = $('<ul class="y-axis"></ul>');
$.each(yLegend, function(i) {
    var listItem = $('<li><span>' + this + '</span>'
    </li>')
    .appendTo(yAxisList);
});
```

```
yAxisList.appendTo(graphContainer);
```

This breaks down as follows:

- Get the relevant table data for our labels, Create an unordered list (`ul`) to contain our list items; Loop through the label data, and create a list item (`li`) for each label, wrapping each label in a `span`; Attach the list item to our list; Finally, attach the list to a container element.

By repeating this technique, we can add the legend, x-axis labels and headings for our graph.

Before we can display our graph, we need to make sure that everything we've done is added to our container element.

```
// Add bars to graph
barContainer.appendTo(graphContainer);

// Add graph to graph container
graphContainer.appendTo(figureContainer);

// Add graph container to main container
figureContainer.appendTo(container);
```

Displaying The Data

All that's left to do in jQuery is set the height of each bar. This is where our earlier work, storing the height property in a bar object, will come in handy.

We're going to animate our graph sequentially, one by one, uno por uno.

One possible solution is to use a callback function to animate the next bar when the last animation is complete. However, the graph would take too long to animate. Instead, our graph will use a timer function to display each bar after a certain amount of time, regardless of how long each bar takes to grow. Rad!

Here's the `displayGraph()` function: // Set the individual height of bars

```
function displayGraph(bars, i) {  
    // Changed the way we loop because of issues with  
    $.each not resetting properly  
    if (i < bars.length) {  
        // Animate the height using the jQuery animate()  
function  
        $(bars[i].bar).animate({  
            height: bars[i].height  
        }, 800);  
        // Wait the specified time, then run the  
displayGraph() function again for the next bar
```

```
barTimer = setTimeout(function() {  
    i++;  
    displayGraph(bars, i);  
}, 100);  
}  
}
```

What's that you say? “Why aren't you using the **\$.each** function like you have everywhere else?” Good question. First, let's discuss what the **displayGraph()** function does, then why it is the way it is.

The **displayGraph()** function accepts two parameters: The **bars** to loop through, An index (**i**) from which to start iterating (starting at **0**).

Let's break down the rest:

- If the value of **i** is less than the number of bars, then keep going; Get the current bar from the array using the value of **i**; Animate the height property (calculated as a percentage and stored in **bars[i].height**); Wait 100 milliseconds; Increment **i** by 1 and repeat the process for the next bar.

“So, why wouldn't you just use the **\$.each** function with a **delay()** before the animation?”

You could, and it would work just fine... the first time. But if you tried to reset the animation via the “Reset graph” button, then the timing events wouldn't clear properly and the bars would animate out of sequence.

Moving on, here's `resetGraph()`: // Reset graph settings and prepare for display

```
function resetGraph() {  
    // Stop all animations and set the bar's height to 0  
  
    $.each(bars, function(i) {  
        $(bars[i].bar).stop().css('height', 0);  
    });  
  
    // Clear timers  
    clearTimeout(barTimer);  
    clearTimeout(graphTimer);  
  
    // Restart timer  
    graphTimer = setTimeout(function() {  
        displayGraph(bars, 0);  
    }, 200);  
}
```

Let's break `resetGraph()` down: Stop all animations, and set the height of each bar back to 0; Clear out the timers so that there are no stray animations; Wait 200 milliseconds; Call `displayGraph()` to animate the first bar (at index 0).

Finally, call `resetGraph()` at the bottom of `createGraph()`, and watch the magic happen as we bask in the glory of our hard work.

Not so fast, sunshine! Before we go any further, we need to put some clothes on.

The CSS

The first thing we need to do is hide the original data table. We could do this in a number of ways, but because our CSS will load well before the JavaScript, let's do this in the easiest way possible: `#data-table` {

```
    display: none;
}
```

Done. Let's create a nice container area to put our graph in. Because a few unordered lists are being used to make our graph, we'll also reset the styles for those. Giving the `#figure` and `.graph` elements a `position: relative` is important because it will anchor the place elements exactly where we want in those containers.

```
/* Containers */

#wrapper {
    height: 420px;
```

```
    left: 50%;
    margin: -210px 0 0 -270px;
    position: absolute;
    top: 50%;
    width: 540px;
}

#figure {
    height: 380px;
    position: relative;
}

#figure ul {
    list-style: none;
    margin: 0;
    padding: 0;
}

.graph {
    height: 283px;
    position: relative;
}
```

Now for the legend. We position the legend right down to the bottom of its container (**#figure**) and line up the items horizontally: `/* Legend */`

```
.legend {
    background: #f0f0f0;
```



```

border-radius: 4px;
bottom: 0;
position: absolute;
text-align: left;
width: 100%;
}

.legend li {
display: block;
float: left;
height: 20px;
margin: 0;
padding: 10px 30px;
width: 120px;
}

.legend span.icon {
background-position: 50% 0;
border-radius: 2px;
display: block;
float: left;
height: 16px;
margin: 2px 10px 0 0;
width: 16px;
}

```

The x-axis is very similar to the legend. We line up the elements horizontally and anchor them to the bottom of its container (**.graph**): `/* x-axis */`

```
.x-axis {
  bottom: 0;
  color: #555;
  position: absolute;
  text-align: center;
  width: 100%;
}

.x-axis li {
  float: left;
  margin: 0 15px;
  padding: 5px 0;
  width: 76px;
}
```

The y-axis is a little more involved and requires a couple of tricks. We give it a **position: absolute** to break it out of the normal flow of content, but anchored to its container. We stretch out each **li** to the full width of the graph and add a border across the top. This will give us some nice horizontal lines in the background.

Using the power of negative margins, we can offset the numerical labels inside the **span** so that they shift up and to the left. Lovely!

```
/* y-axis */

.y-axis {
  color: #555;
```

```

    position: absolute;
    text-align: right;
    width: 100%;
}

.y-axis li {
    border-top: 1px solid #ccc;
    display: block;
    height: 62px;
    width: 100%;
}

.y-axis li span {
    display: block;
    margin: -10px 0 0 -60px;
    padding: 0 10px;
    width: 40px;
}

```

Now for the meat in our endangered species sandwich: the bars themselves. Let's start with the container element for the bars and the columns: `/* Graph bars */`

```

.bars {
    height: 253px;
    position: absolute;
    width: 100%;
    z-index: 10;
}

```

```
}  
  
.bar-group {  
  float: left;  
  height: 100%;  
  margin: 0 15px;  
  position: relative;  
  width: 76px;  
}
```

Nothing too complicated here. We're simply setting some dimensions for the container, and setting a **z-index** to make sure it appears in front of the y-axis markings.

Now for each individual **.bar**: **.bar** {

```
  border-radius: 3px 3px 0 0;  
  bottom: 0;  
  cursor: pointer;  
  height: 0;  
  position: absolute;  
  text-align: center;  
  width: 24px;  
}
```

```
.bar.fig0 {  
  left: 0;  
}
```

```
.bar.fig1 {
```

```
    left: 26px;
}

.bar.fig2 {
    left: 52px;
}
```

The main styles to note here are:

- **position: absolute** and **bottom: 0**, which means that the bars will be attached to the bottom of our graph and grow up; the bar for each species (**.fig0**, **.fig1** and **.fig2**), which will be positioned within **.bar-group**.

Now, why don't we minimize the number of sharp edges on any given page by using the **border-radius** property to round the edges of the top-left and top-right corners of each bar? OK, so **border-radius** isn't really necessary, but it adds a nice touch for browsers that support it. Thankfully, the latest versions of the most popular browsers do support it.

Because we've placed the values from each table cell in each bar, we can add a neat little pop-up that appears when you hover over a bar:

```
.bar span {
    background: #fefefe url(../images/info-bg.gif) 0
100% repeat-x;
    border-radius: 3px;
```

```
left: -8px;
display: none;
margin: 0;
position: relative;
text-shadow: rgba(255, 255, 255, 0.8) 0 1px 0;
width: 40px;
z-index: 20;

-webkit-box-shadow: rgba(0, 0, 0, 0.6) 0 1px 4px;
box-shadow: rgba(0, 0, 0, 0.6) 0 1px 4px;
}

.bar:hover span {
display: block;
margin-top: -25px;
}
```

First, the pop-up is hidden from view via **display: none**. Then, when a **.bar** element is hovered over, we've set **display: block** to bring it into view, and set a negative **margin-top** to make it appear above each bar.

The **text-shadow**, **rgba** and **box-shadow** properties are currently supported by most modern browsers as is. Of these modern browsers, only Safari requires a vendor prefix (**-webkit-**) to make **box-shadow** work. Note that these properties are simply enhancements to our graph and aren't required to understand it. Our baseline of Internet Explorer 8 simply ignores them.

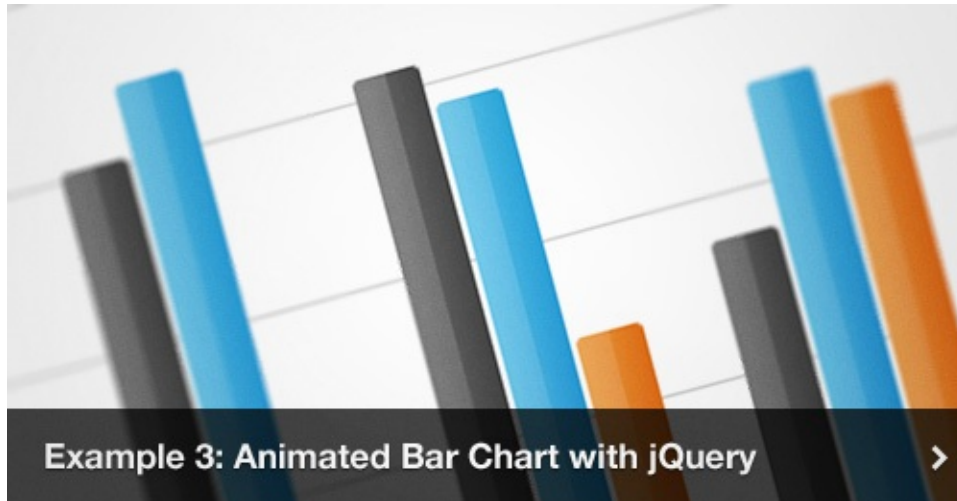
Our final step in bringing everything together is to color code each bar:

```
.fig0 {  
    background: #747474 url(../images/bar-01-bg.gif) 0  
0 repeaty;  
}  
  
.fig1 {  
    background: #65c2e8 url(../images/bar-02-bg.gif) 0  
0 repeaty;  
}  
  
.fig2 {  
    background: #eea151 url(../images/bar-03-bg.gif) 0  
0 repeaty;  
}
```

In this example, I've simply added a **background-color** and a **background-image** that tiles vertically. This will update the styles for the bars and the little icons that represent them in the legend. Nice.

And, believe it or not, that is it!

The Finished Product



That about wraps it up. I hope we've done enough to alert the public to the dangers of zombie over-population. More than that, however, I hope you've gained something useful from this tutorial and that you'll continue to push the boundaries of what can be done in the browser—especially with proper Web standards and without the use of third-party plug-ins. If you've got ideas on how to extend or improve anything you've seen here, find me on Twitter [@derek_mack¹⁵](#).

Bonus: Unleashing The Power Of CSS3

This bonus is not as detailed as our main example. It serves mainly as a showcase of some features being considered in the CSS3 specification.

Because support for CSS3 properties is currently limited, so is their use. Although some of the features mentioned here are making their way into

other Web browsers, Webkit-based ones such as Apple Safari and Google Chrome are leading the way.

We can actually create our graph using no images at all, and even animate the bars using CSS instead of jQuery.

Let's start by removing the background images from our bars, replacing them with the **-webkit-gradient** property: `.fig0 {`

```
    background: -webkit-gradient(linear, left top,
    right top, color-stop(0.0, #747474), color-stop(0.49,
    #676767), color-stop(0.5, #505050), color-stop(1.0,
    #414141));
}
```

```
.fig1 {
    background: -webkit-gradient(linear, left top,
    right top, color-stop(0.0, #65c2e8), color-stop(0.49,
    #55b3e1), color-stop(0.5, #3ba6dc), color-stop(1.0,
    #2794d4));
}
```

```
.fig2 {
    background: -webkit-gradient(linear, left top,
    right top, color-stop(0.0, #eea151), color-stop(0.49,
    #ea8f44), color-stop(0.5, #e67e28), color-stop(1.0,
    #e06818));
}
```

We can do the same with our little number pop-ups: `.bar span {`
 `background: -webkit-gradient(linear, left top, left`
 `bottom, color-stop(0.0, #fff), color-stop(1.0,`
 `#e5e5e5));`
 `...`
}

For more information on Webkit gradients, check out the Surfin' Safari¹⁶ blog.

Continuing with the pop-ups, let's introduce **-webkit-transition**. CSS transitions are remarkably easy to use and understand. When the browser detects a change in an element's property (height, width, color, opacity, etc.), it will transition to the new property.

Again, refer to Surfin' Safari¹⁷ for more information on **-webkit-transition** and CSS3 animation.

Here's an example:

```
.bar span {
    background: -webkit-gradient(linear, left top, left
bottom, color-stop(0.0, #fff), color-stop(1.0,
#e5e5e5));
    display: block;
    opacity: 0;
```

```
    -webkit-transition: all 0.2s ease-out;
}

.bar:hover span {
    opacity: 1;
}
```

When you hover over the bar, the margin and opacity of the pop-up will change. This triggers a transition event according to the properties we have set. Very cool.

Thanks to **-webkit-transition**, we can simplify our JavaScript functions a bit: // Set individual height of bars

```
function displayGraph(bars, i) {
    // Changed the way we loop because of issues with
    $.each not resetting properly
    if (i < bars.length) {
        // Add transition properties and set height via
        CSS
        $(bars[i].bar).css({'height': bars[i].height, '-
        webkit-transition': 'all 0.8s ease-out'});
        // Wait the specified time, then run the
        displayGraph() function again for the next bar
        barTimer = setTimeout(function() {
            i++;
            displayGraph(bars, i);
```

```

        }, 100);
    }
}
// Reset graph settings and prepare for display
function resetGraph() {
    // Set bar height to 0 and clear all transitions
    $.each(bars, function(i) {
        $(bars[i].bar).stop().css({'height': 0, '-
webkit-transition': 'none'});
    });

    // Clear timers
    clearTimeout(barTimer);
    clearTimeout(graphTimer);

