# [Modern Web Development - Part 1](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)

Jan 18, 2012 at 3:30 AM

by [Shawn Wildermuth](http://wildermuth.com/me)

This is the first of ten parts of this blog post. The topics will be:

1. A New World (This post)
2. [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
3. [A Better CSS](http://wildermuth.com/2012/01/31/Modern_Web_Development_-_Part_3)
4. [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
5. [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
6. [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
7. [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
8. [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
9. [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
10. Deploying to the Cloud (upcoming)

In the past year I’ve had a side project. FirstInked’s Beta recently shipped and I wanted to share with you what I’ve learned. To start out, I want to specifically thank two people who were really great in helping me formulate the strategies I’ll talk about. They are:

* Dave Ward ([@encosia](http://twitter.com/Encosia)) answered every twitter pleading with great advice.
* Chris Rauber ([@chrisrauber](http://twitter.com/chrisrauber)) was instrumental in doing the initial groundwork on the web project.

### Where I Came From

I am not new to website development, but with most of my work taking me into the RIA space five years ago, a lot had changed in the interim. Back 10 years ago, much of the web code I saw and wrote looked a lot like this (and no, my blog code doesn’t look like this):

<html>

<head>

  <script type="text/javascript">

    function onInit() {

      var obj = document.getElementById("foo");

      foo.display = "block";

    }

  </script>

<head>

<body onload="onInit()">

  <div id="foo" style="display: none" height="100px">

    <font size="3" color="red">Hello World</font>

</div>

</body>

The mixing of the code, markup and styling was the de facto style back then but that doesn’t mean it was a good idea. It was easier with ASP.NET (classic) to do things this way. I relied a lot more on post-backs than client-side code which wasn’t the best experience for users. Something had to give.

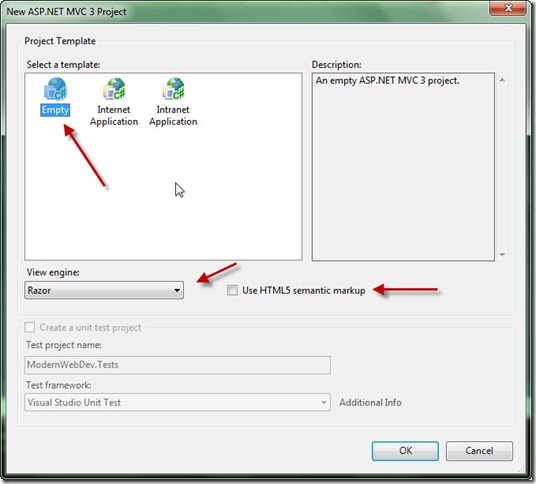
### What Has Changed?

While I’ve dabbled in Routing and MVC for the server-side code, I hadn’t gotten knee deep into the client-side richness now possible with the likes of jQuery and CSS3. ASP.NET MVC 3 has a lot to offer the web developer, but for this article series I am going to focus in on the client-side story since that is where most of the big changes are (at least in my eyes). I found that in building FirstInked that I spent about 20% of my time on the server code and the other 80% on the client code. This number is undoubtedly skewed by the fact that I am super comfortable with the server code and was learning the client code. But I think those numbers indicate a larger sense of what is happening. If your site isn’t about just displaying information (e.g. just information like a news site or blog) that you will be creating a lot more client-code than back in the first generation of web applications.

### An Example

To set some context, let’s take a quick example and build a simple home page using the tools I’ll be talking about.

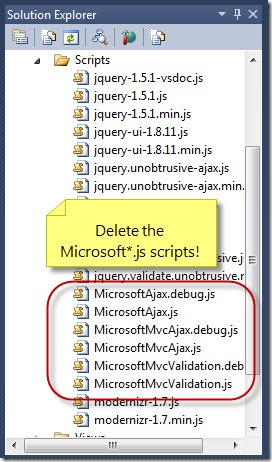
To start out, I create a new ASP.NET MVC 3 Project. This opens the “New ASP.NET MVC 3 Project” template page:

[](http://wildermuth.com/images/NewMvc_2.jpg)

For any new development I do, I start with an empty project (though the Internet Application can be used if you want authentication and some basic start pages). I will be using Razor and HTML5 so I want to pick both the View Engine and make sure I click the “Use HTML5 semantic markup” checkbox. You can create a unit test project if you like (and you should) but I am going to skip that part for this series.

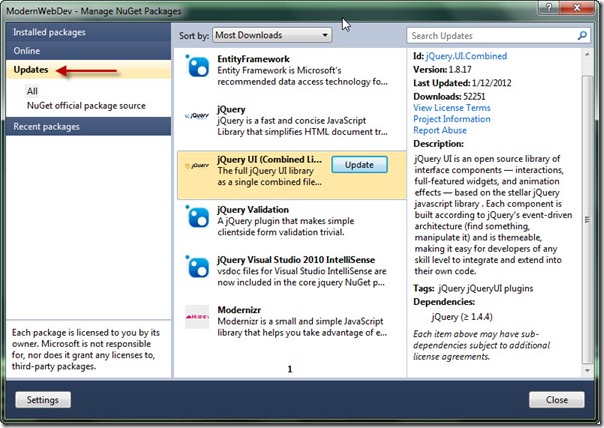
I start with a simple ASP.NET MVC 3 Empty project (for simplicity). The empty project has no controllers or views yet (though it has a \_Layout…e.g. master page using HTML5). I want to clean up the project to be ready for what I want to build.

First, I look in the Scripts folder to see what scripts were added to the project by default. As I won’t be using any of the Microsoft AJAX stuff, I delete them:



I am relying completely on the jQuery stack for my development so the Microsoft scripts aren’t needed.

The next thing I do is upgrade the version of jQuery. The stock ASP.NET MVC 3 ships with the 1.5.1 version of jQuery but using the package manager (assuming you’re using [Nuget](http://nuget.org/" \t "_blank), which if you aren’t you should be) you can just upgrade both jQuery and jQuery UI which I will use extensively. When you open the “Manage NuGet Packages” dialog, you will want to click on “Updates” to update some of the built-in packages:

[](http://wildermuth.com/images/nuget_2.jpg)

I always update the JQuery UI library (since that will update the jQuery library too). You may want to update all the packages, but these are the two that are key to what we’ll talk about. Now that we’ve updated the packages, we’ll need to update the versions on the layout page:

<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8" />

  <title>@ViewBag.Title</title>

  <link href="@Url.Content("~/Content/Site.css")"

        rel="stylesheet"

        type="text/css" />

  <script src="@Url.Content("~/Scripts/jquery-1.7.1.min.js")"

          type="text/javascript"></script>

  <script src="@Url.Content("~/Scripts/modernizr-2.0.6-development.min.js")"

          type="text/javascript"></script>

</head>

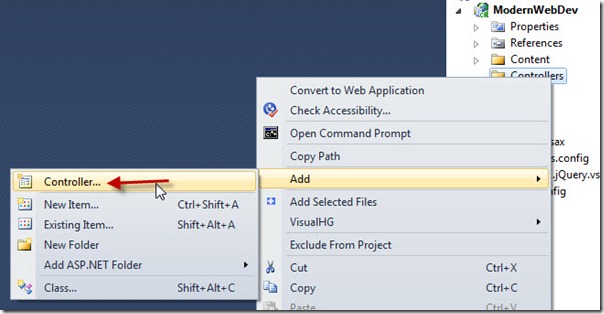
<body>

  @RenderBody()

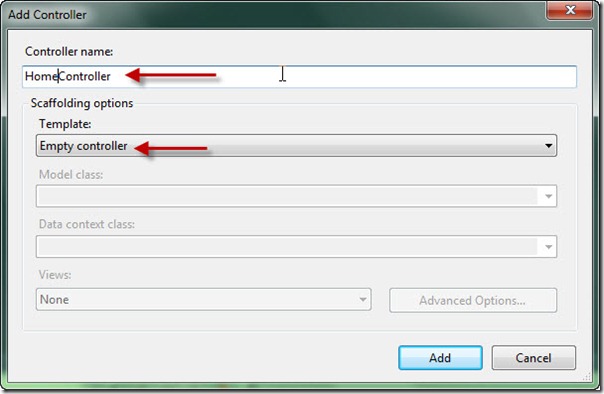
</body>

</html>

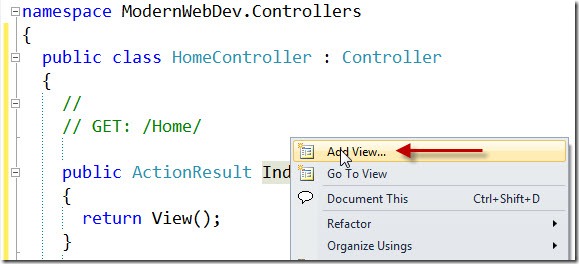
Now that I have the project ready, I want to add a new controller so I right-click the controller folder and add a new Controller:

[](http://wildermuth.com/images/newcontroller_2.jpg)

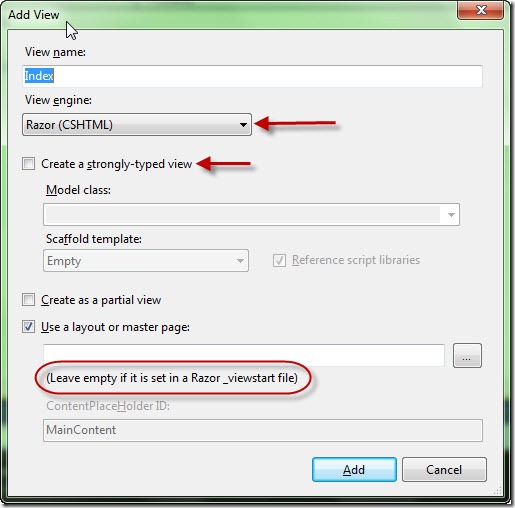
For my needs I want an empty controller (as the scaffolding of a model doesn’t make sense for this simple example):



This results in a new controller file with a single result for “Index”. To create the view I need to right-click the Index method and pick AddView:



This will bring up the Add View Dialog:



I am leaving the view name so that the controller will find this view when I navigate to it. Again, I am only using Razor for my code so I leave that too. If I were using a model class, I’d use a strongly-typed view but for my needs, no model class. Finally, I want to use MVC 3’s Layout class so I leave the master page empty. This leaves us with a very simple home view razor file:

@{

  ViewBag.Title = "Index";

}

<h2>Index</h2>

I am going to focus on the client-side so I won’t put much razor syntax in this file. I have found that keeping a per-view .js and .css file for each page that requires it (not every page will require specialized styling or code, but for this example, I’ll use both). So my new razor file is:

@{

  ViewBag.Title = "Index";

}

<link rel="stylesheet"

      href="@Url.Content("~/Content/Home.Index.css")" />

<h2>

  Index</h2>

<section id="main-section">

  <div>

    This is a client-side example!</div>

  <button>

    Click me to add some elements!</button>

</section>

<script type="text/javascript"

        src="@Url.Content("~/Scripts/Home.Index.js")"></script>

(For you MVC veterans, you are already thinking ahead, be patient and stop trying to fix problems…we’re going to get to it.)

This is a big part of what I want to show is that separating the concerns into structure, look and code (e.g. Markup, CSS and JavaScript).

Since I am using jQuery, my JavaScript will just add some event handlers via jQuery in the document ready handler:

$(document).ready(function () {

  $("#main-section button").on("click", function () {

    $("#main-section").append("<div>New Div</div>");

  });

});

(Again, if you’re a jQuery veteran…wait for the bigger picture in upcoming posts…promise!)

Finally, I will style the page with some CSS:

#main-section

{

  font-size: 85%;

  border: black 1px solid;

  background-color: #ddd;

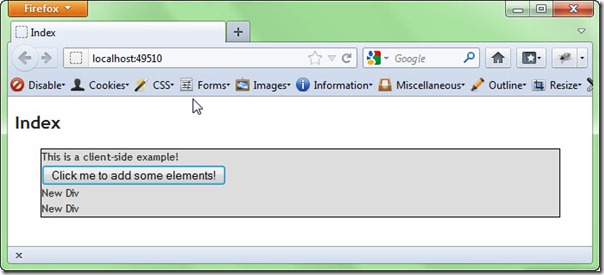
  width: 85%;

  margin-left: auto;

  margin-right: auto;

}

Putting this all together, we get a page that handles the mouse click and some basic styling:

[](http://wildermuth.com/images/example_2.jpg)

You’ll notice that I named the JavaScript and the CSS based on the view and that’s mostly for discoverability while I am doing development. These scripts/stylesheets will be in addition to site-wide scripts and we’ll see later a better way of architecting this. But the general idea to take from this first part is that each view/partialview will consist of mostly these three parts:

* Markup (HTML/Razor)
* Design (CSS)
* Code (JavaScript)

My goal is to finish this series in the next couple of weeks so keep an eye on a part every 2-3 days. You can get the source for what we have so far here:

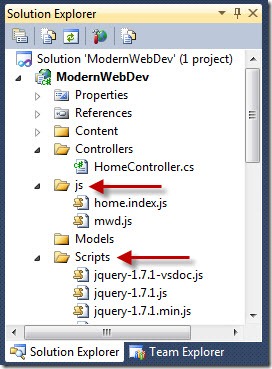
[Source](http://wildermuth.com/downloads/modernwebdev_1.zip)

## [Modern Web Development - Part 2](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)

## ****Architecting JavaScript****

Working with JavaScript can be daunting. In the past, I’ve seen some projects with just a handful of huge files that become difficult to manage. So in architecting what I needed to build, I wanted to adhere to the idea that there was common code and there was view-specific code. There are two different classes of JavaScript that I care about: libraries (i.e. not my code) and site code (i.e. my code).

For now, let’s segregate these two types of code into separate directories as shown below:

[](http://wildermuth.com/images/web1_2.jpg)

The scheme of these directories doesn’t matter, but I decided on keeping the Scripts directory since Nuget packages like to update to the Scripts folder. You’ll see in a later post when I talk about packaging why I separate these.  Most of the time I am ignoring “not my code” and opening “my code” quite a lot.

One strategy I’m using is to load all the scripts on every page. Our total size of my scripts is not enormous and I expect that the initial hit of caching the script(s) on the first page will mean that subsequent pages will be faster. I’ll talk about how we are going to package the scripts in a later part of this series, but for now I can just include all the scripts of our project in the ‘master page’ (i.e. \_Layout view):

<!-- \_Layout.cshtml -->

<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8" />

  <title>@ViewBag.Title</title>

  <link href="@Url.Content("~/Content/Site.css")"

        rel="stylesheet"

        type="text/css" />

  @RenderSection("StyleSheets", required: false)

</head>

<body>

  @RenderBody()

</body>

</html>

<!-- We'll use a packager later, don't freak out guys -->

<script src="@Url.Content("~/Scripts/jquery-1.7.1.min.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/modernizr-2.0.6-development-only.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/js/mwd.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/js/Home.Index.js")"

        type="text/javascript"></script>

This means that each page doesn’t need to add their own scripts (in fact, when I package them, I’ll just include all the scripts in the /js/ directory). This should result in less friction at the cost of an initial page load of a larger script which I am not worried about for desktop use (for mobile, I’ll change this strategy as you’ll see in a later part of the series).

Originally I had planned on having a script per page (if necessary) then just including it on the pages required but that meant I needed to be aware of code collisions and inclusion on each page. To combat this a JavaScript namespace can help segment all of my code in it’s own object so I don’t pollute the global namespace. This is just like in .NET where you will create namespaces for your classes. Though JavaScript doesn’t have a namespace as such so a named object is good enough to do the trick:

// mwd.js

(function (m, $) {

  m.setStatusMessage = function (message) {

    $('#status-message').text(message);

  }

} (window.mwd = window.mwd || {}, jQuery));

This works because of line 7 (highlighted). It executes this anonymous function creating the “mwd” object if it doesn’t exist. It passes in this object (as the m variable) so I can add any global data/functions to the object. The first script to run will not find the ‘window.mwd’ object so it will create a brand new object. Subsequent scripts (which will also have this function wrapper around it) will find the ‘mwd’ object and just pass it in. This way every script that uses this convention will add to the namespace object with our code and data. This simplifies the problem of script loading ordering problems and prevent name collision with objects in the global namespace. You can see that in the above that the mwd object is having a new function added to it. Since it’s an object on the window object, we can access this function from any script but only exposing a single global object.

For page specific code, the function wrapper is still used:

// home.index.js

(function (m, $, undefined) {

  m.initHomeIndex = function () {

    $("#main-section button").on("click", function () {

      m.setStatusMessage("clicked");

      // This works too

      //mwd.setStatusMessage("clicked");

    });

  };

} (window.mwd = window.mwd || {}, jQuery));

This approach of using an initialization function works as this script just registers the startup function for this page but doesn’t call it. This allows me to call the script on my individual page. This is especially helpful if I need to send in some state to the initialization (i.e. some data from the controller).

You can see the highlighted line is calling a common function that is defined in a different .js file (the mwd.js) and this just works as this code isn’t executed until after all the extensions to the mwd object are already added by all the scripts.

My method for calling the initialization function is to use an optional section on the Layout view that wraps whatever the individual page uses for an initialization (though not all pages will need initialization functions):

<!-- \_Layout.cshtml -->

...

<!-- I'll use a packager later, don't freak out guys -->

<script src="@Url.Content("~/Scripts/jquery-1.7.1.min.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/modernizr-2.0.6-development-only.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/js/mwd.js")"

        type="text/javascript"></script>

<script src="@Url.Content("~/js/Home.Index.js")"

        type="text/javascript"></script>

@if (IsSectionDefined("Init"))

{

  <script type="text/javascript">

    $(document).ready(function() {

      @RenderSection("Init", required: false)

    });

  </script>

}

This way if the Init section is defined on a view, I wrap the init code in a jQuery ready call so that it gets called once the page has loaded.  The individual view looks like so:

@{

  ViewBag.Title = "Index";

}

@section Stylesheets

{

  <link rel="stylesheet" href="@Url.Content("~/Content/Home.Index.css")" />

}

@section Init

{

  mwd.initHomeIndex();

}

<h2>

  Index</h2>

<section id="main-section">

  <div>

    This is a client-side example!</div>

  <button>

    Show Message</button>

</section>

<footer>

  <div id="status-message"></div>

</footer>

If this were a strongly typed view I could have send date into the initHomeIndex function by JSON serializing onto the page.  This allows me to efficiently have my view code added to every view but only call it on the particular views I need it. This is not just for calls that are necessary per view, but could be any shared code as well.