    // Restart timer
    graphTimer = setTimeout(function() {
        displayGraph(bars, 0);
    }, 200);
}

```

Here are the main things we've changed: Set the height of the bars via the jQuery **css()** function, and allowed CSS transitions to take care of the animation; When resetting the graph, turned transitions off so that the height of the bars is instantly set to 0.

Check out the example¹⁸ if you have the latest version of Safari or Chrome installed.

ULTRA-MEGA WEBKIT BONUS: NOW IN 3-D!

For a sneak peek of what the future holds, check out a little experiment¹⁹ that I put together, with a 3-D effect and CSS transforms. Again, it requires the latest versions of Safari or Chrome: As in our previous Webkit example, there are **no images**, and **all animation is handled via CSS**. Kiss my face!²⁰

I can't tell you what to do with all this information. But I do caution you about the potential misuse of your new powers. In the words of our friend Captain Planet, "The power is yours!"

Use it wisely. 🐼

—

1. <http://startupista.com/corporate-bullshit-generator/>
2. <http://www.apple.com/hotnews/thoughts-on-flash/>
3. <http://coding.smashingmagazine.com/2009/04/22/progressive-enhancement-what-it-is-and-how-to-use-it/>

4. http://provide.smashingmagazine.com/graph_tut_files/ex_03.html
5. <http://www.farmville.com/>
6. http://en.wikipedia.org/wiki/Comparison_of_HTML_editors
7. <http://jquery.com/>
8. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/mspaint_overview.mspx?mfr=true
9. http://en.wikipedia.org/wiki/Captain_Planet
10. <http://en.wikipedia.org/wiki/Planeteer#Kwame>
11. <http://en.wikipedia.org/wiki/Planeteer#Wheeler>
12. <http://en.wikipedia.org/wiki/Planeteer#Linka>
13. <http://en.wikipedia.org/wiki/Planeteer#Gi>
14. <http://en.wikipedia.org/wiki/Planeteer#Ma-Ti>

15. http://twitter.com/derek_mack
16. <http://webkit.org/blog/175/introducing-css-gradients/>
17. <http://webkit.org/blog/138/css-animation/>
18. http://provide.smashingmagazine.com/graph_tut_files/ex_04.html
19. http://provide.smashingmagazine.com/graph_tut_files/ex_05.html
20. <http://www.youtube.com/watch?v=bkcCBXZHfPw>

A Beginner's Guide To jQuery-Based JSON API Clients

BY BEN HOWDLE 

Are you fascinated by dynamic data? Do you go green with envy when you see tweets pulled magically into websites? Trust me, I've been there.

The goal of this tutorial is to create a simple Web app for grabbing movie posters from TMDb¹. We'll use jQuery and the user's input to query a JSON-based API and deal with the returned data appropriately.

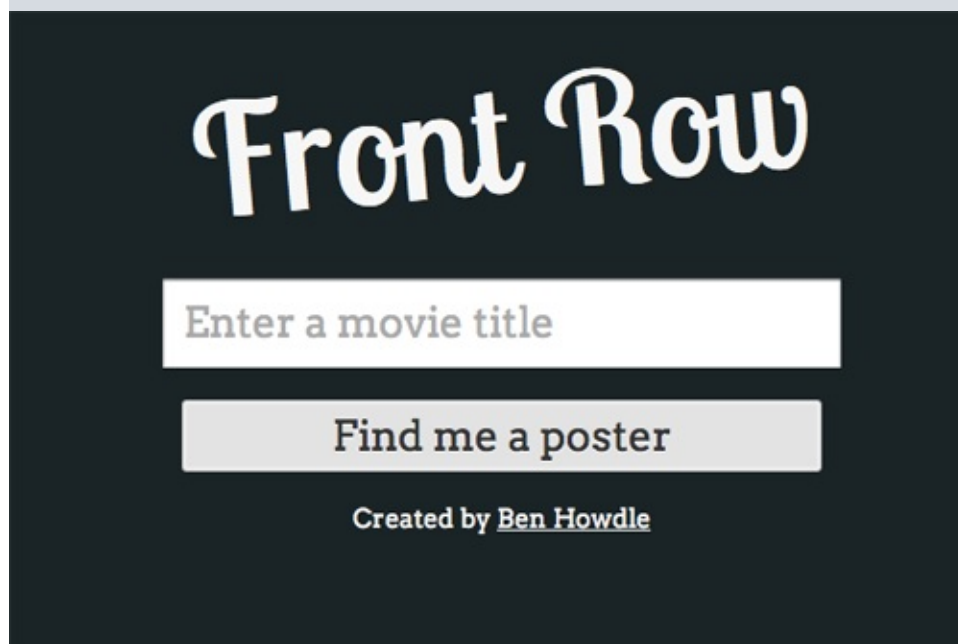
I hope to convince you that APIs aren't scary and that most of the time they can be a developer's best friend.

APIs Are The Future But, More Importantly, The Present

JSON-based APIs are a hot property on the Web right now. I cannot remember the last time I went onto a blog or portfolio without seeing the owner's tweets or Facebook friends staring back at me. This interactivity makes the Web an exciting place. The only limit seems to be people's imagination. As demonstrated by everything from pulled tweets to a self-

aware exchange-rates API², data is currently king, and websites are swapping it freely.

Developers are allowing us to get at their data much more openly now; no longer is everything under lock and key. Websites are proud to have you access their data and, in fact, encourage it. Websites such as TMDb³ and LastFM⁴ allow you to build entirely separate applications based on the data they've spent years accumulating. This openness and receptiveness are fostering an intertwined network of users and their corresponding actions.



This chapter is aimed at people who are competent in HTML and CSS and have basic knowledge of jQuery techniques. We won't delve deep into advanced JavaScript techniques, but will rather help the beginner who wants to create more complex Web tools.

APIS IN A NUTSHELL

In basic terms, an API enables you to access a website's data without going near its databases. It gives us a user-friendly way to read and write data to and from a website's databases.

Sure, Great, But What Code Do I Need?

Like many developers, I bounce merrily between back-end and front-end development, and I am as happy working in PHP as in jQuery. It just depends on which hat I'm wearing that day.

Because this chapter is mainly about jQuery-based JSON API clients, we'll focus on client-side code (i.e. jQuery).

When dealing with APIs, and armed with jQuery, one is more likely to encounter JSON.

PLAYER 1: JSON

JSON⁵ (or JavaScript Object Notation) is a lightweight, easy and popular way to exchange data. jQuery is not the only tool for manipulating and interfacing with JSON; it's just my and many others' preferred method.

A lot of the services we use everyday have JSON-based APIs: Twitter, Facebook and Flickr all send back data in JSON format.

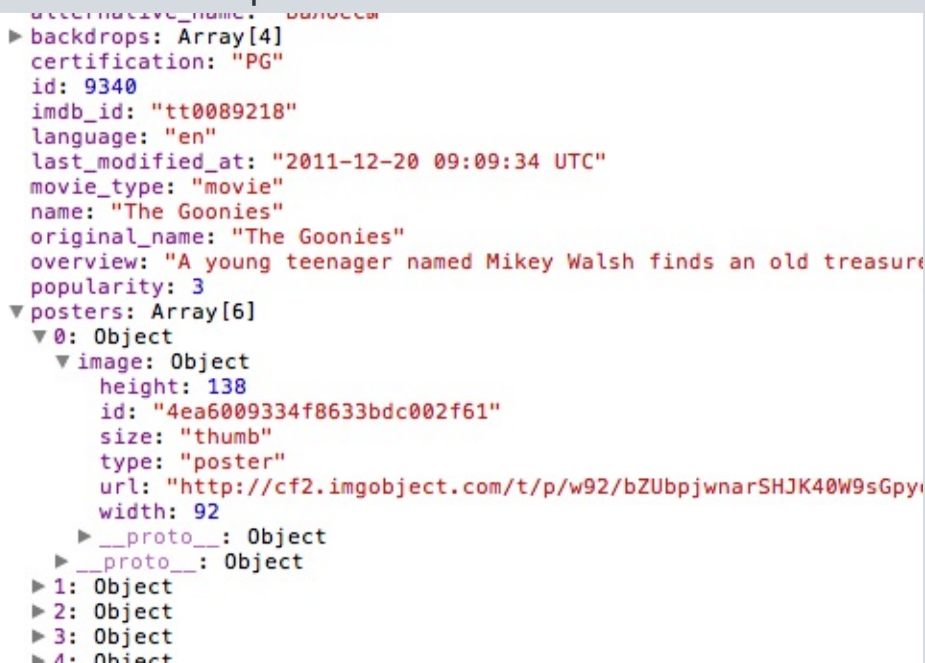
A JSON response from an API looks like this: (`[{"score": null, "popularity": 3, "translated": true, "adult": false, "language": "en", "original_name": "The Goonies", "name": "The Goonies", "alternative_name": "I Goonies", "movie_type": "movie", "id": 9340, "imdb_id": "tt0089218", "url": "http://www.themoviedb.org/movie/9340", "votes": 16, "rating": 9.2, "certification": "PG", "overview": "A young teenager named Mikey Walsh finds an old treasure map in his father's attic. Hoping to save their homes from demolition, Mikey and his friends Data Wang, Chunk Cohen, and Mouth Devereaux runs off on a big quest to find the secret stash of the pirate One-Eyed Willie."}, {"released": "1985-06-07", "posters": [{"image": {"type": "poster", "size": "original", "height": 1500, "width": 1000}]}]`)

```

"url":"http://cf1.imgobject.com/posters/76b/4d406d767b9aa15bb500276b",
goonies-original.jpg",
"id":"4d406d767b9aa15bb500276b"}]], "backdrops":
[{"image":
{"type":"backdrop", "size":"original", "height":1080, "width":1920,
"url":"http://cf1.imgobject.com/backdrops/242/4d690e167b9aa13631001242",
goonies-original.jpg",
"id":"4d690e167b9aa13631001242"}]], "version":3174, "last_modified_at":
"2011-12-20 09:09:34 UTC"}])

```

A bit of a mess, right? Compare this to the same JSON viewed in Google Chrome's developer console:



```

    alternative_name: "Goonies"
    ▶ backdrops: Array[4]
    certification: "PG"
    id: 9340
    imdb_id: "tt0089218"
    language: "en"
    last_modified_at: "2011-12-20 09:09:34 UTC"
    movie_type: "movie"
    name: "The Goonies"
    original_name: "The Goonies"
    overview: "A young teenager named Mikey Walsh finds an old treasure map that leads to a hidden world of adventure."
    popularity: 3
    ▼ posters: Array[6]
    ▼ 0: Object
    ▼ image: Object
    height: 138
    id: "4ea6009334f8633bdc002f61"
    size: "thumb"
    type: "poster"
    url: "http://cf2.imgobject.com/t/p/w92/bZUbpjwnarSHJK40W9sGpyr"
    width: 92
    ▶ __proto__: Object
    ▶ __proto__: Object
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
    ▶ 4: Object

```

The JSON response is accessible via a jQuery function, allowing us to manipulate, display and, more importantly, style it however we want.

PLAYER 2: JQUERY

Personally, I'd pick jQuery over JavaScript any day of the week. jQuery makes things a lot easier for the beginner Web developer who just wants stuff to work right off the bat. I use it every day. If I had to complete the same tasks using native Javascript, my productivity would grind right down. In my opinion, JavaScript is for people who want a deeper understanding of the scripting language and the DOM itself. But for simplicity and ease of use, jQuery is where it's at.

In essence, jQuery is a JavaScript library, with handy functions like **getJSON**. This particular function will be the glue that holds our API client together.

The Goal: A jQuery-Based JSON API Client

I recently submitted to An Event Apart⁶ the Web app that we're about to go through now. It's called Front Row⁷.

The idea of the Web app is to take a movie title inputted by the user, run it through TMDb⁸'s API, and return the relevant poster. The user could then share it or save it to their computer.

The Web app is split into HTML, CSS and jQuery. We'll focus on the

The web app is split into HTML, CSS and jQuery. We'll focus on the jQuery, because that's where the magic happens.

THE HTML

Below is the basic structure of the Web app. Nothing special here.

```
<!DOCTYPE html>
<html>
<head>
  <meta name="author" content="Ben Howdle and Dan
Matthew">
  <meta name="description" content="A responsive
movie poster grabber">
  <title>Front Row by Ben Howdle</title>
  <meta name="viewport" content="width=device-width,
minimum-scale=1.0, maximum-scale=1.0">
  <script
src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.6.2.min.js"></script>
  <!--jQuery, linked from a CDN-->
  <script src="scripts.js"></script>
  <script type="text/javascript"
src="http://use.typekit.com/oYa4cmx.js"></script>
  <script
type="text/javascript">try{Typekit.load();}catch(e){}
</script>
```

```
<link rel="stylesheet" href="style.css" />
</head>
<body>
<div class="container">
  <header>
    <h1>Front Row</h1>
  </header>
  <section id="fetch">
    <input type="text" placeholder="Enter a movie
title" id="term" />
    <button id="search">Find me a poster</button>
  </section>
  <section id="poster">
  </section>
  <footer>
    <p>Created by <a
href="http://twostepmedia.co.uk">Ben Howdle</a></p>
  </footer>
</div>
</body>
</html>
```

All we've got is a bit of Twitter self-indulgence, an input field and a submission button. Done!

The CSS is a bit off topic for this chapter, so I'll leave it to you to inspect the elements of interest on the live website.

THE JQUERY

```
$(document).ready(function(){

    //This is to remove the validation message if no
    poster image is present

    $('#term').focus(function(){
        var full = $("#poster").has("img").length ? true
: false;
        if(full == false){
            $('#poster').empty();
        }
    });
});
```

I like validation messages to disappear when the user starts retyping in an input field. The script below checks whether an image is present (i.e. a movie poster), and if not, empties the container of the validation message once the input field gains focus.

```
//function definition
```

```
var getPoster = function(){
```

```
    //Grab the movie title and store it in a
```


variable

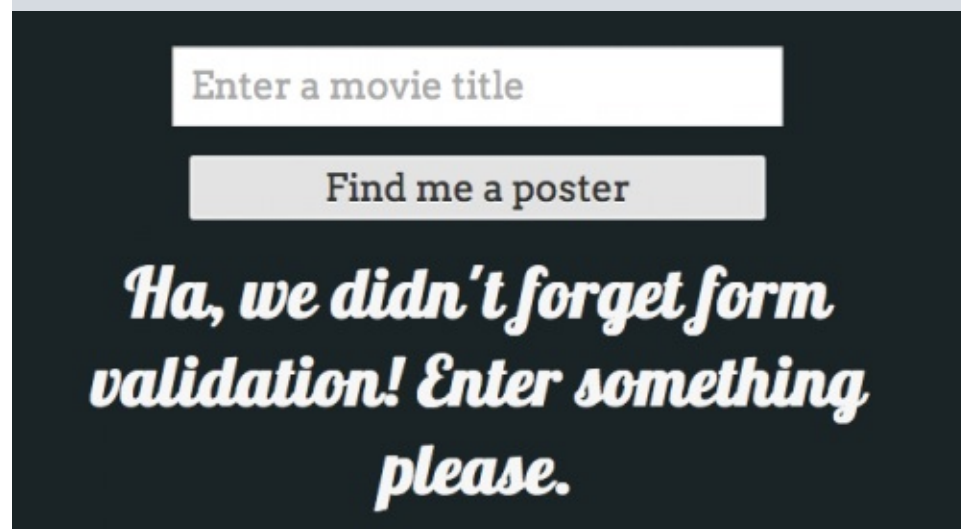
```
var film = $('#term').val();

//Check if the user has entered anything

if(film == ''){

    //If the input field was empty, display a
message

    $('#poster').html("<h2 class='loading'>Ha!
We haven't forgotten to validate the form! Please
enter something.</h2>");
```



The reason why we store the main code that retrieves the data in a function will become clear later on (mainly, it's for DRY programming⁹).

We then store the value of the input in a variable, so that when we use it again in the code, the jQuery doesn't have to rescan the DOM.

Basic validation is performed on the input, checking that something has actually been entered in the field.

In an attempt at humor on my part, I display a message warning the user that they haven't entered anything and asking them to please do so.

```
} else {  
  
    //They must have entered a value, carry on  
    with API call, first display a loading message to  
    notify the user of activity  
  
    $('#poster').html("<h2  
class='loading'>Your poster is on its way!</h2>");
```

If the input contains a value, we then process the user's request. Another message is displayed, letting the user know that something is happening.