There are other strategies for doing this (as [Dave Ward (@encosia)](http://twitter.com/encosia) explained using a class on a body tag with the controller and view name) but for our needs this was the most straightforward since we needed initialization data on a number of different views.

This is a pretty universal way of handling this that worked well for me. One thing I ran into was that our Layout view wanted to have some code/markup that was only to be used during release builds (i.e. Google Analytics), I was surprised that Razor didn’t allow me to test for #DEBUG builds. To fix this, I added an extension method to the HtmlHelper class:

using System.Web.Mvc;

namespace ModernWebDev.Helpers

{

  public static class HelpHelpersExtensions

  {

    public static bool IsDebug(this HtmlHelper htmlHelper)

    {

#if DEBUG

      return true;

#else

      return false;

#endif

    }

  }

}

As this HtmlHelpersExtensions class would probably be a placeholder for a number of extensions for our use, I changed the web.config in the Views directory (not the site-wide one) to include my Helpers namespace on every view:

<?xml version="1.0"?>

<configuration>

  ...

  <system.web.webPages.razor>

    <host factoryType="..." />

    <pages pageBaseType="System.Web.Mvc.WebViewPage">

      <namespaces>

        <add namespace="System.Web.Mvc" />

        <add namespace="System.Web.Mvc.Ajax" />

        <add namespace="System.Web.Mvc.Html" />

        <add namespace="System.Web.Routing" />

        <add namespace="ModernWebDev.Helpers" />

      </namespaces>

    </pages>

  </system.web.webPages.razor>

...

</configuration>

This in a convenience and not necessary, but knowing you can add namespaces to every razor view is a great help so you don’t have to annotate a number of @using statements at the top of every view.

I could then use the new extension method to have conditional code in our Layout view:

<!-- \_Layout.cshtml -->

...

@if (!Html.IsDebug())

{

  <div>Placeholder for Google Analytics</div>

}

What I’ve shown you so far may have some of you scratching your head because the separation of these JavaScript files means that you’re probably not getting any intellisense. I made a decision to just forgo it. If Intellisense is crucial to you, there are some options (none of these were worth my trouble):

* Adding a comment on top of every .js file to the Visual Studio intellisense file to get jQuery intellisense but it didn’t help with overall intellisense.
* Using a 3rd party plugin (i.e. [R#](http://www.jetbrains.com/resharper/)) to add the intellisense.
* Use a non-Visual Studio editor (i.e. [WebStorm](http://www.jetbrains.com/webstorm/" \t "_blank) or other editors)

All of these are adequate solutions, but they all seemed to slow down my development and I am an old guy who can get by without it.  YMMV.

Much of the process of writing the JavaScript can’t be adequately summarized until we get to the debugging process I used in this project which I will cover in the next part of this series. Good hunting!

Here’s a link to the version of the source code so far:

* [Source Code](http://wildermuth.com/downloads/modernwebdev_2.zip)

[**Modern Web Development - Part 3**](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)

Jan 31, 2012 at 8:01 AM

Tags: [ASP.NET MVC](http://wildermuth.com/tags/ASP.NET%20MVC) [HTML5](http://wildermuth.com/tags/HTML5) [CSS](http://wildermuth.com/tags/CSS)

[24 Comments](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)

by [Shawn Wildermuth](http://wildermuth.com/me)

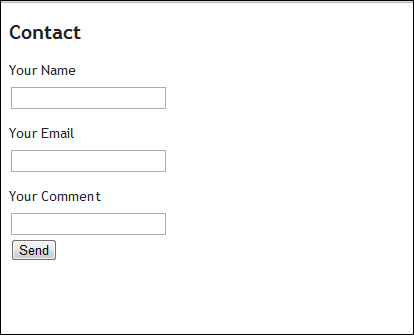
This is the third of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: A Better CSS (This post)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

**What’s Wrong with CSS?**

If you’re going to do web development, you’ll need to learn how cascading style sheets (CSS) work. It’s a fine system for defining the look and feel of your designs but as a developer I find them more painful than necessary. Let’s discuss some of those pains.

So let’s assume I was writing a simple contact page form. So I’ll have a number of rules for the different parts of the form (I would probably not have rules specifically for the contact form, but have a common set of rules, but for this example it’s good enough):



It’s plain enough, so let’s make it pretty with some CSS:

/\* home-contact.css \*/

h2

{

  font-style: italic;

  color: #222;

}

.big-form input[type=text]

{

  width: 300px;

  height: 25px;

  padding: 5px;

  border-radius: 5px;

  border: 1px black solid;

  background-color: #EEE;

  color: #222;

  margin: 2px 0;

  font-size: 18px;

}

.big-form label

{

  color: #222;

  font-size: 14px;

}

This works, but there are some common problems with this including repeating of values, repeating of selector syntax, no good way to handle shims or polyfills as well as no way to calculate values. In this way, CSS is purely content and presents maintenance issues.

**A Better CSS?**

To the rescue are dynamic stylesheet languages. Among these are:

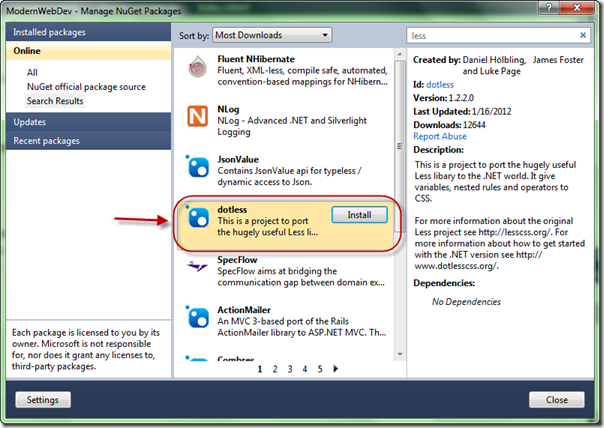
* [Less](http://www.lesscss.org/): Client-side or server-side supported language for solving stylesheet problems.
* [Sass](http://sass-lang.com/): Command-line or server-side supported language for solving stylesheet problems.

Both of these languages are similar in scope as they support features like:

* Nested Styles
* Variables
* Mixins

They are both great languages the key importance here is that you should stop writing plain old CSS and use a dynamic stylesheet language.

While Sass is a bit more powerful, I decided on Less as I found it easier to integrate with ASP.NET MVC. There are a number of ways to do this, but for me the least amount of friction came with a simple package I could install to my web project called [dotless](http://www.dotlesscss.org/" \t "_blank). This project can be installed via Nuget:

[](http://wildermuth.com/images/dotless_2.png)

[Dotless](http://www.dotlesscss.org/) installs a http handler so that if you refer to .less files in your stylesheet declarations, it returns the generated CSS. Some might prefer to compile to .css at compile time, but since the .js can be changed at runtime too, I though this was the best mix of the two solutions. So let’s see how LESS can fix this for us.

**Getting Less Working in ASP.NET MVC**

One of the basic concepts around these dynamic stylesheet languages is that CSS is completely valid. So if I change my .css file to .less (and change the link to point to the .less file instead):

@\* Contact.cshmtl \*@

@model ModernWebDev.Models.SomeFormModel

@{

  ViewBag.Title = "Contact Us!";

}

@section Stylesheets

{

  <link rel="stylesheet" href="@Url.Content("~/Content/Home.Contact.less")" />

}

<h2>

  Contact</h2>

At this point the page still works because the new .less stylesheet is returned as the CSS it contains. Let’s refactor the CSS to use some of our new found power.

**Refactoring the CSS with Less**

The first change I want to make is to centralize the text color so that if I change it, I don’t have to search and replace. Less has the concept of variables. These variables start with the ‘@’ symbol like so:

@variable\_name: value;

This means it can add a variable to hold our color at the top of our less file and use it everywhere:

/\* home-contact.less \*/

@text-color: #222;

h2

{

  font-style: italic;

  color: @text-color;

}

.big-form input[type=text]

{

  width: 300px;

  height: 25px;

  padding: 5px;

  border-radius: 5px;

  border: 1px black solid;

  background-color: #EEE;

  color: @text-color;

  margin: 2px 0;

  font-size: 18px;

}

.big-form label

{

  color: @text-color;

  font-size: 14px;

}

While the variables tend to be constants, you can perform operations on them too. For example, I could use another variable to store the base size of my font then use operations to change the value as necessary:

/\* home-contact.less \*/

@text-color: #222;

@base-font-size: 14px;

h2

{

  font-style: italic;

  color: @text-color;

}

.big-form input[type=text]

{

  width: 300px;

  height: 25px;

  padding: 5px;

  border-radius: 5px;

  border: 1px black solid;

  background-color: #EEE;

  color: @text-color;

  margin: 2px 0;

  font-size: @base-font-size + 4;

}

.big-form label

{

  color: @text-color;

  font-size: @base-font-size;

}

Line 21 is the magical line here. Notice that we’re saying that the size of the textbox text is going to be 4 larger than the base font, whatever that ends up being. Less can infer many types of operations and data types. For example, all of these work fine:

@aColor: Blue;

@aDarkerColor: @aColor + 80%;

@font-size: 14px;

@large-font-size: @font-size + 4;

@huge-font-size: @font-size + 8px;

Next thing is nested rules. Instead of repeating the name of the class (.big-form), less can nest it so that the nested styles will include the class name like so:

/\* home-contact.less \*/

@text-color: #222;

@base-font-size: 14px;

h2

{

  font-style: italic;

  color: @text-color;

}

.big-form

{

  input[type=text]

  {

    width: 300px;

    height: 25px;

    padding: 5px;

    border-radius: 5px;

    border: 1px black solid;

    background-color: #EEE;

    color: @text-color;

    margin: 2px 0;

    font-size: @base-font-size + 4;

  }

  label

  {

    color: @text-color;

    font-size: @base-font-size;

  }

}

While this is a whole lot more readable, it generates the right CSS to do the job that is implied. The CSS will include the duplication but this will be much easier to read and maintain.

We can also support mixins. The idea of mixins is to allow function-like syntax to add one or more properties. Mixins are defined with a ‘.’ to start then a name and parameter list. A simple mixin could be:

.solid-border()

{

  border: 1px black solid;

}

This allows us to use it in a rule and everything inside it is dumped in the rule:

.big-form

{

  input[type=text]

  {

    .solid-border();

    //...

  }

  // ...

}

This results in the border line being added to the input. A great reason to use mixins is to handle cross-browser css like border-radius:

.rounded-corners-all(@size) {

  border-radius: @size;

  -webkit-border-radius: @size;

  -moz-border-radius: @size;

}

Notice that the @size variable is in the parameter list then used in the mixin. It can be used to leave size information in the rule like so:

.big-form

{

  input[type=text]

  {

    .rounded-corners-all(5px);

    .solid-border();

    //...

  }

}

This will result in all three values being put in the rule with a size of 5px. But what’s interesting is that if the value passed in is multi-values, it can still accept it as a single value for example:

.big-form

{

  input[type=text]

  {

    .rounded-corners-all(5px 10px 5px 15px);

    //...

  }

}

This works fine because the language knows that without any commas it should be treated as one value.

We’re getting close to a good solution, but we have all this information in one less file and it hurts its reuse. So we should refactor into several files moving the colors and mixins into their own files. Less has an interesting concept of import. If we move the colors and mixins to their own files, we can then use @import to pull in these other files:

/\* home-contact.less \*/

@import "colors";   // will find .less if it exists

@import "mixins";   // otherwise will drop to css

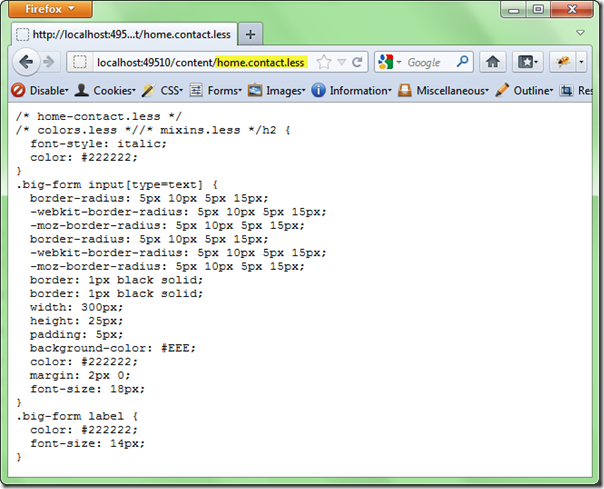
@import "site.css"; // importing css is fine too

By specifying the import without the extension, it will try and find a .less file or fallback to a .css file. This way as you refactor into .less you don’t have to go fix up names as you need less features (pun intended).

This import will bring in complete .css files as well. The import here doesn’t just refer to the imported files but actually merges them into the file for you. So the resulting .less file contains all of the imported files. This means that you can build your CSS modularly without having to worry about the number of stylesheets that the browser can handle.

**Debugging Less**

Finally, sometimes you just need to see what is happening after processing. The easiest way to see this is to just browse to the .less file:

[](http://wildermuth.com/images/results_2.png)

You can see this all working the working progress of the source code:

* [Source Code](http://wildermuth.com/downloads/modernwebdev_3.zip)

What do you think?

[**Modern Web Development - Part 4**](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)

Feb 08, 2012 at 4:50 AM

Tags: [ASP.NET MVC](http://wildermuth.com/tags/ASP.NET%20MVC) [HTML5](http://wildermuth.com/tags/HTML5) [JavaScript](http://wildermuth.com/tags/JavaScript)[CSS](http://wildermuth.com/tags/CSS)

[13 Comments](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)

by [Shawn Wildermuth](http://wildermuth.com/me)

This is the fourth of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: Debugging (This Article)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

**Debugging**

I’ve had Visual Studio open constantly on a laptop for the better part of ten years (and fifteen if you count Visual C++). I am used to setting breakpoints inside of the Visual Studio editor and pressing F5 to see what is happening.

This works in Visual Studio with JavaScript but I’ve only had a good experience doing this with Internet Explorer…and when things go wrong in other browsers, that doesn’t work effectively enough for me.

The other problem with Visual Studio debugging is that code isn’t all I want to debug. I also want to debug CSS and that is just plain nasty in Visual Studio (and don’t tell me to use Expression Web…really). So what’s a web dev to do?

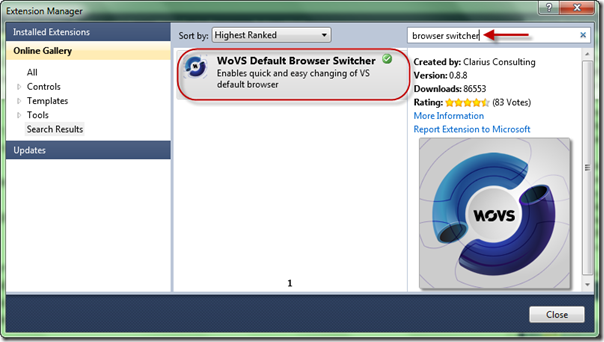
The trick is to use the browser to debug. But which browser?  All of them. The fact of the matter is that when you’re doing web development, you’re going to need to test your code with all four major browsers:

* Internet Explorer
* Firefox
* Chrome
* Safari
* Opera (ok, just kidding…)

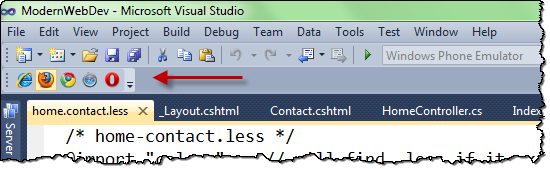
I don’t test every page as I develop on all four, but once I get testers, I need all four browsers installed for when stuff doesn’t work right on a particular browser.

**Getting Prepared for Debugging**

Before you start, I suggest you go and get the Default Browser Switcher in the Extension Manager:

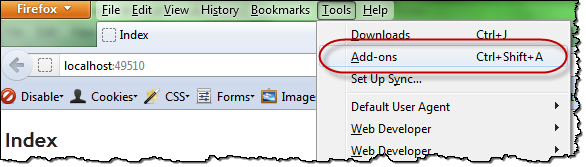
[](http://wildermuth.com/images/2-8-2012%202-19-15%20AM_2.png)

This extension will let you switch the default browser by just clicking on the toolbar:



As you can see above, I have this set to Firefox as I find that to be the easiest of the browsers to debug with. Honestly, most of the browsers are close in functionality now and some people really like the Chrome debugging experience (or even the IE one is passable), but I’ll focus on Firefox.