```
$.getJSON("http://api.themoviedb.org/2.1/Movie.search/  
en/json/23afca60ebf72f8d88cdcae2c4f31866/" + film + "?
```

```
callback=?", function(json) {  
  
    //TMDb is nice enough to return a  
    message if nothing was found, so we can base our if  
    statement on this information  
  
    if (json != "Nothing found."){  
  
        //Display the poster and a message  
        announcing the result  
  
        $('#poster').html('<h2  
class="loading">Well, gee whiz! We found you a poster,  
skip!</h2><img id="thePoster" src=' +  
json[0].posters[0].image.url + ' />');
```



Here we get to the meat of our API client. We use jQuery's `getJSON`¹⁰ function, which, by definition, loads “JSON-encoded data from the server using a `GET` HTTP request.”

We then use the API's URL, supplied in this case by TMDb¹¹. As with many other APIs, you have to register your application in order to receive a key (a 30-second process). We insert the API key (`23afca60ebf72f8d88cdcae2c4f31866`) into the URL and pass the user's movie title into the URL as a search parameter.

One thing to mention is that appending `callback=?` to the end of the URL enables us to make cross-domain JSON and AJAX calls. Don't forget this, otherwise the data will be limited to your own domain! This method uses what's called JSONP (or JSON with padding), which basically allows a script to fetch data from another server on a different domain. To do this, we must specify the callback above when jQuery loads the data. It replaces the `?` with our custom function's name, thereby allowing us to make cross-domain calls with ease.

In the function's callback, we have put the word `json` (which holds our retrieved data), but we could have put `data` or `message`.

The next check is to see whether any data was returned. TMDb is kind enough to supply us with a message of “Nothing found” when it can't find anything. So, we've based our `if` statement on this string's value.

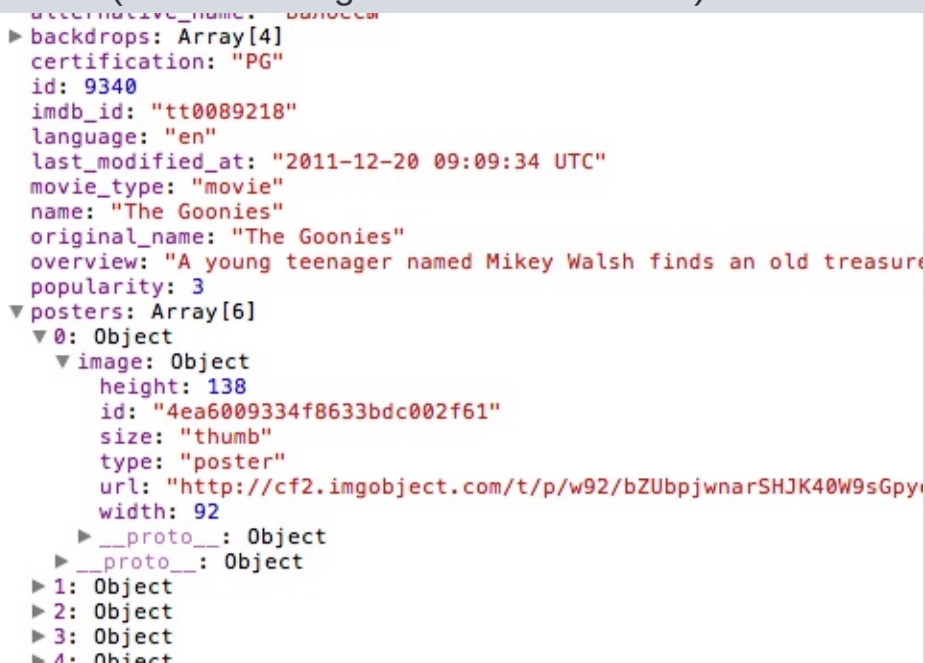
This check is API-specific. Usually if no results are found, we would expand the object to find a property named `length`, which would tell us how many results were returned. If this happens, the code might look something like this: `if (json.length !== 0){`

As a side note, before writing even a line of code in the callback function of the JSON call, we should become familiar with the results returned in Chrome's console or in Firebug. This would tell us exactly what to check for in `if` statements and, more importantly, what path to take to grab the data we want.

Let's add `console.log(json);`, like so:

```
$.getJSON("http://api.themoviedb.org/2.1/Movie.search/en/json/
+ film + "?callback=?", function(json) {
    console.log(json);
```

This will output something like the following in the console of your favorite browser (click the image to see the full size):



```
alternative_name: "Goonies"
backdrops: Array[4]
certification: "PG"
id: 9340
imdb_id: "tt0089218"
language: "en"
last_modified_at: "2011-12-20 09:09:34 UTC"
movie_type: "movie"
name: "The Goonies"
original_name: "The Goonies"
overview: "A young teenager named Mikey Walsh finds an old treasure
popularity: 3
posters: Array[6]
  0: Object
    image: Object
      height: 138
      id: "4ea6009334f8633bdc002f61"
      size: "thumb"
      type: "poster"
      url: "http://cf2.imgobject.com/t/p/w92/bZUbpjwnarSHJK40W9sGpy
      width: 92
    __proto__: Object
  1: Object
  2: Object
  3: Object
  4: Object
```

The last line of this code outputs our poster. We display another message to the user saying that we've found a result, and then proceed to show the image.

Let's spend a moment figuring out how we got to the poster images using the line `json[0].posters[0].image.url`.

The reason we use `json[0]` is that — since we want to display only one poster, and knowing how relevant TMDb's results are — we can gamble on the first result. We then access the `posters` array like so:

`json[0].posters[0]`. Chrome even tells us that `posters` is an array, so we know what we're dealing with. Again, we access the first value of the array, having faith that it will be most relevant. It then tells us that `image` is an object, so we can access it like so:

`json[0].posters[0].image`. By expanding our object further, we see that `image` contains a property named `url`. Jackpot! This contains a direct image link, which we can use in the `src` attribute of our image element.

```
} else {
```

```
    //If nothing is found, I attempt humor by
    displaying a Goonies poster and confirming that their
    search returned no results.
```

```
$.getJSON("http://api.themoviedb.org/2.1/Movie.search/
en/json/
23afca60ebf72f8d88cdcae2c4f31866/goonies?callback=?",
function(json) {
```

```
    $('#poster').html('<h2 class="loading">We're
afraid nothing was found for that search. Perhaps you
were looking for The Goonies?</h2><img id="thePoster"
src=' + json[0].posters[0].image.url + ' />');
```

```
});  
}
```

Having determined that the API has no results for the user, we could display an error message. But this being a movie-related Web app, let's give the user a preset poster of The Goonies and let them know we couldn't find anything. We'll use the exact same **src** attribute for the image that we used before, this time with **goonies** hardcoded into the API call's URL.

```
});  
  
    }  
  
    return false;  
}
```

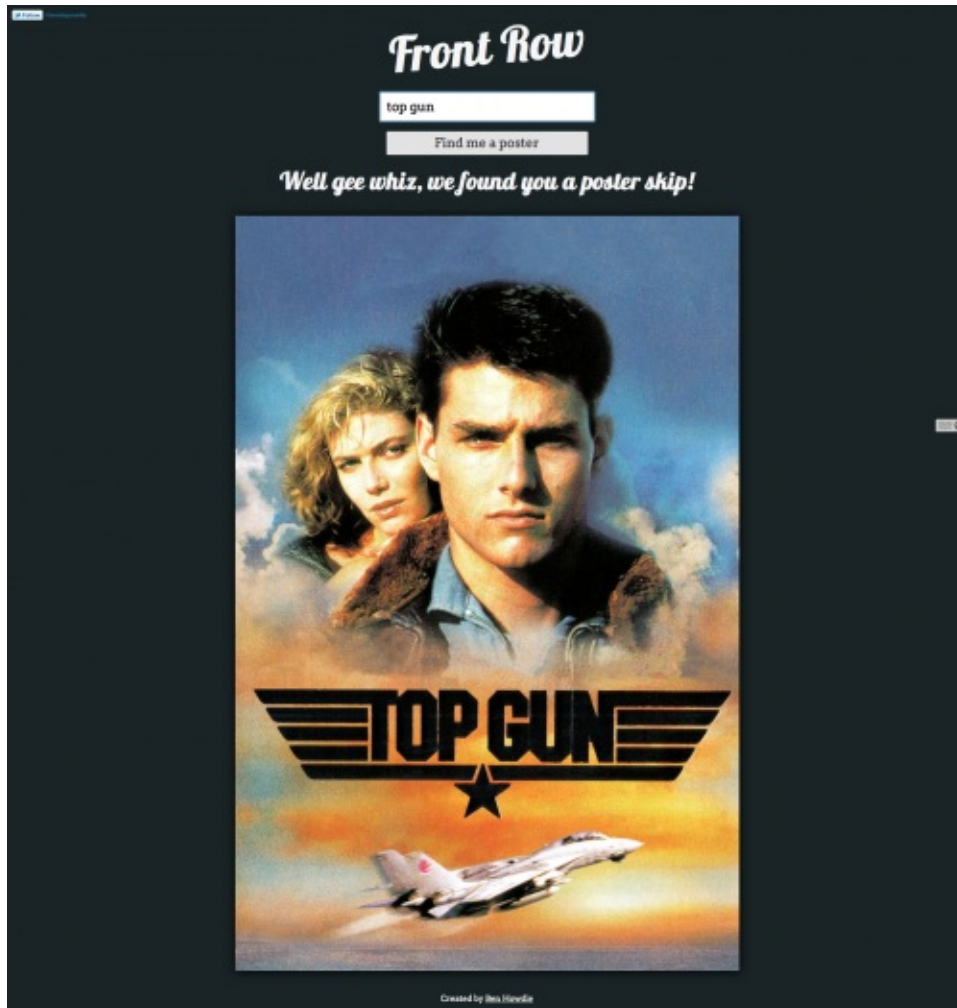
//Because we've wrapped the JSON code in a function, we can call it on mouse click or on a hit of the Return button while in the input field

```
$('#search').click(getPoster);  
  
$('#term').keyup(function(event){  
  
    if(event.keyCode == 13){
```



```
        getPoster();  
  
    }  
  
    });  
  
});
```

It is now clear why we wrapped our JSON call in a function: doing so allows us to run the function when the user hits the submission button or presses Enter while in the input field.



The Full Code

THE HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta name="author" content="Ben Howdle and Dan
```

```
Matthew">
  <meta name="description" content="A responsive
movie poster grabber">
  <title>Front Row by Ben Howdle</title>
  <meta name="viewport" content="width=device-width,
minimum-scale=1.0, maximum-scale=1.0">
  <script
src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.6.2.min.js"></script>
    <!--jQuery, linked from a CDN-->
    <script src="scripts.js"></script>
    <script type="text/javascript"
src="http://use.typekit.com/oYa4cmx.js"></script>
    <script
type="text/javascript">try{Typekit.load();}catch(e){}
</script>
    <link rel="stylesheet" href="style.css" />
</head>
<body>
<div class="container">
  <header>
    <h1>Front Row</h1>
  </header>
  <section id="fetch">
    <input type="text" placeholder="Enter a movie
title" id="term" />
    <button id="search">Find me a poster</button>
  </section>
  <section id="poster">
  </section>
```

```
<footer>
    <p>Created by <a
href="http://twostepmedia.co.uk">Ben Howdle</a></p>
</footer>
</div>
</body>
</html>
```

THE JQUERY

```
$(document).ready(function(){

    $('#term').focus(function(){
        var full = $("#poster").has("img").length ? true
: false;
        if(full == false){
            $('#poster').empty();
        }
    });

    var getPoster = function(){

        var film = $('#term').val();

        if(film == ''){

            $('#poster').html("<h2 class='loading'>Ha!
We haven't forgotten to validate the form! Please
```

```

enter something.</h2>");

    } else {

        $('#poster').html("<h2
class='loading'>Your poster is on its way!</h2>");

$.getJSON("http://api.themoviedb.org/2.1/Movie.search/
en/json/
23afca60ebf72f8d88cdcae2c4f31866/" + film + "?
callback=?", function(json) {
    if (json != "Nothing found."){
        $('#poster').html('<h2
class="loading">Well, gee whiz! We found you a poster,
skip!</h2><img id="thePoster" src=' +
json[0].posters[0].image.url + ' />');
    } else {

$.getJSON("http://api.themoviedb.org/2.1/Movie.search/
en/json/
23afca60ebf72f8d88cdcae2c4f31866/goonies?callback=?",
function(json) {

    console.log(json);
    $('#poster').html('<h2
class="loading">We're afraid nothing was found for
that search. Perhaps you were looking for The Goonies?
</h2><img id="thePoster" src=' +
json[0].posters[0].image.url + ' />');
    });

```

```
        }
    });

    }

    return false;
}

$('#search').click(getPoster);
$('#term').keyup(function(event){
    if(event.keyCode == 13){
        getPoster();
    }
});
});
```

Conclusion

That's it: a handy method of reading data from a remote API with jQuery, and manipulating the JSON output to fit our needs.

Every API is different, and every one returns different results in a different structure — it's all part of the fun! So, get used to using **console.log()**, and familiarize yourself with the results set before trying to access it with code or using it in your application.

Start with something practical and entertaining: build a checkin checker with Gowalla's API; visualize trends with Twitter's API; or make a face-recognition app with Face.com's API.

APIs are fun. By definition, the data they bring to the page is dynamic, not static. 🐙

FURTHER RESOURCES

- “How to Use JSON APIs With jQuery¹²,” Joel Sutherland, HiFi “How to Use jQuery With a JSON Flickr Feed to Display Photos¹³,” Richard Shepherd “Bing Instant Search With jQuery and Ajax¹⁴,” Srinivas Tamada, 9Lessons

— <http://www.themoviedb.org/>

<http://joscrowcroft.github.com/open-exchange-rates/>

<http://www.themoviedb.org/>

<http://www.last.fm/>

<http://www.json.org/>

<http://10k.aneventapart.com/>

<http://10k.aneventapart.com/Entry/Details/550>

<http://www.themoviedb.org/>

http://en.wikipedia.org/wiki/Don't_repeat_yourself

<http://api.jquery.com/jQuerygetJSON/>

<http://api.themoviedb.org/2.1/>

<http://www.gethifi.com/blog/how-to-use-json-apis-with-jquery>

<http://richardshepherd.com/how-to-use-jquery-with-a-json-flickr-feed-to-display-photos/>

<http://www.9lessons.info/2011/02/bing-instant-search-with-jquery-and.html>

How To Build A RealTime Commenting System

BY PHIL LEGGETTER 

The Web has become increasingly interactive over the years. This trend is set to continue with the next generation of applications driven by the **realtime Web**. Adding realtime functionality to an application can result in a more interactive and engaging user experience. However, setting up and maintaining the server-side realtime components can be an unwanted distraction. But don't worry, there is a solution.

Cloud hosted Web services and APIs have come to the rescue of many a developer over the past few years, and realtime functionality is no different. The focus at Pusher¹, for example, is to let you concentrate on building your realtime Web applications by offering a hosted API which makes it quick and easy to add scalable realtime functionality to Web and mobile apps. In this tutorial, I'll show how to convert a basic blog commenting system into a realtime engaging experience where you'll see a comment made in one browser window "magically" appear in a second window.

Why Should We Care About The RealTime Web?

Although the RealTime Web² is a relatively recent mainstream phrase, realtime Web technologies have been around for over 10 years. They were mainly used by companies building software targeted at the financial services sector or in Web chat applications. These initial solutions were classed as "hacks". In 2006 these solutions were given an umbrella term called Comet³, but even with a defined name the solutions were still considered hacks. **So, what's changed?**

In my opinion there are a number of factors that have moved realtime Web technologies to the forefront of Web application development.

SOCIAL MEDIA DATA

Social media, and specifically Twitter, has meant that more and more data is becoming instantly available. Gone are the days where we have to wait an eternity for Google to find our data (blog posts, status updates, images). There are now platforms that not only make our data instantly discoverable but also instantly deliver it to those who have declared an interest. This idea of Publish/Subscribe⁴ is core to the realtime Web, especially when building Web applications.

INCREASED USER EXPECTATIONS

As more users moved to using applications such as Twitter and

Facebook, and the user experiences that they deliver, their perception of what can be expected from a Web application changed. Although applications had become more dynamic through the use of JavaScript, the experiences were seldom truly interactive. Facebook, with its realtime wall (and later other realtime features) and Twitter with its activity stream centric user interface, and focus on conversation, demonstrated how Web applications could be highly engaging.

WEBSOCKETS

HTML



Earlier on I stated that previous solutions to let servers instantly push data to Web browsers were considered "hacks". But this didn't remove the fact that there was a requirement to be able to do this in a cross-browser and standardised way. Our prayers have finally been answered with **HTML5** WebSockets⁵. WebSockets represent a standardized API⁶ and protocol⁷ that allows realtime server and client (web browser) two

way communication over a single connection. Older solutions could only achieve two-way communication using two connections so the fact the WebSockets use a single connection is actually a big deal. It can be a massive resource saving to the server and client, with the latter being particularly important for mobile devices where battery power is extremely valuable.

HOW ARE REALTIME TECHNOLOGIES BEING USED?

Realtime Web technologies are making it possible to build all sorts of engaging functionality, leading to improved user experiences. Here are a handful of common use cases: **Realtime Stats** – The technology was first used in finance to show stock exchange information so it's no surprise that the technology is now used more than ever. It's also highly beneficial to sports, betting and analytics.

- **Notifications** – when something a user is interested in becomes available or changes.
- **Activity Streams** – streams of friend or project activity. This is commonly seen in apps like Twitter, Facebook, Trello, Quora and many more.
- **Chat** – the 101 or realtime Web technology but still a very valuable

function. It helps delivery instant interaction between friends, work colleagues, customers and customer service *etc.*

- **Collaboration** – Google docs offers this kind of functionality within its docs, spreadsheets and drawing applications and we're going to see similar use cases in many more applications.
- **Multiplayer Games** – The ability to instantly deliver player moves, game state changes and score updates is clearly important to multiplayer gaming.

In the rest of this tutorial I'll cover building a basic blog commenting system, how to progressively enhance it using jQuery and finally I'll also progressively enhance it using the realtime Web service I work for, Pusher, which will demonstrate not just how easy it can be to use realtime Web technology, but also the value and increased engagement that a realtime factor can introduce.

Creating Generic Blog Commenting System

START FROM A TEMPLATE

I want to focus on adding realtime commenting to a blog post so **let's**

start from a template⁸.

This template re-uses the HTML5 layout defined in the post on Coding An HTML 5 Layout From Scratch⁹ and the file structure we'll start with is as follows (with some additions that we don't need to worry about at the moment):

- css (dir) (incl. *global-forms.css*, *main.css*, *reset.css*)

- images (dir)
- index.php

HTML

The template HTML, found in *index.php*, has been changed from the HTML5 Layout article to focus on the content being a blog post with comments. You can view the HTML source here¹⁰.

The main elements to be aware of are:

- <section id="content">** – the blog post content
- <section id="comments">** – where the comments are to appear. This is where the majority of our work will be done

COMMENTS

Now that we've got the HTML in place for our blog post and for displaying the comments we also need a way for our readers to submit comments, so let's add a `<form>` element to collect and submit the comment details to `post_comment.php`. We'll add this at the end of the `<section id="comments">` section wrapped in a `<div id="respond">`.

```
<div id="respond">

    <h3>Leave a Comment</h3>

    <form action="post_comment.php" method="post"
id="commentform">

        <label for="comment_author"
class="required">Your name</label>
        <input type="text" name="comment_author"
id="comment_author" value="" tabindex="1"
required="required">

        <label for="email" class="required">Your
email;</label>
        <input type="email" name="email" id="email"
value="" tabindex="2" required="required">

        <label for="comment" class="required">Your
message</label>
```



```
        <textarea name="comment" id="comment"
rows="10" tabindex="4" required="required">
</textarea>

        <-- comment_post_ID value hard-coded as 1 -->
        <input type="hidden" name="comment_post_ID"
value="1" id="comment_post_ID" />
        <input name="submit" type="submit"
value="Submit comment" />

    </form>

</div>
```

COMMENT FORM CSS

Let's apply some CSS to make things look a bit nicer by adding the following to *main.css*: `#respond` {

```
    margin-top: 40px;
}
```

```
#respond input[type='text'],
#respond input[type='email'],
#respond textarea {
    margin-bottom: 10px;
    display: block;
    width: 100%;
```

```
border: 1px solid rgba(0, 0, 0, 0.1);  
-webkit-border-radius: 5px;  
-moz-border-radius: 5px;  
-o-border-radius: 5px;  
-ms-border-radius: 5px;  
-khtml-border-radius: 5px;  
border-radius: 5px;  
  
line-height: 1.4em;  
}
```

Once the HTML structure, the comment form and the CSS are in place our blogging system has started to look a bit more presentable.

Smashing HTML5!

HTML5 in the year 2022 2012

[home](#) [portfolio](#) [blog](#) [contact](#)

Building a Pusher-powered Real-Time Commenting System

10th February 2012 By [Phil Leggetter](#)

The web has become increasingly interactive over the years. This trend is set to continue with the next generation of applications driven by the **real-time web**. Adding real-time functionality to an application can result in a more interactive and engaging user experience. However, setting up and maintaining the server-side realtime components can be an unwanted distraction. But don't worry, there is a solution.

Comments

Be the first to add a comment.

Leave a Comment

Your name *

Your email *

Your message *

[Submit comment](#)

blogroll

[HTML5 Doctor](#)

[W3C](#)

[HTML5 Doctor](#)

[W3C](#)

[HTML5 Doctor](#)

[W3C](#)

[HTML5 Spec \(working draft\)](#)

[Wordpress](#)

[HTML5 Spec \(working draft\)](#)

[Wordpress](#)

[HTML5 Spec \(working draft\)](#)

[Wordpress](#)

[Smashing Magazine](#)

[Wikipedia](#)

[Smashing Magazine](#)

[Wikipedia](#)

[Smashing Magazine](#)

[Wikipedia](#)

social

[delicious](#)

[digg](#)

[facebook](#)

[last.fm](#)

[rss](#)

[twitter](#)

Smashing Magazine
Amazing Magazine



Smashing Magazine is a website and blog that offers resources and advice to web developers and web designers. It was founded by Sven Lennartz and Vitaly Friedman.

2009-2012 [Smashing Magazine](#).

HANDLING COMMENT SUBMISSION

The next step is to write the PHP form submission handler which accepts the request and stores the comment, *post_comment.php*. You should create this file in the root of your application.

As I said earlier I'm keen to focus on the realtime functionality so a class exists within the template that you've downloaded which encapsulate some of the standard data checking and persistence functionality. This class is defined in *Persistence.php* (you can view the source here¹¹), is in no way production quality, and handles: Basic validation Basic data sanitization Very simple persistence using a user **\$_SESSION**. This means that a comment saved by one user will not be available to another user.

This also means that we don't need to spend time setting up a database and all that goes with it and makes *post_comment.php* very simple and clean. If you wanted to use this functionality in a production environment you would need to re-write the contents of *Persistence.php*. Here's the code for *post_comment.php*.

```
<?php
require('Persistence.php');

$db = new Persistence();
if( $db->add_comment($_POST) ) {
    header( 'Location: index.php' );
}
else {
    header( 'Location: index.php?error=Your comment
was not posted due to errors in your form submission'
);
}
?>
```

The above code does the following:

- Includes the *Persistence.php* class which handles saving and fetching comments.
- Creates a new instances of the **Persistence** object and assigns it to the variable **\$db**.
- Tries to add the comment to the **\$db**. If the comment is successfully added it redirects back to the blog post. If it fails the redirection also occurs but some error text is appended to an *error* query parameter.

DISPLAYING THE COMMENTS WITH THE BLOG POST

The final thing we need to do to have our Generic Blog Commenting System up and running is to update the blog page, *index.php*, to fetch and display the comments from the **Persistence** object.

- Since this isn't a real blogging system we'll hard code the **\$comment_post_ID** value to be **1**.

- Include the *Persistence.php* class and fetch the comments from it. Comments are associated with a blog post using a unique **`$comment_post_ID`**.

```
<?php
require('Persistence.php');
$comment_post_ID = 1;
$db = new Persistence();
$comments = $db->get_comments($comment_post_ID);
$has_comments = (count($comments) < 0);
?>
```

Since we now have the **`$comment_post_ID`** accessible via PHP we should update the HTML for the comment form to use this value.

```
<input type="hidden" name="comment_post_ID" value="
<?php echo($comment_post_ID); ?>" id="comment_post_ID"
/>
```

We now have all the comments related to the blog post referenced by the **`$comments`** variable we need to display them on the page. To do this we need to update the PHP in *index.php* to iterate through them and create the required HTML.

```

<ol id="posts-list" class="hfeed">
    <li class="no-comments">Be the first to add a
comment.</li>
    <?php
        foreach ($comments as $comment) {
            <?php
                <li>
                    <div class="comment">
                        <div class="comment-content">
                            <div class="comment-author">
                                By <a class="url fn" href="#">
<?php echo($comment['comment_author']); ?></a>
                            </div>
                            <div class="comment-text">
                                <p><?php
echo($comment['comment']); ?></p>
                            </div>
                        </div>
                    </li>
                <?php
            }
        }
    </li>
</ol>

```


You'll notice that if the value of `$has_comments` is true an additional CSS class is applied to the ordered list called *has-comments*. This is so we can hide the list item with the "Be the first to add a comment"

message when comments are being displayed using CSS: `#posts-list.has-comments li.no-comments { display: none; }`

Now that all this is in place we have a functional blog commenting system. If you would like to start writing your code from this basic functioning blog commenting system you can also download the code completed up to here¹².

Comments

15 February 2012
By [Joe Blogs](#)

Interesting post Phil. It's great to see that a blog really can come alive when the comments update in real-time. The commenting system becomes a conversation platform.

15 February 2012
By [Phil Leggetter](#)

Thanks Joe (great name by the way). I'm pleased you see the benefits of adding realtime functionality to a commenting system. It really can draw users in and turn a standard blog post into a place where conversation takes place. Old style commenting is still great, but real-time comments are really engaging and can make a page much more sticky and engaging.

15 February 2012
By [Max Williams](#)

Phil - great post. Keep up the good work.

Leave a Comment

Your name *

Your email *

Please fill out this field.

Progressive Enhancement With jQuery

The first step in making our blog commenting system feel less like a Web page and more like an application is to stop page reloads when a user submits a comment. We can do this by submitting the comments to the server using an AJAX request. Since jQuery is probably the defacto standard for cross browser JavaScript functionality we'll use it here. Although I'm using jQuery here, I'd also like to highlight that it's a good idea to not always use jQuery¹³. Instead, analyze your scenario and make a considered decision because there are some cases¹⁴ where you are best not to.

In an attempt to try and keep as much scripting (PHP and JavaScript) from the *index.php* file we'll create a new folder for our JavaScript and in there a file for our application JavaScript. The path to this file should be *js/app.js*. This file should be included after the jQuery include.

```
<script src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
<script src="js/app.js"></script>
```

CAPTURE THE COMMENT FORM SUBMISSION

When the page is ready bind to the **submit** event of the form.

```
$(function() {  
    $('#commentform').submit(handleSubmit);  
});
```

When the form is submitted and the **handleSubmit** function is called the comment data we want to send to the server is extracted from the form. There are more elegant ways of fetching the data from the form but this approach clearly shows where we're getting the values from and the **data** object we are creating.

```
function handleSubmit() {  
    var form = $(this);  
    var data = {  
        "comment_author":  
form.find('#comment_author').val(),  
        "email": form.find('#email').val(),  
        "comment": form.find('#comment').val(),  
        "comment_post_ID":  
form.find('#comment_post_ID').val()  
    };  
  
    postComment(data);  
  
    return false;  
}
```

```
function postComment(data) {  
    // send the data to the server  
}
```

POST THE COMMENT WITH AJAX

Within the **postComment** function make a *POST* request to the server passing the data that we've retrieved from the form. When the request is made an additional HTTP header will be sent to identify the request as being an AJAX request. We want to do this so that we can return a JSON response if it is an AJAX request and maintain our basic functionality if it isn't (for more information on this see [Detected AJAX events on the Server¹⁵](#)). We also define two handlers; **postSuccess** for handling the comment being successfully stored and **postError** to handle any failures.

```
function postComment(data) {  
    $.ajax({  
        type: 'POST',  
        url: 'post_comment.php',  
        data: data,  
        headers: {  
            'X-Requested-With': 'XMLHttpRequest'  
        },  
        success: postSuccess,  
        error: postError  
    });  
}
```

```
}

function postSuccess(data, textStatus, jqXHR) {
    // add comment to UI
}

function postError(jqXHR, textStatus, errorThrown) {
    // display error
}
```

DYNAMICALLY UPDATING THE USER INTERFACE WITH THE COMMENT

At this point the comment data is being sent to the server and saved, but the AJAX response isn't providing any meaningful response. Also, the comments section isn't being updated to show the newly submitted comment so the user would have to refresh the page to see it. Let's start by writing the code to update the UI and test that functionality.

If you are thinking "Hang on a minute! We don't have all the data we need from the Web server to display the comment" then you are correct. However, this doesn't stop us writing the code to update the UI and also testing that it works. Here's how:

```
function postSuccess(data,
textStatus, jqXHR) {
    $('#commentform').get(0).reset();
    displayComment(data);
}
```

```

function displayComment(data) {
    var commentHtml = createComment(data);
    var commentEl = $(commentHtml);
    commentEl.hide();
    var postsList = $('#posts-list');
    postsList.addClass('has-comments');
    postsList.prepend(commentEl);
    commentEl.slideDown();
}

function createComment(data) {
    var html = '' +
        '<li><article id="' + data.id + '"
class="hentry">' +
        '<footer class="post-info">' +
            '<abbr class="published" title="' +
data.date + '">' +
                parseDisplayDate(data.date) +
            '</abbr>' +
            '<address class="vcard author">' +
                'By <a class="url fn" href="#">' +
data.comment_author + '</a>' +
            '</address>' +
        '</footer>' +
        '<div class="entry-content">' +
            '<p>' + data.comment + '</p>' +
        '</div>' +
        '</article></li>';

```

```

    return html;
}

function parseDisplayDate(date) {
    date = (date instanceof Date? date : new Date(
Date.parse(date) ) );
    var display = date.getDate() + ' ' +
                    ['January',
'February', 'March',
                    'April', 'May',
'June', 'July',
                    'August',
'September', 'October',
                    'November',
'December'][date.getMonth()] + ' ' +
                    date.getFullYear();

    return display;
}

```

The code above does the following:

- Within the **postSuccess** function we clear the form values and call **displayComment**.
- **displayComment** first calls the **createComment** function to create the list item (****) HTML as a **String**.

- We then convert the HTML to a jQuery object using `$(commentHtml)` and hide the element.
- The comment list item is then prepended to the comments ordered list (``). The list also has a class called *has-comments* added to it so we can hide the first list item which contains the "Be the first to comment" statement.
- Finally, we call `commentEl.slideDown()` so that the comment is shown in what is becoming the standard "here's a new item" way.

The functionality is now implemented but we want to test it out. This can be achieved in two ways: The `displayComment` is a global function so we can call it directly using the JavaScript console of the browser.

- We can bind to an event on the page that triggers a fake update which calls the `displayComment` function. Let's go with the latter and bind to the "u" key being released by binding to the `keyup` event. When it is, we'll create a fake `data` object containing all the information required to create a new comment and pass it to the `displayComment` function. That comment will then be displayed in the UI.