In Chrome, the tools are built-in but in Firefox I use the tried and true Firebug. To follow along you may need to install it. You can do this in the Add-ons option in the menu in Firefox:

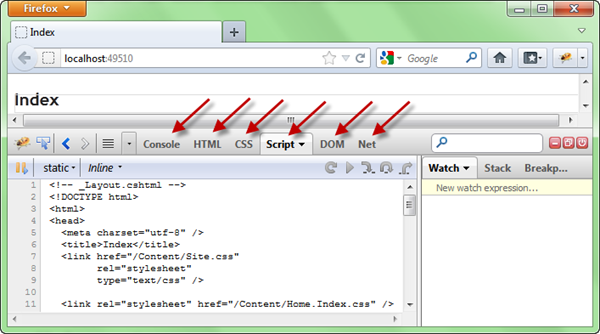


Once you have Firebug installed, it will add a button to the top right of Firefox (or hitting F12 will make it appear):

[](http://wildermuth.com/images/2-8-2012%203-04-24%20AM_2.png)

Once you open Firebug, it attaches to the bottom of the browser (though you can move it or open it in a different window). A common approach is to keep it on one monitor and your browser on another if you’re using more than one monitor. I’m not, so it usually looks like this:

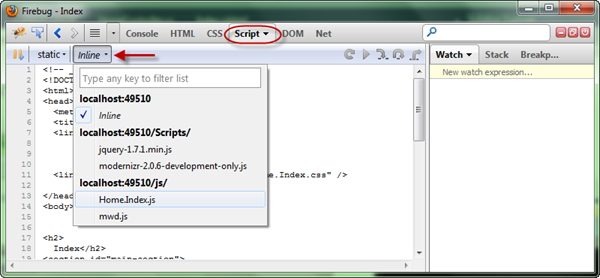
Once you open Firebug, it attaches to the bottom of the browser (though you can move it or open it in a different window). A common approach is to keep it on one monitor and your browser on another if you’re using more than one monitor. I’m not, so it usually looks like this:

[](http://wildermuth.com/images/2-8-2012%203-09-21%20AM_2.png)

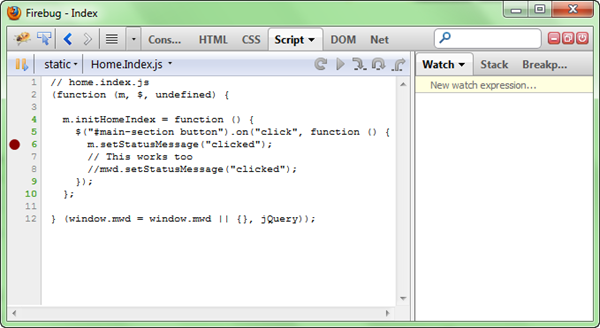
The arrows are pointing at the different tabs that will allow you to do different kinds of debugging. Let’s start with JavaScript.

**Debugging JavaScript**

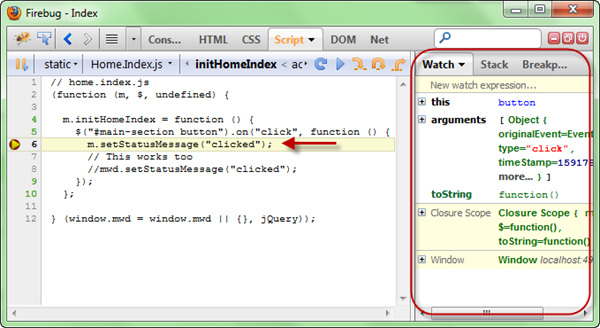
To get started, let’s begin in the Script tab. Once you select the script tag, you can use the dropdown to pick which script to open:

[](http://wildermuth.com/images/2-8-2012%203-12-32%20AM_2.png)

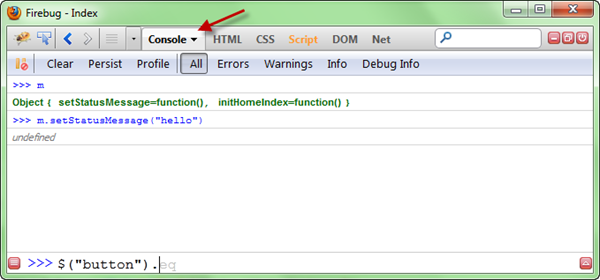
For example, I can pick the Home.Index.js file and use the mouse to click on the line inside the click handler:

[](http://wildermuth.com/images/2-8-2012%203-22-33%20AM_2.png)

This should feel like it does in Visual Studio. At that point you can just click on the button and the breakpoint will stop as expected. You can see the code stop on the right line and even see and set Watches on the right side:

[](http://wildermuth.com/images/2-8-2012%203-24-10%20AM_2.png)

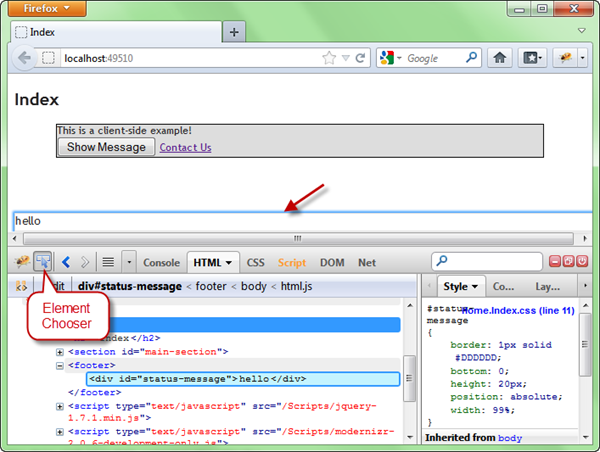
In addition to the Watch, Stack and Breakpoint tabs on the right, you can also go to the Console tab and run any JavaScript command you want (including jQuery if you like):

[](http://wildermuth.com/images/2-8-2012%203-27-47%20AM_4.png)

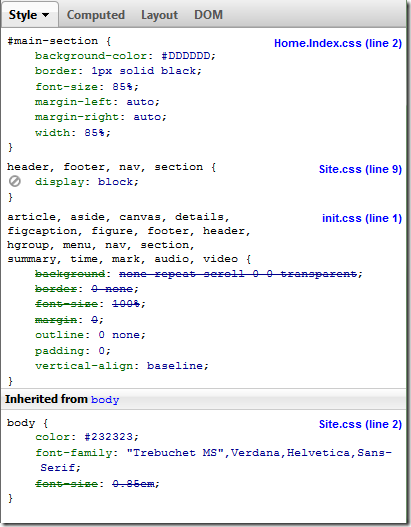
So you might be wondering, why not just do this in Visual Studio. The cool part is that if you find a problem, change it in Visual Studio and just hit F5 (to refresh the browser) and you’ll test the change. I have found that this workflow makes work really fast (for you VB6 guys, you’re all nodding your head). It’s pretty close to edit and continue ;)

**Debugging CSS**

While some people feel like CSS is a black art, using the browser can make this much easier. While in Firebug, go to the HTML tab. You can use the Element Chooser to help with this process (highlighted below):

[](http://wildermuth.com/images/2-8-2012%203-33-56%20AM_2.png)

The element chooser let’s you hover over your page and it will select the element in the HTML window for you. That will navigate to the part of the DOM that is under the cursor. This makes is relatively easy to find elements in the DOM. With the element chosen, you can see the style to the right of the window. This will show you not only the properties that the styles set for the element, but it will show you where they came from. For example, if I select the main section, I can see full style inheritance:

[](http://wildermuth.com/images/2-8-2012%204-19-39%20AM_2.png)

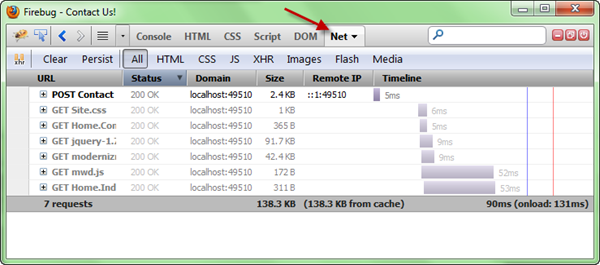
Most of the style comes from the **#main-section** rule in the Home.Index.css file, but then it uses the display property from the **section** rule in the Site.css, etc. This will show you where the properties are really coming from. When you see that some properties struck out, that means that further cascaded styles have overridden the value.

The really cool thing about this pane is that you can actually just edit the values and see them change in the browser. This means that I can manipulate the style in real-time. This is how I tweak most of my CSS. I change the CSS here and then just copy it into the CSS file when it’s what I want. You can select the values and change them by typing or using the arrow keys to change the values. In addition, you can add new values or disable values as you play with the values. You can see it work here:

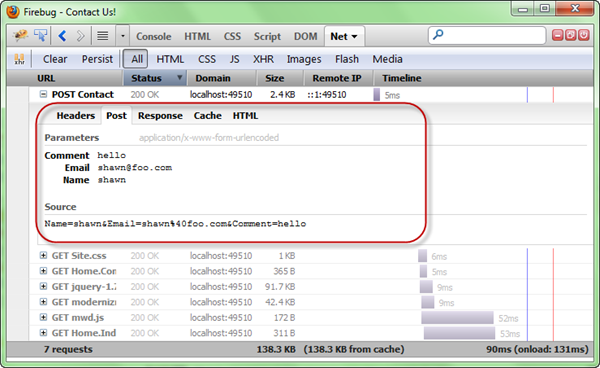
Editing CSS in Firebug (YouTube Video: <http://www.youtube.com/watch?v=esKiq6Xyt7A&feature=player_embedded>)

**Debugging Networking**

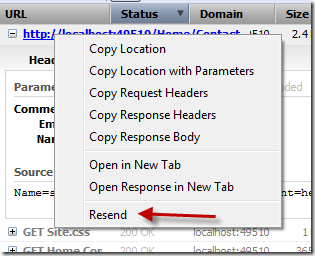
The last common use for Firebug has been to debug what networking is happening in the browser. If you click on the “Net” tab, you can see all the requests (the page, images and network/ajax calls):

[](http://wildermuth.com/images/2-8-2012%204-39-14%20AM_2.png)

For example, on the Contact page, when I submit the form, I can see the “POST” at the top of the list. Notice the + sign. When you open it you can interrogate the Headers, post, response and cache:

[](http://wildermuth.com/images/2-8-2012%204-45-47%20AM_2.png)

You can even resend the network call if you need to quickly test a scenario (useful if the request requires a lot of steps to achieve):

[](http://wildermuth.com/images/2-8-2012%204-47-06%20AM_2.png)

Remember that since this is just a browser app, you can make changes to the code, recompile and just hit resend to test your code change. This is much easier than breakpoints in Visual Studio in my opinion.