Hit the "u" key a few times and see the comments appear.

```
$(function() {  
  
    $(document).keyup(function(e) {  
        e = e || window.event;  
        if(e.keyCode === 85){  
            displayComment({  
                "id": "comment_1",  
                "comment_post_ID": 1,  
                "date": "Tue, 21 Feb 2012 18:33:03  
+0000",  
                "comment": "The realtime Web rocks!",  
                "comment_author": "Phil Leggetter"  
            });  
        }  
    });  
});
```

Great! We now know that our **displayComment** function works exactly as we expect it to. **Remember to remove the test function** before you go live or you'll really confuse your user every time they press "u".

Comments

21 February 2012
By [Phil Leggetter](#) The realtime web rocks!

21 February 2012
By [Phil Leggetter](#) The realtime web rocks!

21 February 2012
By [Phil Leggetter](#) The realtime web rocks!

21 February 2012
By [Phil Leggetter](#) The realtime web rocks!

21 February 2012
By [Phil Leggetter](#) The realtime web rocks!

DETECT AND RESPONDING TO THE AJAX REQUEST

All that's left to do is update the *post_comment.php* file to detect the AJAX call and return information about the newly created comment.

Detecting the AJAX request is done by checking for the **X-Requested-With** header: `$ajax = ($_SERVER['HTTP_X_REQUESTED_WITH'] === 'XMLHttpRequest');`

Once we know the request is an AJAX request we can update the code to respond with an appropriate status code and the data representing the comment. We also need to ensure that the original functionality is maintained. The *post_comment.php* code now looks as follows:

```
<?php
```

```
require('Persistence.php');
```

```
$ajax = ( $_SERVER[ 'HTTP_X_REQUESTED_WITH' ] ===
```

```
'XMLHttpRequest');

$db = new Persistence();
$added = $db->add_comment($_POST);

if($ajax) {
    sendAjaxResponse($added);
}
else {
    sendStandardResponse($added);
}

function sendAjaxResponse($added) {
    header("Content-Type: application/x-javascript");
    if($added) {
        header( 'Status: 201' );
        echo( json_encode($added) );
    }
    else {
        header( 'Status: 400' );
    }
}

function sendStandardResponse($added) {
    if($added) {
        header( 'Location: index.php' );
    }
    else {
        header( 'Location: index.php?error=Your
comment was not posted due to errors in your form
```

```
submission' );  
    }  
}  
?>
```

Key points from the above code are:

- The `$db->add_comment($_POST)` call returns the data from the added comment which is assigned to the `$added` variable.
- By setting the response *Content-Type* to “application/json” we tell jQuery to convert the returned string into a JavaScript object. For more information on calling JSON Web services see [A Beginner's Guide To jQuery-Based JSON API Clients¹⁶](#).
- The `201` status code indicates a successful call and also that a resource (the comment) was created by the call.

The blog commenting system now works in a much more dynamic way, instantly showing the user that their comment has been posted without refreshing the page. In addition, the way the we've added the JavaScript based functionality to the page means that if JavaScript is disabled or a JavaScript file fails to load that the system will fallback to the standard functionality we first implemented.

Getting RealTime—Finally!

As with any "from scratch" tutorial it can take a bit of time to get to the **really** interesting part, but we're finally here. However, **all the work we've up in has been worth it**. Because we've built our commenting system up in a progressively enhanced way, plugging Pusher into it is going to be really easy

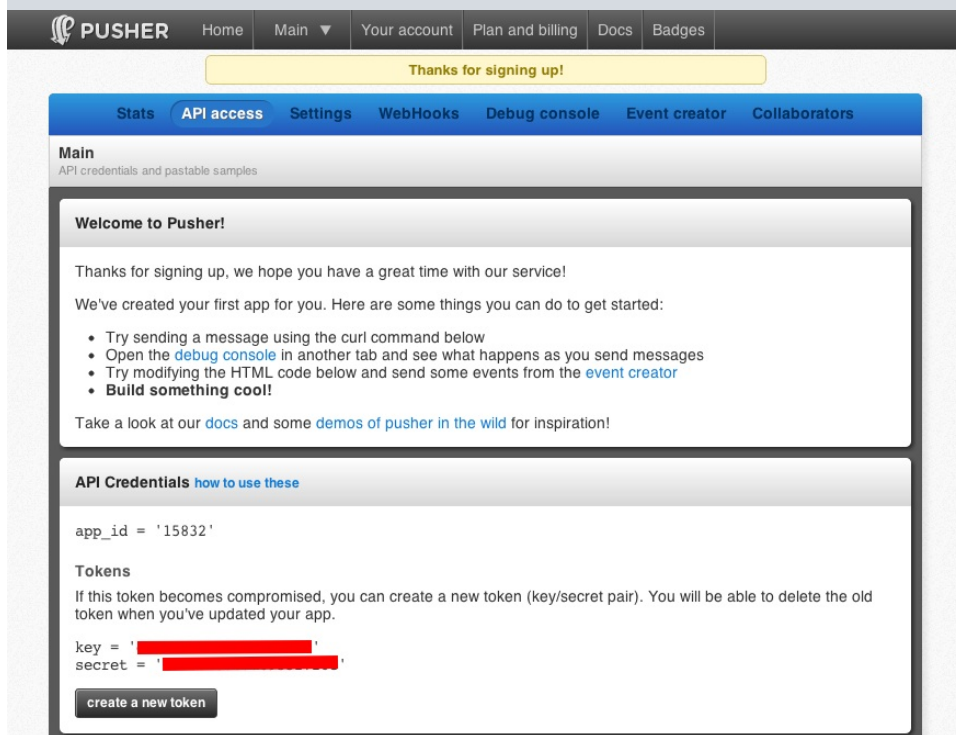
WHAT IS PUSHER?

At the start of the tutorial we said that we would use Pusher to add the realtime functionality to the application. But what is Pusher?

Pusher is a hosted service for quickly and easily adding realtime features into Web and mobile applications. It offers a RESTful API that makes it really easy to publish events from any application that can make a HTTP request and a WebSocket API for realtime bi-directional communication. You don't even need to use the APIs directly as there are server¹⁷ (PHP, Ruby, node.js, ASP.NET, Python and more) and client¹⁸ (JavaScript, iOS, Android, .NET, ActionScript, Arduino and more) libraries available in a host of technologies which means you can add realtime functionality to an app within minutes - I'm confident you'll be surprised just how easy!

SIGN UP FOR PUSHER AND GET YOUR API CREDENTIALS

In order to add Pusher-powered realtime functionality to a Web application you first need to sign up for a free Sandbox account¹⁹. After you have signed up you'll be taken to the Pusher dashboard where you'll see that a "Main" application has been created for you. You'll also see you are in the "API Access" section for that application where you can grab your API credentials.



For ease of access create a *pusher_config.php* file and **define** the credentials in there so we can refer to them later:

```
<?php
```

```
define('APP_ID', 'YOUR_APP_ID');  
define('APP_KEY', 'YOUR_APP_KEY');  
define('APP_SECRET', 'YOUR_APP_SECRET');  
?>
```

In your version of *pusher_config.php* be sure to replace the values which being 'YOUR_' with your actual credentials.

You should also **require** this in your *index.php* file. We should also make the **APP_KEY** available to the JavaScript runtime as we are going to need it to connect to Pusher.

```
<?php  
require('pusher_config.php');  
?>  
  
<script>  
var APP_KEY = '<?php echo(APP_KEY); ?>';  
&lt;/script&gt;
```

REALTIME JAVASCRIPT

The first thing you need to do when adding Pusher to a Web application is include the Pusher JavaScript library and connect to Pusher. To

connect you'll need to use the *key* which you grabbed from the Pusher dashboard. Below you can see all that is required to hook up the front-end application to Pusher.

```
Include the Pusher JavaScript library after the app.js include: <script
src="http://code.jquery.com/jquery-1.7.1.min.js">
</script>
<script src="http://js.pusher.com/1.11/pusher.min.js">
</script>
<script src="js/app.js"></script>
```

```
Add the Pusher functionality to app.js: var pusher = new
Pusher(APP_KEY);
var channel = pusher.subscribe('comments-' +
$('#comment_post_ID').val());
channel.bind('new_comment', displayComment);
```

This probably looks too easy to be true, so here are the details about what the above code does: **var pusher = new Pusher(APP_KEY);**

Creates a new instance of a **Pusher** object and in doing so connects you to Pusher. The application to use is defined by the **APP_KEY** value that you pass in and that we set up earlier.

- **var channel = pusher.subscribe('comments-' + \$('#comment_post_ID').val());**

Channels²⁰ provide a great way of organizing streams of

realtime data. Here we are subscribing to comments for the current blog post, uniquely identified by the value of the `comment_post_ID` hidden form input element.

- `channel.bind('new_comment', displayComment);`

Events²¹ are used to further filter data and are ideal for linking updates to changes in the UI. In this case we want to bind to an event which is triggered whenever a new comment is added and display it. Because we've already created the `displayComment` function we can just pass in a reference to the call to `bind`.

SENDING REALTIME EVENTS USING THE EVENT CREATOR

We can also test out this functionality without writing any server-side code by using the **Event Creator** for your app which can also be found in the Pusher dashboard. The Event Creator lets you publish events on a channel through a simple user interface. From the code above we can see that we want to publish an event named *"new_comment"* on the *"comments1"* channel. From the earlier test function we also have an example of the test data we can publish.

The screenshot shows the Pusher web interface. At the top is a dark navigation bar with the Pusher logo and links: Home, Smashing Mag (with a dropdown arrow), Your account, Plan and billing, Docs, and Badges. Below this is a blue sub-navigation bar with links: Stats, API access, Settings, WebHooks, Debug console, Event creator (highlighted), and Collaborators. The main content area is titled 'Smashing Mag Event creator'. It contains a message: 'Use this form to send events to your connected clients.' Below this is a form with three fields: 'Channel name' with the value 'comments-1' and a hint 'e.g. "project_42", "room_8".'; 'Event name' with the value 'new_comment' and a hint 'e.g. "user_created", "new_message".'; and 'Event data' with a JSON object:

```
{ "date": "Tue, 21 Feb 2012 18:33:03 +0000", "comment": "Awesome real-time blog post!", "comment_author": "Phil Leggetter" }
```

 and a hint 'e.g. "{ "name": "Joe", "message_count": 23 }'.'. At the bottom right of the form is a 'Send event' button. Below the form, it says 'No events sent yet'. The footer of the interface shows '© 2012 Pusher | Support | Logout'.

REALTIME PHP

Again, we've proven that our client-side functionality works without having to write any server-side code. Now let's add the PHP code we need to trigger the new comment event as soon as a comment is posted in our comment system.

Pusher offers a number of server-side libraries²² which make it easy to publish events in addition to helping with functionality such as private channel²³ authentication and providing user information for presence channels²⁴. We just want to use the basic event triggering functionality in the *post_comment.php* file so we need to download the Pusher PHP library²⁵ (direct zip file download²⁶).

Once you've downloaded this zip file, unzip it into the directory along with your other files. Your file structure will now look something like this:
index.php css (dir) images (dir) post_comment.php pusher_config.php
Persistence.php squeeks-Pusher-PHP (dir) including *lib* (dir) with
Pusher.php

An event can be triggering in just a few lines of code:

```
<?php
require('squeeks-Pusher-PHP/lib/Pusher.php');
require('pusher_config.php');

$pusher = new Pusher(APP_KEY, APP_SECRET, APP_ID);
$pusher->trigger('comments1', 'new_comment', array(
    "comment_post_ID" => 1,
    "date" => "Tue, 21 Feb 2012 18:33:03 +0000",
    "comment" => "The realtime Web rocks!",
    "comment_author" => "Phil Leggetter"
));
?>
```

However, we need to apply a some additional logic before we trigger the event: Check that the comment was added.

- Extract the unique comment identifier from the **\$added** array.

- Build the text to identify a channel name for the submitted comment.
- Trigger a *new_comment* event on the channel with the **\$added** data.
*Note: The library automatically converts the **\$added** array variable to JSON to be sent through Pusher.*

Therefore the full *post_comment.php* file ends up looking as follows:

```
<?php
require('Persistence.php');
require('squeeks-Pusher-PHP/lib/Pusher.php');
require('pusher_config.php');

$sajax = ($_SERVER[ 'HTTP_X_REQUESTED_WITH' ] ===
'XMLHttpRequest');

$db = new Persistence();
$added = $db->add_comment($_POST);

if($added) {
    $channel_name = 'comments-' .
$added['comment_post_ID'];
    $event_name = 'new_comment';

    $pusher = new Pusher(APP_KEY, APP_SECRET, APP_ID);
    $pusher->trigger($channel_name, $event_name,
$added);
}
```

```
if($ajax) {
    sendAjaxResponse($added);
}
else {
    sendStandardResponse($added);
}

function sendAjaxResponse($added) {
    header("Content-Type: application/json");
    if($added) {
        header( 'Status: 201' );
        echo( json_encode($added) );
    }
    else {
        header( 'Status: 400' );
    }
}

function sendStandardResponse($added) {
    if($added) {
        header( 'Location: index.php' );
    }
    else {
        header( 'Location: index.php?error=Your
comment was not posted due to errors in your form
submission' );
    }
}

?>
```

If you run the app now in two different browser windows you'll see that as soon as you submit a comment in one window that comment will instantly ("magically") appear in the second window. **We now have a realtime commenting system!**

But..., we're not done quite yet. You'll also see that the comment is shown twice in the window of the user who submitted it. This is because the comment has been added by the AJAX callback, and by the Pusher event. Because this is a very common scenario, especially if you've built an application in a progressively enhanced way as we have, the Pusher server libraries expose a way of excluding a connection/user²⁷ from receiving the event via Pusher.

In order to do this you need to send a unique connection identifier called a **socket_id** from the client to the server. This identifier can then be used to define who will be excluded.

```
function handleSubmit() {  
    var form = $(this);  
    var data = {  
        "comment_author":  
form.find('#comment_author').val(),  
        "email": form.find('#email').val(),  
        "comment": form.find('#comment').val(),  
        "comment_post_ID":
```

```

form.find('#comment_post_ID').val()
    };

    var socketId = getSocketId();
    if(socketId !== null) {
        data.socket_id = socketId;
    }

    postComment(data);

    return false;
}

function getSocketId() {
    if(pusher && pusher.connection.state ===
'connected') {
        return pusher.connection.socket_id;
    }
    return null;
}

```

The changes we've made are:

- A new function called **getSocketId** has been added to get the **socket_id**. It wraps a check to ensure that the **pusher** variable has been set and also that the client is connected to Pusher.

- The `handleSubmit` has been updated to check to see if a `socket_id` is available. If it is, this information is posted to the server along with the comment data.

On the server we need to use the `socket_id` parameter if it is present and therefore exclude the connection and user who submitted the comment, or pass in `null` if it's not:

```
$channel_name = 'comments-' .  
$added['comment_post_ID'];  
$event_name = 'new_comment';  
$socket_id = (isset($_POST['socket_id']))?  
$_POST['socket_id']:null);  
  
$pusher = new Pusher(APP_KEY, APP_SECRET, APP_ID);  
$pusher->trigger($channel_name, $event_name,  
$added, $socket_id);
```

And as simple as that we have **a fully realtime enabled blog commenting system** and we only send messages to users who really need to see them. As with the AJAX functionality the realtime functionality has been added in a progressively enhancing way, to ensure it doesn't impact on any other functionality. You can find the a demo running here²⁸ and the completed solution in the realtime commenting repository²⁹ in github.

Good RealTime App Development Practices

Realtime application development shares common good development practices with general Web development. However, I thought I would share a couple of tips that can come in particularly handy.