**Wrapping It Up**

I know for some of you that debugging outside of Visual Studio is going to feel weird and perhaps like cheating, but once you get comfortable with it, you’ll see that the development process is going to be faster and perhaps even more enjoyable!

What do you think?

[**Modern Web Development - Part 5**](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)

Feb 15, 2012 at 8:37 PM

Tags: [JQuery](http://wildermuth.com/tags/JQuery) [JavaScript](http://wildermuth.com/tags/JavaScript)

[8 Comments](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)

by [Shawn Wildermuth](http://wildermuth.com/me)

This is the fifth of ten parts of this blog post. The topics will be:

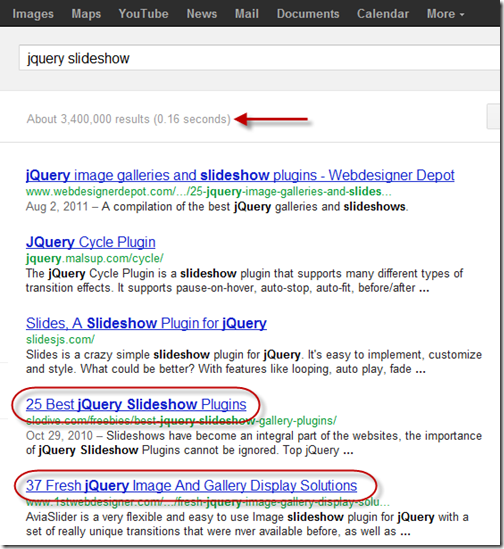
* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: Joy and Pain of jQuery Plugins (this article)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

**jQuery Plugins**

As I read on the web, the easy way to get lots of comments is to just point out the jQuery plugins I used…the blogosphere seems rife with those. But I am going to avoid that.  So what am I going to talk about? How about how to find the right plugin for the right job.

The jQuery plugin eco system is certainly impressive. There is a, seemingly, unless supply of great code to help you do all sorts of cool user interface implementations using jQuery plugins. But the strength of the ecosystem is also its weakness. How do you find the ‘right’ plugin for a particular job. That’s the difficult part of the job.

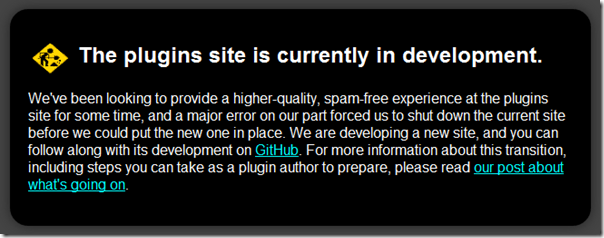
This problem is exemplified by the fact that many of the jQuery blog entries out there are like “Top 25 jQuery Plugins for Slideshows”. Wait, there are 25 plugins just for slideshows? How could they be so different as to warrant twenty-five different implementations? And, more importantly, how do you figure out which one you want:

[](http://wildermuth.com/images/2-15-2012%207-01-20%20PM_7.png)

So let’s talk about the different aspects of finding plugins.

**Finding Plugins**

It used to be that the first place I’d go is <http://plugins.jquery.com/> but (rightly so) jQuery has taken down the mess that was their plugin site. In fact the site now says:



The jQuery plugin site became a haven for spam and dead plugins so they did the right thing as get rid of it. So how do I find plugins?

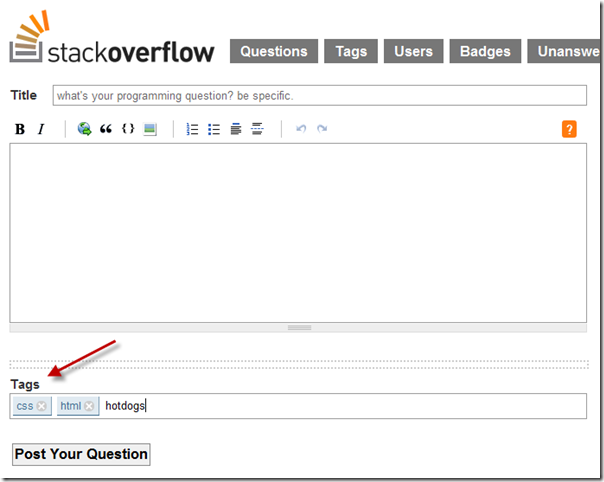
* **Web searches**: Yeah, it’s an inexact science but it works.
* **Ask around**: My Twitter feed has been a great place to get advice for all sorts of web-heads about the right plugin that people have already had success with.

I’ve tried to use GitHub’s search to find plugins but it’s just not there. In addition, I prefer individual blogs that talk about a plugin instead of the “Top 25 plugins of all times” types of posts.

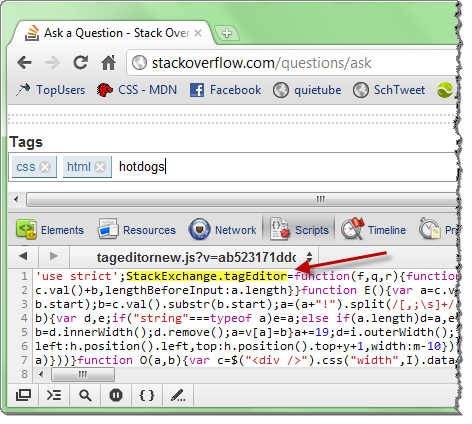
**What’s the Name of that Plugin Again?**

One thing that I find really common is that the name of a plugin isn’t unique enough. There is no jQuery plugin naming authority. Let me talk about one plugin as an example.

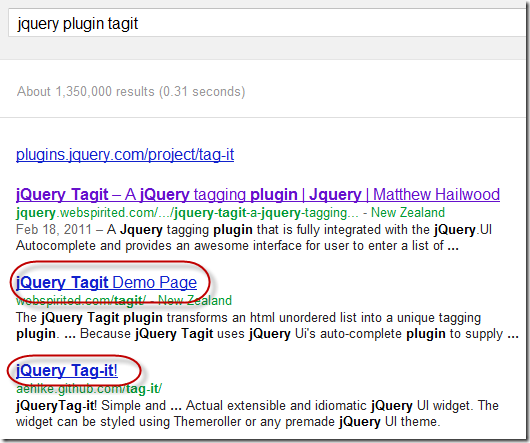
I was on StackOverflow (like most of us) and while adding a new question, I loved the way that the tag control worked:

[](http://wildermuth.com/images/2-15-2012%207-37-30%20PM_2.png)

I loved how simple and quick this was so I did a quick F12 to see how it worked and more importantly what plugin (if any) they used:

[](http://wildermuth.com/images/2-15-2012%207-40-18%20PM_2.png)

Hrmph, evidently they either wrote their own (probably) or adapted a plugin. This led me to start a web search which was too hard to find what I was looking for as the words were too generic (e.g. “jquery plugin tags”). So out to Twitter I went and I was told but a few people that TagIt was the plugin to use.  Awesome!  Searched for TagIt and came up with:

[](http://wildermuth.com/images/2-15-2012%207-59-08%20PM_2.png)

The real problem here is that these are two \*very\* different implementations and throughout using TagIt, I was pulling the wrong versions all the time. This was exacerbated because I was using the TagIt Nuget package which sent me to:

<https://github.com/aehlke/tag-it>

But what I wanted was:

<https://github.com/hailwood/jQuery-Tagit>

I can’t tell you how many times I pulled the wrong version down (thinking it was an update) and all hell broke loose. In this case that hyphen was all the difference between awesome and annoying.

**Evaluating Plugins**

So you’ve found some plugins that you like…or at least the demo page looks great. How do you know if it’s any good? I have a few criteria that I use:

*How is it hosted?*

If the plugin is commercial and hosted on the company website, has it been around and is the company seem reputable? (This is harder to determine in some cases, but if I don’t trust them with my Credit Card, I don’t’ trust their code).

If the plugin is open source, I want it to be hosted on a public repository that takes feedback (e.g. pull requests on GitHub). This means that if the person disappears, I still can get the source and the community (if any) can help improve it.

*Are there Docs?*

When I look at a plugin, I have to be able to understand what it does. Too many plugins simply say:

// It's Easy!!!!!!

$("foo").myPlugin();

Unfortunately I need more. I want there to be an online demo and a description of how you would use it in a non-standard way (as I seem to almost always need to change the defaults).