USE BROWSER DEVELOPER TOOLS

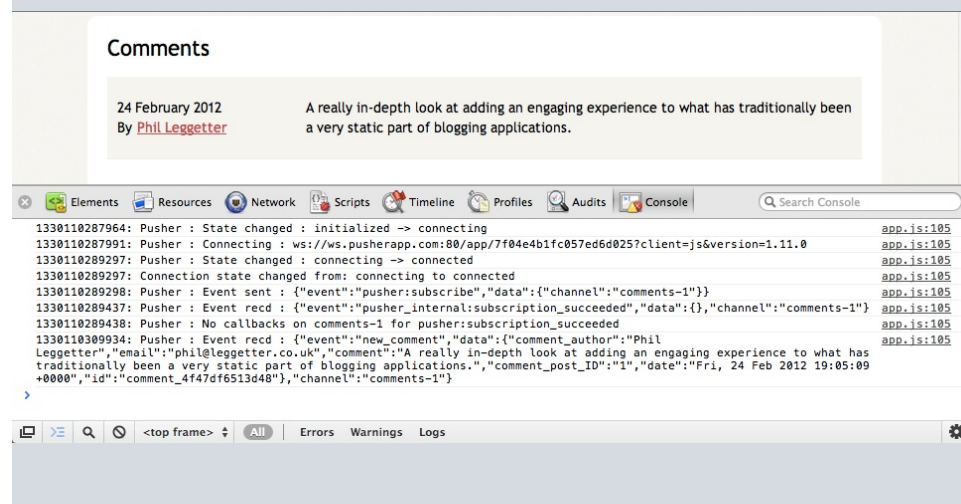
When you start doing a lot of JavaScript development the browser developer tools becomes your best friend. It's the same when adding realtime functionality to your Web app, not only because you are using JavaScript, but also because the JavaScript library you are using is likely to be doing some reasonably complex things internally. So, the best way of understanding what is going on and if your code is using it as expect is to enable logging which usually goes to the developer console. All major browser vendors now offer good developer tools which include a console: Firebug addon³⁰ for Firefox Google Chrome Developer Tools³¹

- Internet Explorer F12 developer tools³²
- Opera Dragonfly³³
- Safari Developer Tools³⁴

The Pusher JavaScript library provides a way to hook into the logging functionality. All you need to do is assign a function to the **Pusher.log**³⁵ static property. This function will then receive all log messages. You can do what you like within the function but best practice is to log the messages to the developer console. You can do this as follow, ensuring the code it executed after the Pusher JavaScript library include:

```
Pusher.log = function(msg) {  
    if(window.console && window.console.log) {  
        window.console.log(new Date().getTime() + ': '  
+ msg);  
    }  
};
```

The code above checks to make sure the **console** and **log** function is available – it's not in older browsers – and then logs the messages along with a timestamp to the JavaScript console. This can be incredibly handy in tracking down problems.



CHECK CONNECTIVITY

Any good realtime technology will maintain a persistent connection between the client (web browser) and the Web server or service. Sometimes the client will lose connectivity and when the client isn't connected to the Internet the realtime functionality won't work. This can happen a lot with applications running on mobile devices which rely on mobile networks. So, if your application relies on that connectivity and functionality then it's important to deal with scenarios where the client isn't connected. This might be by displaying a message to tell the user they are offline or you might want to implement some alternative functionality.

The Pusher JavaScript library exposes connectivity state³⁶ via the `pusher.connection` object, which we briefly saw earlier when fetching the `socket_id`. Binding to state changes and implementing your required functionality is quite simple as it follows the same mechanism as binding to events on channels: `var pusher = new`

```
Pusher(APP_KEY);
pusher.connection.bind('state_change',
function(states) {
    Pusher.log('Connection state changed from: ' +
states.previous + ' to ' + states.current);
});
```

Conclusion

We're seeing realtime functionality appearing in a large number of high profile applications: some have it at the core of what they offer whilst others use it sparingly. No matter how it is used the end goal is generally the same; to enhance the user experience and keep users engaged with an application. By converting a basic blog commenting system into a more engaging communication platform I hope I've demonstrated that the functionality and experience can easily be layered on existing application.

The ease of access to this technology is a relatively new thing and we've only just touched the potential uses for the technology. Realtime stats, instant news updates, activity streams, social interaction, collaboration and gaming are just a few common uses but as more developers become aware of, and comfortable with, the technology I'm confident that we're going to see some truly amazing innovation. An "Internet of RealTime Things"? 🐼

-
1. <http://pusher.com>
 2. http://en.wikipedia.org/wiki/Real-time_web
 3. [http://en.wikipedia.org/wiki/Comet_\(programming\)](http://en.wikipedia.org/wiki/Comet_(programming))

4. http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
5. <http://pusher.com/websocket>
6. <http://www.w3.org/TR/websockets/>
7. <http://tools.ietf.org/html/rfc6455>
8. <https://github.com/pusher/realtime-commenting/zipball/start>
9. <http://coding.smashingmagazine.com/2009/08/04/designing-a-html-5-layout-from-scratch/>
10. <https://github.com/pusher/realtime-commenting/blob/start/index.php>
11. <https://github.com/pusher/realtime-commenting/blob/start/Persistence.php>
12. <https://github.com/pusher/realtime-commenting/zipball/basic>
13. <http://www.leggetter.co.uk/2012/02/19/jquery-uk-2012-event-dont-always-use-jquery.html>
14. <http://www.leggetter.co.uk/2012/02/12/considerations-when-updating-the-dom-to-display-realtime-data.html>

15. <http://www.learningjquery.com/2010/03/detecting-ajax-events-on-the-server>
16. <http://coding.smashingmagazine.com/2012/02/09/beginners-guide-jquery-based-json-api-clients/>
17. http://pusher.com/docs/rest_libraries
18. http://pusher.com/docs/client_libraries
19. <http://pusher.com/signup>
20. <http://pusher.com/docs/channels>
21. http://pusher.com/docs/client_api_guide/client_events
22. http://pusher.com/docs/rest_libraries
23. http://pusher.com/docs/private_channels
24. <http://pusher.com/docs/presence>
25. <https://github.com/squeeks/Pusher-PHP>

26. <https://github.com/squeeks/Pusher-PHP/zipball/master>
27. http://pusher.com/docs/publisher_api_guide/publisher_excluding_recipients
28. <http://www.leggetter.co.uk/pusher/realtime-commenting/>
29. <https://github.com/pusher/realtime-commenting>
30. <http://getfirebug.com/>
31. <http://code.google.com/chrome/devtools/docs/overview.html>
32. [http://msdn.microsoft.com/en-us/library/gg589507\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/gg589507(v=VS.85).aspx)
33. <http://www.opera.com/dragonfly/documentation/>
34. <https://developer.apple.com/library/safari/#documentation/appleapplications/Conceptual/Sa>
35. http://pusher.com/docs/client_api_guide/client_global_configuration#pusher_log
36. http://pusher.com/docs/connection_states

The Developer's Guide To Conflict-Free JavaScript And CSS In WordPress

BY PETER WILSON 

Imagine you're playing the latest hash-tag game on Twitter when you see this friendly tweet:

You might want to check your #WP site. It includes two copies of jQuery. Nothing's broken, but loading time will be slower.

You check your source code, and sure enough you see this: `<script src="wp-includesjs/jquery/jquery.js?ver=1.6.1" type="text/javascript"></script><script src="wp-contentplugins/some-plugin/jquery.js"></script>`



WHAT WENT WRONG?

The first copy of jQuery is included the WordPress way, while **some-plugin** includes jQuery as you would on a static HTML page.

A number of JavaScript frameworks are included in WordPress by default, including: Scriptaculous, jQuery (running in noConflict mode¹), the jQuery UI core and selected widgets, Prototype.

A complete list can be found in the Codex². On the same page are instructions for using jQuery in noConflict mode³.

AVOIDING THE PROBLEM

WordPress includes these libraries so that plugin and theme authors can avoid this problem by using the `wp_register_script` and `wp_enqueue_script` PHP functions to include JavaScript in the HTML.

Registering a file alone doesn't do anything to the output of your HTML; it only adds the file to WordPress's list of known scripts. As you'll see in the next section, we register files early on in a theme or plugin where we can keep track of versioning information.

To output the file to the HTML, you need to enqueue the file. Once you've done this, WordPress will add the required script tag to the header or footer of the outputted page. More details are provided later in this article.

Registering a file requires more complex code than enqueueing the files; so, quickly parsing the file is harder when you're reviewing your code. Enqueueing the file is far simpler, and you can easily parse how the HTML is being affected.

For these techniques to work, the theme's *header.php* file must include the line `<?php wp_head(); ?>` just before the `</head>` tag, and the *footer.php* file must include the line `<?php wp_footer(); ?>` just before the `</body>` tag.

Registering JavaScript

Before registering your JavaScript, you'll need to decide on a few additional items: the file's handle (i.e. the name by which WordPress will know the file); other scripts that the file depends on (jQuery, for example); the version number (optional); where the file will appear in the HTML (the header or footer).

This chapter refers to building a theme, but the tips apply equally to building a plugin.

EXAMPLES

We'll use two JavaScript files to illustrate the power of the functions: The first is *base.js*, which is a toolkit of functions used in our example website.

```
function makeRed(selector){
    var $ = jQuery; //enable $ alias within this scope
    $(function(){
        $(selector).css('color', 'red');
    });
}
```

The *base.js* file relies on jQuery, so jQuery can be considered a dependency.

This is the first version of the file, version 1.0.0, and there is no reason to run this file in the HTML header.

The second file, *custom.js*, is used to add the JavaScript goodness to our website.

```
makeRed( '*' );
```

This *custom.js* file calls a function in *base.js*, so *base.js* is a dependency.

Like *base.js*, *custom.js* is version 1.0.0 and can be run in the HTML footer.

The *custom.js* file also indirectly relies on jQuery. But in this case, *base.js* could be edited to be self-contained or to rely on another framework. There is no need for jQuery to be listed as a dependency of *custom.js*.

It's now simply a matter of registering your JavaScript using the function

wp_register_script. This takes the following arguments: **\$handle**

A string **\$source**

A string **\$dependencies**

An array (optional) **\$version**

A string (optional) **\$in_footer**

True/false (optional, default is false) When registering scripts, it is best to use the **init** hook and to register them all at once.

To register the scripts in our example, add the following to the theme's

functions.php file: **function** mytheme_register_scripts() {

```
//base.js – dependent on jQuery
```

```
wp_register_script(
```

```
    'theme-base', //handle
```

```
    'wp-content/themes/mytheme/base.js', //source
```

```
    array('jquery'), //dependencies
```

```
    '1.0.0', //version
```

```
    true //run in footer
```

```
);
```

```
//custom.js – dependent on base.js
```

```
wp_register_script(
```

```
    'theme-custom',
```

```
    'wp-content/themes/mytheme/custom.js',
```

```
    array('theme-base'),
```

```
    '1.0.0',
```

```
    true
```

```
);
```

```
}
```

```
add_action('init', 'mytheme_register_scripts');
```

There is no need to register jQuery, because WordPress already has. Re-registering it could lead to problems.

YOU HAVE ACHIEVED NOTHING!

All of this registering JavaScript files the WordPress way has, so far, achieved nothing. Nothing will be outputted to your HTML files.

To get WordPress to output the relevant HTML, we need to enqueue our files. Unlike the relatively long-winded commands required to register the functions, this is a very simple process.

Outputting the JavaScript HTML

Adding the **<script>** tags to your HTML is done with the **wp_enqueue_script** function. Once a script is registered, it takes one argument, the file's handle.

Adding JavaScript to the HTML is done in the **wp_print_scripts** hook

with the following code: `function mytheme_enqueue_scripts(){
 if (!is_admin()):
 wp_enqueue_script('theme-custom'); //custom.js
 endif; //!is_admin
}
add_action('wp_print_scripts',
'mytheme_enqueue_scripts');`

Of our two registered JavaScript files (*base.js* and *custom.js*), only the second adds JavaScript functionality to the website. Without the second file, there is no need to add the first.

Having enqueued *custom.js* for output to the HTML, WordPress will figure out that it depends on *base.js* being present and that *base.js*, in turn, requires jQuery. The resulting HTML is: `<script src="wp-includesjs/jquery/jquery.js?ver=1.6.1" type="text/javascript"></script>
<script src="wp-contentthemes/mytheme/base.js?ver=1.0.0" type="text/javascript"></script>
<script src="wp-contentthemes/mytheme/custom.js?ver=1.0.0" type="text/javascript"></script>`

Registering Style Sheets

Both of the functions for adding JavaScript to our HTML have sister PHP functions for adding style sheets to the HTML: `wp_register_style`

and `wp_enqueue_style`.

As with the JavaScript example, we'll use a couple of CSS files throughout this chapter, employing the mobile-first methodology for responsive Web design.

The *mobile.css* file is the CSS for building the mobile version of the website. It has no dependencies.

The *desktop.css* file is the CSS that is loaded for desktop devices only. The desktop version builds on the mobile version, so *mobile.css* is a dependency.

Once you've decided on version numbers, dependencies and media types, it's time to register your style sheets using the

`wp_register_style` function. This function takes the following arguments:

- `$handle`

- A string `$source`

- A string `$dependencies`

- An array (optional, default is none) `$version`

- A string (optional, the default is the current WordPress version number) `$media_type`

A string (optional, the default is all) Again, registering your style sheets using the `init` action is best.

To your theme's *functions.php*, you would add this: `function`

```
mytheme_register_styles(){
    //mobile.css for all devices
    wp_register_style(
        'theme-mobile', //handle
        'wp-content/themes/mytheme/mobile.css', //source
        null, //no dependencies
        '1.0.0' //version
    );

    //desktop.css for big-screen browsers
    wp_register_style(
        'theme-desktop',
        'wp-content/themes/mytheme/desktop.css',
        array('theme-mobile'),
        '1.0.0',
        'only screen and (min-width : 960px)' //media type
    );

    /* keep reading */
}
add_action('init', 'mytheme_register_styles');
```

We have used CSS3 media queries to prevent mobile browsers from parsing our desktop style sheet. But Internet Explorer versions 8 and below do not understand CSS3 media queries and so will not parse the desktop CSS either.

IE8 is only two years old, so we should support its users with conditional comments.

CONDITIONAL COMMENTS

When registering CSS using the register and enqueue functions, conditional comments are a little more complex. WordPress uses the object `$wp_styles` to store registered style sheets.

To wrap your file in conditional comments, add extra information to this object.

For Internet Explorer 8 and below, excluding mobile IE, we need to register another copy of our desktop style sheet (using the media type `all`) and wrap it in conditional comments.

In the code sample above, *`/* keep reading */`* would be replaced with the following: `global $wp_styles;`

```
wp_register_style(
    'theme-desktop-ie',
    'wp-content/themes/mytheme/desktop.css',
    array('theme-mobile'),
    '1.0.0'
```

```
);  
  
$wp_styles->add_data(  
    'theme-desktop-ie', //handle  
    'conditional', //is a conditional comment  
    '!(IEMobile)&(lte IE 8)' //the conditional comment  
);
```

Unfortunately, there is no equivalent for wrapping JavaScript files in conditional comments, presumably due to the concatenation of JavaScript in the admin section.

If you need to wrap a JavaScript file in conditional comments, you will need to add it to *header.php* or *footer.php* in the theme. Alternatively, you could use the `wp_head` or `wp_footer` hooks.

Outputting The Style Sheet HTML

Outputting the style sheet HTML is very similar to outputting the JavaScript HTML. We use the enqueue function and run it on the `wp_print_styles` hook.

In our example, we could get away with telling WordPress to queue only the style sheets that have the handles `theme-desktop` and `theme-`

desktop-ie. WordPress would then output the **mobile/all** media version.

However, both style sheets apply styles to the website beyond a basic reset: *mobile.css* builds the website for mobile phones, and *desktop.css* builds on top of that. If it does something and I've registered it, then I should enqueue it. It helps to keep track of what's going on.

Here is the code to output the CSS to the HTML: **function**

```
mytheme_enqueue_styles(){
    if (!is_admin()):
        wp_enqueue_style('theme-mobile'); //mobile.css
        wp_enqueue_style('theme-desktop'); //desktop.css
        wp_enqueue_style('theme-desktop-ie');
//desktop.css lte ie8
    endif; //!is_admin
}
add_action('wp_print_styles',
'mytheme_enqueue_styles');
```

What's The Point?

You may be wondering what the point is of going through all of this extra effort when we could just output our JavaScript and style sheets in the theme's *header.php* or using the **wp_head** hook.

In the case of CSS in a standalone theme, it's a valid point. It's extra work without much of a payoff.

But with JavaScript, it helps to prevent clashes between plugins and themes when each uses a different version of a JavaScript framework. It also makes page-loading times as fast as possible by avoiding file duplication.

WORDPRESS FRAMEWORKS

This group of functions can be most helpful when using a framework for theming. In my agency, Soupgiant⁴, we've built a framework to speed up our production of websites.

As with most agencies, we have internal conventions for naming JavaScript and CSS files.

When we create a bespoke WordPress theme for a client, we develop it as a child theme⁵ of our framework. In the framework itself, we register a number of JavaScript and CSS files in accordance with our naming convention.

In the child theme, we then simply enqueue files to output the HTML.

```
function clienttheme_enqueue_css() {
    if (!is_admin()):
        wp_enqueue_style('theme-mobile');
        wp_enqueue_style('theme-desktop');
        wp_enqueue_style('theme-desktop-ie');
    endif; //!is_admin
}

add_action('wp_print_styles',
'clienttheme_enqueue_css');

function clienttheme_enqueue_js() {
    if (!is_admin()):
        wp_enqueue_script('theme-custom');
    endif; //!is_admin
}

add_action('wp_print_scripts',
'clienttheme_enqueue_js');
```

Adding CSS and JavaScript to our themes the WordPress way enables us to keep track of exactly what's going on at a glance.

A Slight Limitation

If you use a JavaScript framework in your theme or plugin, then you're stuck with the version that ships with the current version of WordPress, which sometimes falls a version or two behind the latest official release of the framework. (Upgrading to a newer version of the framework is technically possible, but this could cause problems with other themes or plugins that expect the version that ships with WordPress, so I've omitted this information from this chapter.) While this prevents you from using any new features of the framework that were added after the version included in WordPress, the advantage is that *all* theme and plugin authors know which version of the framework to expect.

A Single Point Of Registration

Register your styles and scripts in a single block of code, so that when you update a file, you will be able to go back and update the version number easily.

If you use different code in different parts of the website, you can wrap the logic around the enqueue scripts.

If, say, your archive pages use different JavaScript than the rest of the website, then you might register three files: base JavaScript (registered as **theme-base**), archive JavaScript (registered as **theme-archive**), general JavaScript (registered as **theme-general**).

Again, the base JavaScript adds nothing to the website. Rather, it is a group of default functions that the other two files rely on. You could then enqueue the files using the following code: `function`

```
mytheme_enqueue_js(){
    if (is_archive()) {
        wp_enqueue_script('theme-archive');
    }
    elseif (!is_admin()) {
        wp_enqueue_script('theme-general');
    }
}
add_action('wp_print_scripts', 'mytheme_enqueue_js');
```

Using The Google AJAX CDN

While using JavaScript the WordPress way will save you the problem of common libraries conflicting with each other, you might prefer to serve these libraries from Google's server rather than your own.

Using Jason Penny's Use Google Libraries⁶ plugin is the easiest way to do this. The plugin automatically keeps jQuery in noConflict mode.

Putting It All Together

Once you've started registering and outputting your scripts and styles the WordPress way, you will find that managing these files becomes a series of logical steps: **1. Registration to manage:**

- version numbers, file dependencies, media types for CSS, code placement for JavaScript (header or footer); **2. Enqueue/output files to HTML:**
- logic targeting output to specific WordPress pages, WordPress automating dependencies.

Eliminating potential JavaScript conflicts from your WordPress theme or plugin frees you to get on with more important things, such as following up on sales leads or getting back to that hash-tag game that was so rudely interrupted. 🐼

1. <http://api.jquery.com/jquery.noConflict/>

2. http://codex.wordpress.org/Function_Reference/wp_enqueue_script#Default_scripts_included

3. http://codex.wordpress.org/Function_Reference/wp_enqueue_script#jQuery_noConflict_wr
4. <http://soup giant.com/>
5. http://codex.wordpress.org/Child_Themes
6. <http://wordpress.org/extend/plugins/use-google-libraries/>

Optimizing Long Lists Of Yes/No Values With JavaScript

BY LEA VEROU 🐼

Very frequently in Web development (and programming in general), you need to store a long list of boolean values (yes/no, true/false, checked/unchecked... you get the idea) into something that accepts only strings. Maybe it's because you want to store them in **localStorage** or in a cookie, or send them through the body of an HTTP request. I've needed to do this countless times.

The last time I stumbled on such a case wasn't with my own code. It was when Christian Heilmann¹ showed me his then new slide deck², with a cool feature where you could toggle the visibility of individual slides in and out of the presentation. On seeing it, I was impressed. Looking more closely, though, I realized that the checkbox states did not persist after the page reloaded. So, someone could spend a long time carefully tweaking their slides, only to accidentally hit F5 or crash their browser, and then—boom!—all their work would be lost. Christian told me that he was already working on storing the checkbox states in **localStorage**. Then, naturally, we endlessly debated the storage format. That debate inspired me to write this chapter, to explore the various approaches in depth.

Using An Array

We have two (reasonable) ways to model our data in an array. One is to store true/false values, like so: `[false, true, true, false, false, true, true]`

The other is to store an array of 0s and 1s, like so: `[0, 1, 1, 0, 0, 1, 1]`

Whichever solution we go with, we will ultimately have to convert it to a string, and then convert it back to an array when it is read. We have two ways to proceed: either with the old `Array#join()` (or `Array#toString()`) and `String#split()`, or with the fancier `JSON.stringify()` and `JSON.parse()`.

With the JSON way, the code will be somewhat shorter, although it is the JavaScript equivalent of slicing bread with a chainsaw. Not only there is a performance impact in most browsers³, but you're also cutting down browser support quite a bit.

The main drawback of using array-based strings is their size in bytes. If you go with the number method, you would use almost 2 characters per number (or, more precisely, $2N - 1$, since you'd need one delimiter per number, except for the last one): `[0, 1, 1, 0, 0, 1, 1].toString().length // 13, for 7 values`

So, for 512 numbers, that would be 1023 characters or 2 KB, since

JavaScript uses UTF-16⁴. If you go with the boolean method, it's even worse: `[false, true, true, false, false, true, true].toString().length // 37`, also for 7 values

That's around 5 to 6 characters per value, so 2560 to 3072 characters for 512 numbers (which is 5 to 6 KB). `JSON.stringify()` even wastes 2 more characters in each case, for the opening and closing brackets, but its advantage is that you get your original value types back with `JSON.parse()` instead of strings.

Using A String

Using a string saves some space, because no delimiters are involved. For example, if you go with the number approach and store strings like `'01001101010111'`, you are essentially storing one character per value, which is 100% better than the better of the two previous approaches. You can then get the values into an array by using

```
String#split: '01001101010111'.split(''); //  
['0', '1', '0', '0', '1', '1', '0', '1', '0', '1', '0', '1', '1', '1']
```

Or you could just loop over the string using `string.charAt(i)` —or even the string indexes (`string[i]`), if you don't care about older browsers.

Using Bitfields

Did the previous method make you think of binary numbers? It's not just you. The concept of bitfields⁵ is quite popular in other programming languages, but not so much in JavaScript. In a nutshell, bitfields are used to pack a lot of boolean values into **the bits** of the boolean representation of a number. For example, if you have eight values (true, false, false, true, false, true, true, false), the number would be 10010110 in binary; so, 150 in decimal and 96 in hex. That's 2 characters instead of 8, so **75% saved**. In general, 1 digit in the hex representation corresponds to exactly 4 bits. (That's because $16 = 2^4$. In general, in a $\text{base}2^n$ system, you can pack n bits into every $\text{base}2^n$ digit.) So, **we weren't lucky with that 75%; it's always that much**.

Thus, instead of storing that string as a string and using 1 character per value, we can be smarter and convert it to a (hex) number first. How do we do that? It's no more than a line of code: `parseInt('10010110', 2).toString(16); // returns '96'`

And how do we read it back? That's just as simple: `parseInt('96', 16).toString(2); // returns '10010110'`

From this point on, we can follow the same process as the previous method to loop over the values and do something useful with them.

Can We Do Better?

In fact, we can! Why convert it to a hex (base 16) number, which uses only 6 of the 26 alphabet letters? The `Number#toString()` method allows us to go up to base 36⁶ (throwing a `RangeError` for `>= 37`), which effectively uses *all* letters in the alphabet, all the way up to z! This way, we can have a compression of up to 6 characters for 32 values, which means saving up to 81.25% compared to the plain string method! And the code is just as simple: `parseInt('1001011000', 2).toString(36); // returns 'go' (instead of '258', which would be the hex version)`
`parseInt('go', 36).toString(2); // returns '1001011000'`

For some of you, this will be enough. But I can almost hear the more inquisitive minds out there shouting, “But we have capital letters, we have other symbols, we are still not using strings to their full potential!” And you'd be right. There is a reason why every time you open a binary file in a text editor, you get weird symbols mixed with numbers, uppercase letters, lowercase letters and whatnot. Every character in an UTF-16 string is a 2 bytes (16 bits), which means that if we use the right compression algorithm, we should be able to store 16 yes/no values in it (saving 93.75% from the string method).

The problem is that JavaScript doesn't offer a built-in way to do that, so the code becomes a bit more complicated.

Packing 16 Values Into One Character

You can use `String.fromCharCode` to get the individual characters. It accepts a numerical value of up to 65,535 and returns a character (and for values greater than that, it returns an empty string).

So, we have to split our string into chunks of 16 characters in size. We can do that through `.match(/.{1,16}/g)`⁷. To sum up, the full solution would look like this: `function pack(/* string */`

```
values) {  
    var chunks = values.match(/.{1,16}/g), packed =  
    '';  
    for (var i=0; i < chunks.length; i++) {  
        packed +=  
String.fromCharCode(parseInt(chunks[i], 2));  
    }  
    return packed;  
}
```

```
function unpack(/* string */ packed) {  
    var values = '';  
    for (var i=0; i < packed.length; i++) {  
        values += packed.charCodeAt(i).toString(2);  
    }  
    return values;  
}
```

It wasn't that hard, was it?

With these few lines of code, you can pack the aforementioned 512 values into—drum roll, please—**32 characters (64 bytes)**!

Quite an improvement over our original 2 KB (with the array method), isn't it?

Limitations

Numbers in JavaScript have limits. For the methods discussed here that involve an intermediate state of converting to a number, the limit appears to be **1023** yes/no values, because `parseInt('1111...1111', 2)` returns **Infinity** when the number of ones is bigger than 1023. This limit does not apply to the last method, because we're only converting blocks of bits instead of the whole thing. And, of course, it doesn't apply to the first two methods (array and string) because they don't involve packing the values into an integer.

“I Think You Took It A Bit Too Far”

This might be overkill for some cases. But it will definitely come in handy when you want to store a lot of boolean values in any limited space that can only store strings. And no optimization is overkill for things that go

can only store strings. And no optimization is overkill for things that go through the wire frequently. For example, cookies are sent on every single request, so they should be as tiny as possible. Another use case would be online multiplayer games, for which response times should be lightning-fast, otherwise the games wouldn't be fun.

And even if this kind of optimization isn't your thing, I hope you've found the thought process and the code involved educational. 🐼

—

1. <http://twitter.com/#!/codepo8>
2. <http://icant.co.uk/talks/koc2011/>
3. <http://jsperf.com/json-vs-split-join-for-simple-arrays>
4. <http://es5.github.com/#x4.3.16>
5. http://en.wikipedia.org/wiki/Bit_field
6. http://en.wikipedia.org/wiki/Base_36

7. <http://stackoverflow.com/questions/7033639/javascript-split-large-string-in-n-size-chunks>

Building A Relationship Between CSS & JavaScript

BY TIM WRIGHT 

jQuery, Prototype, Node.js, Backbone.js, Mustache and thousands of JavaScript microlibraries all combine into a single undeniable fact: **JavaScript is popular**. It's so popular, in fact, that we often find ourselves using it in places where another solution might be better in the long run.

Even though we keep JavaScript, CSS and HTML in different files, the concepts behind progressive enhancement are getting all knotted up with every jQuery plugin we use and with every weird technique that crops up. Because JavaScript is so powerful, there are a lot of overlaps in capability between JavaScript and HTML (building document structure) and JavaScript and CSS (injecting style information). I'm not here to pick on any JavaScript library, bootstrap or boilerplate; I'm just here to offer a little perspective as to where we are and how we can realign our goals.

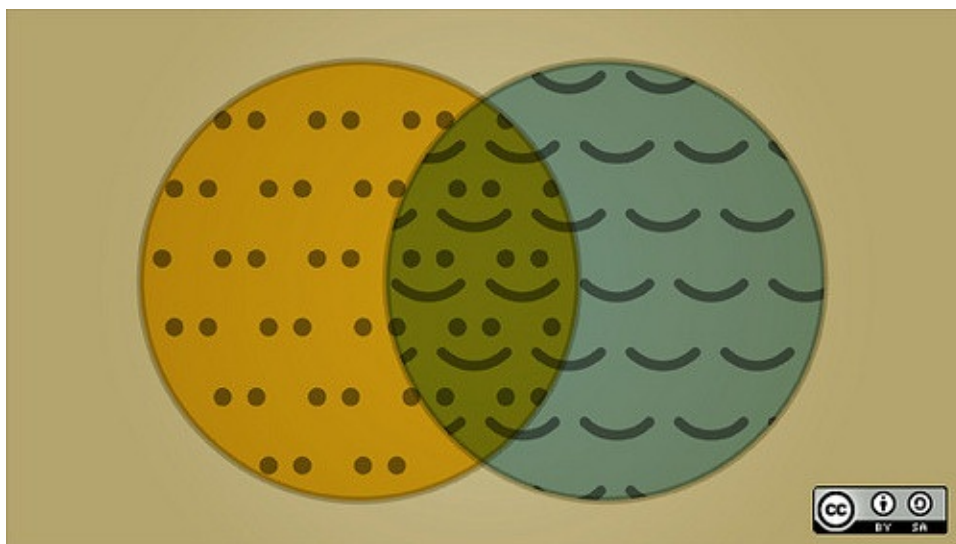


Image Credit: [opensourceway](#)¹.

Keeping CSS Out Of Your JavaScript

CSS can hook into HTML with a variety of different selectors; this isn't anything new. By using IDs, classes or any attribute you can think of (even custom attributes), you have easy access to style an element. You can also do this with a slew of JavaScript methods, and honestly, it's the same basic process with a different syntax (one of my JavaScript ah-ha moments). Being able to natively access HTML from JavaScript *and from* CSS is one of the reasons progressive enhancement has been such a successful development model. It allows a point of reference to guide us and to serve as a reminder as we develop a project, so we don't “cross the streams”².