*What Does the Source Look Like?*

I ignore the visual look of a plugin but defer to the look of the code. The reasons is that many of the plugins I needed, I had to make modifications (or submit pull requests). This means the quality of the code is crucial. I should be able to see one of a number of things from the code:

* Version History
* Comments about what is happening
* Default Options

When the source is hosted in a good repository (e.g. GitHub), you can view the source before going any further. For example, here is a link to the source of the TagIt plugin:

<https://github.com/hailwood/jQuery-Tagit/blob/master/js/tagit.js>

Clean, elegant and maintainable if necessary.

**Fix it…really!**

Back in the day of bulletin boards (yeah, a long time ago) there was a sense that you could download software and not upload any of your own. These people were called leeches and there were quotas in place to make it painful for these people.

In the world of open source there aren’t any such mechanisms so I want to encourage you to help out your favorite project. If you’re using a plugin and you need a new method or option, fork the project and submit a pull request. You’re helping make the project better for everyone involved.

**What Plugins Am I Using?**

So I avoided the obvious “the best 25 plugins eva” post, but I guess it would help to see what plugins I am using for FirstInked. I used plugins as they helped me get some specific functionality working. I try to buy versus build and some of the plugins I use are commercial, but most are free. Here you go:

The Core (not necessarily actually jQuery plugins, but JavaScript tools):

* [**KnockoutUI**](http://knockoutjs.com/): Not really a plugin, but a framework for building MVVM in the browser and I didn’t feel right not including it here.
* [**PLUpload**](http://plupload.com/): An image uploader that will do batching for you.
* [**jQuery Validation**](http://bassistance.de/jquery-plugins/jquery-plugin-validation/): Doesn’t everyone use this? Included in MVC3 Default Project.
* **jQuery Unobtrusive Validation**: See jQuery Validation above ;)

Controls/Widgets:

* [**jQuery UI**](http://jqueryui.com/): I use this for dialogs and some controls like ButtonBar (et al.) Almost a must for any project really. It almost goes beyond a plugin.
* [**jQuery Checkbox**](http://widowmaker.kiev.ua/checkbox/): A simple checkbox replacement that is visually more obvious.
* [**TagIt**](http://jquery.webspirited.com/2011/02/jquery-tagit-a-jquery-tagging-plugin/): The Tagging Control

Shiny and Pretty:

* [**FancyBox**](http://fancyapps.com/fancybox/): For image popups and image slideshows.
* [**Zurb's Orbit**](http://www.zurb.com/playground/orbit-jquery-image-slider): My favorite Slider. Very customizable.
* [**Zurb's Joyride**](http://www.zurb.com/playground/jquery_joyride_feature_tour_plugin):User Feature Tours.
* [**FullScreenr**](http://nanotux.com/plugins/fullscreenr/index.html): Stretches image to be full page regardless of browser size.

**Where Are We?**

So what do you think?  Do you have your favorite plugins or different criteria for finding plugins?

[**Modern Web Development - Part 6**](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)

Feb 20, 2012 at 5:28 PM

Tags: [Web Development](http://wildermuth.com/tags/Web%20Development) [HTML5](http://wildermuth.com/tags/HTML5)[JavaScript](http://wildermuth.com/tags/JavaScript) [CSS](http://wildermuth.com/tags/CSS)

[20 Comments](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)

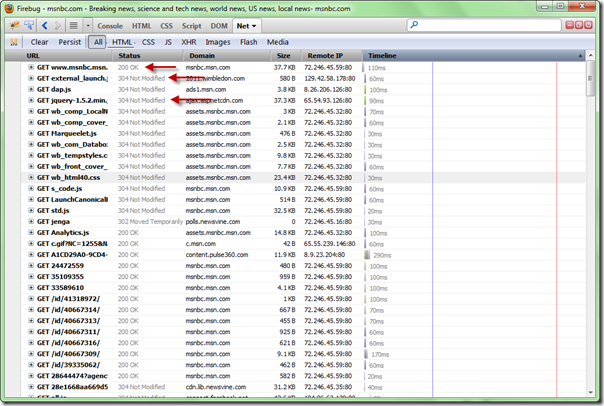
by [Shawn Wildermuth](http://wildermuth.com/me)

[[](http://wildermuth.com/images/mortarpestle_2.jpg)](http://wildermuth.com/images/mortarpestle_2.jpg)This is the sixth of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: Packaging Assets (this article)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

**The Problem**

As you develop HTML apps, one of the issues you’ll face is that your application doesn’t come to the browser in one fell-swoop. A typical web page receives content from a number of sources. Below you can see the first bun of requests from a site (in this case MSNBC.com) as shown in Firebug:

[](http://wildermuth.com/images/2-19-2012%207-44-06%20PM_2.png)

While reducing this number and size of these requests is laudable, you will also want to take browser cache into account. In the image above, you can see that some of the assets (e.g. jquery-1.5.2.min.js) returned a status of “304 Not Modified”. This status implies that the browser found the latest version of this asset in it’s cache and didn’t need to download a new one (as it hasn’t change…or was “NOT MODIFIED” from it’s current version).

For me, this meant that I wanted two things from packaging of assets:

* Reduce the size and number of downloads whenever possible
* Let the browser cache do it’s job

**What Do I Mean by Packaging?**

Part of the work involved in building a web project is taking all those assets (JavaScript, style sheets and even images) and making it efficient for your pages to use. In the case of JavaScript and style sheets, you will want to both compress (i.e. minify) as well as batch (combine multiple files). So let’s talk about them separately.

**Packaging Style Sheets**

While most solutions for packaging assets take style sheets into account, for me I decided that the dynamic style sheet languages (LESS and SASS) do this adequately. (If you haven’t read my post on using dynamic style sheet languages, see it [here](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3).)

While using the @import declaration will merge style sheets, you may also want to minify the style sheets too. In the case of **dotless** (which I am using for delivering my LESS files), you can use the configuration to turn on minimizing of the style sheets:

<dotless minifyCss="false"

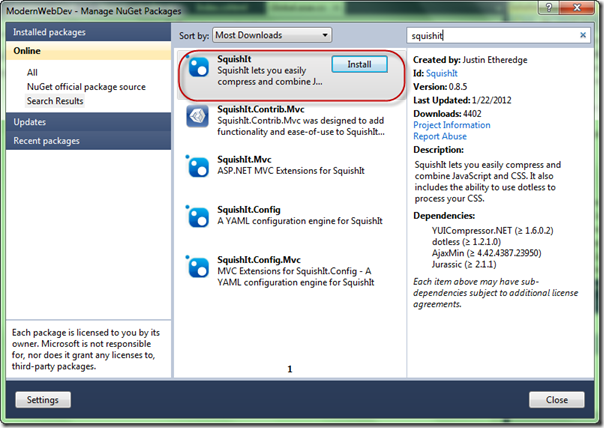
          cache="true"

          web="false" />

The **minifyCss** property can be turned on (usually in my **Web.Release.Config** file) to make the output of your LESS files to be minimized to decrease their size.

**Packaging JavaScript**

To help with the merging and minimizing of my JavaScript, I use a great library from Justin Etheredge ([@JustinEtheredge](https://twitter.com/#!/JustinEtheredge)) called [SquishIt](http://www.codethinked.com/squishit-the-friendly-aspnet-javascript-and-css-squisher" \t "_blank) (though I won’t discuss the importance of the capital letters). This library uses commonly used open source projects to do the minimizing and merging but packages it so you can call it from ASP.NET code easily. To start off with SquishIt, I would suggest just adding it to your project via Nuget:

[](http://wildermuth.com/images/2-20-2012%204-56-21%20AM_2.png)

If you read the docs for SquishIt, Justin shows it used directly in your Razor code like so:

<!-- \_Layout.cshtml -->

@using SquishIt.Framework

<!DOCTYPE html>

<html>

<head>

  ...

</head>

<body>

  @RenderBody()

</body>

</html>

@Bundle.JavaScript()

       .Add("~/scripts/jquery-1.7.1.min.js")

       .Add("~/scripts/jquery-ui-1.8.17.min.js")

       .Render("~/scripts/combined.js")

The idea of how this works is that you add individual scripts (or whole directories) into a bundle of related scripts. When Render is called, it determines whether to package up all the scripts into a single script (name combined.js in this case) or to leave them as separate scripts.

I find managing this kind of code in the markup makes it harder for me to maintain the code so I decided to do it as extension methods (of HtmlHelper):

public static class HelpHelpersExtensions

{

  public static MvcHtmlString PackageLibs(this HtmlHelper htmlHelper)

  {

    var client = Bundle.JavaScript()

     .Add("~/scripts/jquery-1.7.1.min.js")

     .Add("~/scripts/jquery-ui-1.8.17.min.js")

     .Render("~/scripts/combined.js");

    return new MvcHtmlString(client);

  }

}

To get this into my razor files, I simply just call the HtmlHelper extension method:

<!-- \_Layout.cshtml -->

<!DOCTYPE html>

<html>

<head>

...