But, as you move forward with JavaScript and build applications with highly interactive elements, it gets harder to not only keep HTML out of

your JavaScript, but also to catch yourself before injecting style information into a document. Of course, the case for not injecting style with JavaScript certainly isn't a binary one (yes/no, true/false, 0/1); there are plenty of cases where you might need to apply styles progressively, for example, in a drag and drop interface where positioning information needs to be constantly updated based on cursor (or finger) position.

But generally speaking, **you can safely house all the style information you need within CSS** and reference styles as reusable classes. This is a much more flexible model than sprinkling CSS throughout a JavaScript file, and it very closely compares to the model of adding style information into your HTML. We follow this model when it's only HTML and CSS, but for some reason it has a tendency to fall apart once JavaScript gets added into the mix. It's certainly something we need to keep an eye on.

A lot of front-end developers take real pride in having clean HTML. It's easy to work with, and to certain super-geeks it can even be artful. It's great to have clean, static HTML, but what good is that if your generated HTML is riddled with injected style and non-semantic markup? By “generated HTML,” I'm referencing how the HTML looks after it's been consumed and barfed back up after being passed around all those plugins and extra JavaScript. If step one to having clean HTML and separated progressive enhancement layers is to not use a **style** attribute, I'd have to say that step two is to avoid writing JavaScript that injects a **style** attribute for you.

CLEANING UP YOUR HTML

We can probably all agree that blindly using a technology is a terrible idea, and I think we're at a point with jQuery where we are, indeed, blindly using a lot of the features without fully understanding what's going on under the hood. The example I lean on pretty heavily for keeping CSS out of my JavaScript is the behavior of jQuery's `hide()` method. Based on the principles of progressive enhancement, you wouldn't code something with inline CSS like this: `<div class="content-area" style="display:none;"></div>`

We don't do that because a screen reader won't pick up an element if the style is set to `display:none`, and it also muddies up the HTML with unnecessary presentational information. When you use a jQuery method like `hide()`, that's exactly what it does: it will set a `style` attribute on the target area and add a display property of `none`. It's **very easy to implement, but not very good for accessibility**. It also violates the principles of progressive enhancement when you inject style into the document like that (we're all sorts of messed up, huh?). It's not uncommon for this method to be used within a tabbing interface to hide content. The result is that the content is nonexistent to a screen reader. Once we realize that adding style from JavaScript isn't ideal in most cases, we can move it into the CSS and reference it as a class: **CSS**

```
.hide {  
    display: none;  
}
```

jQuery

```
$( '.content-area' ).addClass( 'hide' );
```

We still have to address the accessibility problem of hiding content with **display:none**, but since we're not using a built-in jQuery method anymore, we can control exactly how content gets hidden (whichever accessible method you prefer is probably fine). For example we could do something like: **CSS**

```
.hide {  
    position: absolute;  
    top: -9999px;  
    left: -9999px;  
}  
  
.remove {  
    display: none;  
}
```

In the above example, you can see that even though both classes result in content being removed from view, they function very differently from an accessibility standpoint. Looking at the code like this makes it clear that we really are dealing with style information that belongs in a CSS file. Using utility classes in this way can not only help your JavaScript slim down, but also have double usage in an Object Oriented CSS (OOCSS)

development model. This is truly **a way to not repeat yourself** (Don't Repeat Yourself, or DRY) within CSS, but also across a whole project, creating a more holistic approach to front-end development. Personally, I see a lot of benefit in controlling your behaviors this way, but some people have also called me a control-freak in the past.

WEB AND TEAM ENVIRONMENTS

This is a way we can start opening up lines of communication between CSS and JavaScript and lean on the strengths of each language without overdoing it. Creating **a developmental balance on the front end is very important**, because the environment is so fragile and we can't control it like we can on the back end with a server. If a user's browser is old and slow, most of the time you can't sit down and upgrade it (aside: I do have my grandmother using Chrome); all you can do is embrace the environmental chaos, build for the best and plan for the worst.

Some people have argued with me in the past that this style of development, where you're referencing CSS classes in JavaScript, doesn't work well in team development environments because the CSS is usually built by the time you're diving into the JavaScript, which can cause these classes to get lost in the mix and create a lot of inconsistency in the code (the opposite of DRY). To those people I say: poke your head over the cube wall, open AIM, GTalk or Skype, and communicate to the rest of the team that these classes exist specifically to be used with JavaScript. I know the concept of developers communicating outside of GIT commit messages seems like madness

communicating outside of Git commit messages seems like madness, but it'll be okay, I promise.

Using Behavioral CSS With JavaScript Fallbacks

Using these CSS objects as hooks for JavaScript can go far beyond simple hiding and showing of content into an area of behavioral CSS, transitions, animations and transforms that are often done with JavaScript animations. With that in mind, let's take a look at a common interaction model of fading out a **div** on click, and see how it would be set up with this development model, while providing the proper fallbacks for browsers that might not support the CSS transition we're going to use.

For this example we'll be using: jQuery³

- Modernizr⁴

First, let's set up our **body** element: `<body>`

```
<button type="button">Run Transition</button>
<div id="cube"></div><!--/#cube-->
</body>
```

From there we'll need to set up the CSS: `#cube {`
`height: 200px;`

```
width: 200px;
background: orange;
-webkit-transition: opacity linear .5s;
  -moz-transition: opacity linear .5s;
   -o-transition: opacity linear .5s;
    transition: opacity linear .5s;
}

.fade-out {
  opacity: 0;
}
```

Before we add on the JavaScript layer, **lets take a moment and talk about the flow of what's going to happen**. Use Modernizr to check for CSS Transition support. If Yes, a) set up a click event on the button to add a “fade-out” class to **#cube** and b) Add another event listener to catch when the transition is finished so we can time the execution of a function that will remove **#cube** from the DOM.

- If No, a) Set up a click even on the button to use jQuery's **animate()** method to manually fade **#cube** out and b) Execute a callback function to remove **#cube** from the DOM.

This process will introduce a new event called **transitionend**, which will execute at the end of a CSS transition. It's amazing, FYI. There is also a companion event called **animationend**, which will execute at the end of a CSS animation for more complex interactions.

First thing we need to do is set up our variables in the JavaScript:

```
(function () {  
  
    // set up your variables  
    var elem = document.getElementById('cube'),  
        button = document.getElementById('do-it'),  
        transitionTimingFunction = 'linear',  
        transitionDuration = 500,  
        transitionend;  
  
    // set up the syntax of the transitionend event  
    with proper vendor prefixes  
    if ($.browser.webkit) {  
        transitionend = 'webkitTransitionEnd'; //  
safari & chrome  
    } else if ($.browser.mozilla) {  
        transitionend = 'transitionend'; // firefox  
    } else if ($.browser.opera) {  
        transitionend = 'oTransitionEnd'; // opera  
    } else {  
        transitionend = 'transitionend'; // best guess  
at the default?  
    }  
  
    //... rest of the code goes here.  
  
})(); // end wrapping function
```

You might notice that our new **transitionend** event needs a vendor prefix; we're doing a little browser detection to take care of that. Normally you might detect for the vendor prefix and add it onto the event name, but in this instance the cases for the syntaxes are a little different, so we need to get the whole name of the event for each prefix.

In the next step we'll use Modernizr to detect support, and add our event listeners to each case (all of this stuff gets added inside the wrapping function): `// detect for css transition support with Modernizr`

```
if(Modernizr.csstransitions) {

    // add our class on click
    $(button).on('click', function () {
        $(elem).addClass('fade-out');
    });

    // simulate a callback function with an event
    listener
    elem.addEventListener(transitionend, function () {
        theCallbackFunction(elem);
    }, false);

} else {

    // set up a normal click/animate listener for
    unsupported browsers
    $(button).on('click', function () {
```

```
        $(elem).animate({
            'opacity' : '0'
        }, transitionDuration,
transitionTimingFunction, function () {
            theCallbackFunction(elem);
        });

    }); // end click event

} // end support check
```

Finally, we need to define a shared function between the two actions (DRY) which executes after the transition (or animation) is complete. For the sake of this demonstration we can just call it **theCallbackFunction()** (even though it's not technically a callback function). It will remove an element from the DOM and spit out a message in the console letting us know that it worked.

```
// define your callback function, what happens after
the transition/animation
function theCallbackFunction (elem) {

    'use strict';

    // remove the element from the DOM
    $(elem).remove();

    // log out that the transition is done
```

```
console.log('the transition is complete');  
  
}
```

In the browser, this should work the same way in IE 7 (on the low end) as it does in mobile Safari or Chrome for Mobile (on the high end). The only difference is under the hood; the experience never changes for the user. This is a way you can use cutting-edge techniques without sacrificing the degraded user experience. It also keeps CSS out of your JavaScript, which was really our goal the whole time.

The Moral Of The Story

You might be asking yourself why we should even bother going through all this work. We wrote about 60 lines of JavaScript to accomplish the same design aesthetic that could be created with eight lines of jQuery. Well, no one ever said that keeping clean code and sticking to progressive enhancement was the easiest thing to do. In fact, it's a lot easier to ignore it entirely. But as responsible developers, it's our duty to build applications in a way that is accessible and easily scales to the future. If you want to go that extra mile and create a seamless user experience as I do, then it's well worth the extra time it takes to dot all the i's and cross all the t's in a project to create an overall experience that will gracefully degrade and progressively enhance.

Using this model also lets us lean heavily on CSS for its strengths, like responsive design and using breakpoints to redefine your interaction at the various screen sizes. It also helps if you're specifically targeting a device with a constrained bandwidth, because, as we all know, CSS is much lighter than JavaScript in both download and execution time. **Being able to offload some of the weight JavaScript carries onto CSS is a great benefit.**

In production, we are currently using CSS animations and transitions for micro interactions like hover effects and maybe a spinning graphic or a pulsating knot. We've come to a point where CSS is a pretty powerful language that performs very well in the browser and it's okay to use it more heavily for those macro interactions that are typically built using JavaScript. If you're looking for a lightweight and consistent experience that's relatively easy to maintain while allowing you to use the latest and greatest browser features — it's probably time to start mending fences and build strength back into the relationship between CSS and JavaScript. As a great man once said, “The key to writing great JavaScript is knowing when to use CSS instead.” (It was me... I said that.) 🐼

— <http://www.flickr.com/photos/opensourceway/5755219051/>

<http://www.youtube.com/watch?v=jyaLZHiJJnE>

<http://jquery.com>

<http://modernizr.com>

About The Authors

Ben Howdle

Ben Howdle is a JavaScript Developer from London, UK. He works by day at KashFlow¹ and by night and weekends on Two Step Media². A JavaScript guy building slick UIs.

Christian Heilmann

An international Developer Evangelist working for Mozilla in the lovely town of London, England. You can find him on Wait-Till-I.com³ and follow him on Twitter⁴.

Daniel Sternlicht

Daniel Sternlicht is a Front End Developer based on Israel who has a great passion for UX and Web Design. He specializes in CSS3, HTML5, and JavaScript. In his free time you'll find him exploring new web technologies, developing cool web apps and make his wife happy :)
Portfolio⁵ | Blog⁶ | Dribbble⁷ | Twitter⁸

Derek Mack

Derek Mack is a designer from Melbourne, Australia. He is one half of The Graphic Order⁹, designers of Verbs IM¹⁰ for iOS and the award-winning Aussie Rules Live¹¹ for iOS and Android.

Jon Raasch

Jon Raasch is the author of the book Smashing Webkit¹². He's a freelance front-end web developer¹³ and UI designer with endless love for jQuery, CSS3, HTML5 and performance tuning. Follow him on Twitter¹⁴ or read his blog¹⁵.

Lea Verou

Lea works as a Developer Advocate for W3C¹⁶. She has a long-standing passion for open web standards, which she fulfills by researching new ways to use them, blogging¹⁷, speaking¹⁸, writing¹⁹, and coding popular open source projects²⁰ to help fellow developers. She is a member of the CSS Working Group²¹, which architects the language itself. Lea studied Computer Science in Athens University of Economics and Business,

where she co-organized and occasionally lectured a cutting edge Web development course²² for 4th year undergrads. She is one of the few misfits who love code and design almost equally.

Peter Wilson

Peter Wilson is a Web developer based in Melbourne, Australia, and started making Websites in 1994. Peter co-founded web production studio Soupgiant²³ in 2009 and forms opinions on all things web at Big Red Tin²⁴ and on Twitter²⁵.

Phil Leggetter

Phil Leggetter is a Developer Evangelist at Pusher²⁶. He's been involved in developing and using realtime web technologies for over 10 years. His focus is to help developers use these technologies to build the next generation of interactive and engaging realtime web applications. Follow him at @leggetter²⁷.

Tim Wright

Tim Wright is the Senior User Interface Designer and Developer at FreshTilledSoil²⁸ - a team of designers, coders and UX experts that helps entrepreneurs and businesses create bloody brilliant user experiences for web and mobile applications through consulting, training, and events. Tim is a frequent speaker, blogger, article writer, and published author. You might call him the Hemingway of the development world. His knowledge of JavaScript is unmatched. In fact, he wrote the book on it: Learning JavaScript: A Hands-On Guide to the Fundamentals of Modern JavaScript.

About Smashing Magazine

Smashing Magazine²⁹ is an online magazine dedicated to Web designers and developers worldwide. Its rigorous quality control and thorough editorial work has gathered a devoted community exceeding half a million subscribers, followers and fans. Each and every published article is carefully prepared, edited, reviewed and curated according to the high quality standards set in Smashing Magazine's own publishing policy³⁰.

Smashing Magazine publishes articles on a daily basis with topics ranging from business, visual design, typography, front-end as well as back-end development, all the way to usability and user experience design. The magazine is—and always has been—a professional and independent online publication neither controlled nor influenced by any third parties, delivering content in the best interest of its readers. These

guidelines are continually revised and updated to assure that the quality of the published content is never compromised.

About Smashing Media GmbH

Smashing Media GmbH³¹ is one of the world's leading online publishing companies in the field of Web design. Founded in 2009 by Sven Lennartz and Vitaly Friedman, the company's headquarters is situated in southern Germany, in the sunny city of Freiburg im Breisgau. Smashing Media's lead publication, Smashing Magazine, has gained worldwide attention since its emergence back in 2006, and is supported by the vast, global Smashing community and readership. Smashing Magazine had proven to be a trustworthy online source containing high quality articles on progressive design and coding techniques as well as recent developments in the Web design industry.

—

1. <http://kashflow.com/>
2. <http://twostepmedia.co.uk/>

3. <http://www.wait-till-i.com/>
4. <http://www.twitter.com/codepo8>
5. <http://danielsternlicht.com/>
6. <http://gandtblog.com/>
7. <http://dribbble.com/fastrd>
8. <http://twitter.com/#!/gandtblog>
9. <http://dribbble.com/alanvanroemburg>
10. <http://verbs.im/>
11. <http://www.sportsmatemobile.com/>
12. <http://www.amazon.com/Smashing-WebKit-Magazine-Book/dp/1119999138>
13. <http://jonraasch.com/>
14. <http://twitter.com/jonraasch>

15. <http://jonraasch.com/blog/>
16. <http://w3.org/>
17. <http://lea.verou.me/>
18. <http://lea.verou.me/speaking>
19. <http://lea.verou.me/publications>
20. <http://lea.verou.me/projects/>
21. <http://www.w3.org/Style/CSS/members.en.php3>
22. <http://lea.verou.me/2010/07/organizing-a-university-course-on-modern-web-development/>
23. <http://soupgiant.com/>
24. <http://peterwilson.cc/>
25. <http://twitter.com/pwcc/>

26. <http://pusher.com/>

27. <http://twitter.com/leggetter>

28. <http://www.freshtilledsoil.com/#welcome>

29. <http://www.smashingmagazine.com>

30. <http://www.smashingmagazine.com/publishing-policy/>

31. <http://www.smashing-media.com>