</head>

<body>

  @RenderBody()

</body>

</html>

@Html.PackageLibs()

This works fine…except that I am packaging the minimized versions of the libraries in all cases. This will make debugging more difficult, so it would be better if I could do this differently in debug or release builds.  Easy:

public static MvcHtmlString PackageLibs(this HtmlHelper htmlHelper)

{

  var client = Bundle.JavaScript()

#if DEBUG

    .Add("~/scripts/jquery-1.7.1.js")

    .Add("~/scripts/jquery-ui-1.8.17.js")

#else

  .Add("~/scripts/jquery-1.7.1.min.js")

    .Add("~/scripts/jquery-ui-1.8.17.min.js")

#endif

    .Render("~/scripts/combined.js");

  return new MvcHtmlString(client);

}

This is better as we’re not using the minimized versions, but in release mode, I don’t want to use my local versions, but I want to rely on content delivery for very common scripts (mostly jQuery and jQuery UI). SquishIt let’s me do this by adding a CDN link (with the local link as a backup) using the AddRemote method:

public static MvcHtmlString PackageLibs(this HtmlHelper htmlHelper)

{

  var client = Bundle.JavaScript()

#if DEBUG

    .Add("~/scripts/jquery-1.7.1.js", )

    .Add("~/scripts/jquery-ui-1.8.17.js")

#else

    .AddRemote("~/scripts/jquery-1.7.1.min.js",

               "<http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js>")

    .AddRemote("~/scripts/jquery-ui-1.8.17.min.js",

               "<http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.17/jquery-ui.min.js>")

#endif

    .Render("~/scripts/combined.js");

  return new MvcHtmlString(client);

}

Pretty clean so far.  And as I use plugins, I’ll just add them here so all my plugins work. But one issue for me is that the **Render** method uses the debug flag in the web.config to determine whether it merges all the scripts into a single file. For my needs, I want the libraries to \*always\* be separate. To accomplish this the SquishIt framework allows you to call **ForceDebug**before **Render**:

public static MvcHtmlString PackageLibs(this HtmlHelper htmlHelper)

{

  var client = Bundle.JavaScript()

#if DEBUG

    .Add("~/scripts/jquery-1.7.1.js", )

    .Add("~/scripts/jquery-ui-1.8.17.js")

#else

    .AddRemote("~/scripts/jquery-1.7.1.min.js",

                "<http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js>")

    .AddRemote("~/scripts/jquery-ui-1.8.17.min.js",

                "<http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.17/jquery-ui.min.js>")

#endif

    .ForceDebug()

    .Render("~/scripts/combined.js");

  return new MvcHtmlString(client);

}

The way the works is crucial for me as I want to keep my library scripts (e.g. jQuery, jQuery UI and plugins) separate so I can gain the benefit of browser cache as much as possible. When I create a bundle of my own scripts, I go ahead and let it bundle it into a single file. In fact, for my own scripts, instead of adding the scripts one by one, I add all the scripts in the js directory:

public static MvcHtmlString PackageScripts(this HtmlHelper htmlHelper)

{

  // Get a list of the scripts

  var path = HttpContext.Current.Server.MapPath("~/js/");

  var scripts = Directory.EnumerateFiles(path, "\*.js", SearchOption.TopDirectoryOnly);

  var scriptPaths = new List<string>();

  foreach (var script in scripts)

  {

    var scriptName = Path.GetFileName(script);

    // Only add the scripts that aren't generated

    if (!scriptName.StartsWith("ModernWebDev"))

    {

      var filename = string.Concat("/js/", Path.GetFileName(script));

      scriptPaths.Add(filename);

    }

  }

  var client = Bundle.JavaScript()

    .Add(scriptPaths.ToArray())

#if DEBUG

    .ForceDebug()

#else

    .WithMinifier<MsMinifier>()

    .ForceRelease()

#endif

    .Render("~/js/ModernWebDev\_#.js");

  return new MvcHtmlString(client);

}

The first section of method is just using System.IO to search through the directory to find all the JavaScript files to include. There are a couple of new wrinkles here near the bottom of the method though.

First, the **WithMinifier** method allows you to specify which minifier to use (I’ve had better luck with the Microsoft minimizer but you can use whatever one you want). In the **Render**method I’ve also included a pound sign (#) in the path to the merged JavaScript file. What this does is tell SquishIt to generate a unique number (I believe based on a hash of the current build) so that the generated file is versioned and as new builds are created, the new version is guaranteed to download instead from browser cache to prevent accidental old versions of code.

Be aware, that using the merging (e.g. ForceRelease) of the scripts will actually generate the ModernWebDev\_#.js file in the file system. Because the code is building a list of scripts dynamically, we need to ensure that that file isn’t included in our list of scripts (as it would then become recursive and the same methods will be defined).

You may notice that since I am building all my scripts into one file, that means that all scripts will be loaded on the first and all subsequent pages. There isn’t a wrong and right here. I made the decision that loading all of it and keeping it in browser cache was easier and faster than segmenting it. But my site isn’t enormous and when you start working with very large applications you will find the need to break it up in to modules (e.g. multiple different merged scripts, probably along directories is how I’d do it).

**What about ASP.NET 4’s Web Optimization Stuff?**

As you may know, Microsoft (as of the writing of this article) has just released the Beta of ASP.NET MVC 4 and that includes a new stack for optimizing assets called**System.Web.Optimization**. There is an article that covers the basics (though it’s a little out of date) by Scott Guthrie:

<http://shawnw.me/xuPcoo>

I considered moving to this as it’s pretty slick and pluggable, but it was missing some key features for me:

* Packaging of groups of assets but not including them separately (the Libraries packaging is an example of this).
* No support for CDN delivery of libraries.
* Wasn’t a clear way to not minify in debug versus release mode or provide different versions for those modes.

Because of these issues, I am sticking with SquishIt (and to follow the adage that says “old code is good code”). I am sure it will improve (either by people writing plugins to it or MS fixing it) to address these issues.  For many projects that don’t care about some of the control I needed, it is a great solution.

**In Action**

You can see it work in action with the latest version of this example:

<http://wildermuth.com/downloads/modernwebdev_6.zip>

## [Modern Web Development - Part 7](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)

Mar 06, 2012 at 12:50 AM

Tags: [Development](http://wildermuth.com/tags/Development)

[12 Comments](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)

by [Shawn Wildermuth](http://wildermuth.com/me)

[[](http://wildermuth.com/images/breakinsafe_2.jpg)](http://wildermuth.com/images/breakinsafe_2.jpg)This is the seventh of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: Distributed Version Control (this article)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

Before I wax poetically about why to use distributed source control, let me talk about what it is (and why it is different).

Back in the very old days (did I mention I am old?) I would keep my source on a floppy disk and put in a safe every night (no, not 9 track tapes like some of you are thinking..I am not \*that\* old). This was a way to secure the source in case of disaster…but all it did was keep the source secure. Source control was more than that. Later as I used a myriad of source control vendors (yes, including the dreaded Source Safe), they all seemed to have some common features:

* Centralized Storage of Assets
* Management of History
* Handled Consistency of Assets

While I know some companies still don’t use source control of any kind, most do. For many companies they use source control so that they have control of the source. They know where it is and they can back it up and not rely on developers to secure their assets.

In many ways, the key here was that that the source control systems were really version control systems and in fact that is what they became to be called about 10 years into my career. This clarified a real fact, that these were repositories that kept versions of files, not just source code.

The centralization that was the hallmark of those version control systems (VCS) came at a price. The process of pushing changes to the VCS tended to be slow and either relied on locks (check-out, check-in) or handled merging of changes. Since this was a tedious task, developers tended to make big, irregular check-ins as it interrupted their workflow. This is how it was for a long time.

About a decade ago, it became apparent that these VCS systems were also problematic as development teams became more virtual. These VCS systems didn’t handle offsite, or remote situations well…and if they did, they were horrifically slow. For onsite development teams over Ethernet they were fine…but for anything like a distributed team…awful.

### Distributed Version Control Systems (DVCS)

Distributed Version Control (or Distributed Source Control if you prefer) is simply a way that turns version control on it’s head. Instead of a single master repository for the source, everyone gets their own repository. This means you can check-in often and really quickly. This encourages developers to make check-ins very granular which makes merging changes from different groups simplier. What happens for the developer is:

* Makes a change.
* Checks in the change to their local repository.
* Eventually bundles a number of these changes to sync with the remote (or master) repository.

Even with small teams, this speeds up development and is just simply cleaner.

### But Why?

You may be wondering: “Why is version control part of the Modern Web Development story? Source Safe is working just fine for us.” Ok, I hope that isn’t what you’re thinking..at least the last part.

In building FirstInked, I had to get comfortable with a lot of different ideas. I had used distributed source control before then, but it was purely for open source projects. I had used it to send patches, upload my source to CodePlex (et al). But to use it on a real project with multiple developers, my head was still stuck in the old way.

I knew that I was going to be working with people from wherever was the right place for those people to be. At a coffee shop, in their home town on another continent, wherever. I also knew that I didn’t want to be stuck not being able to do check-ins and keep the granular nature of work even if they (\*gasp\*) didn’t have an Internet connection. DVCS was what I wanted to be able to get the job done.

In addition, I knew I wanted to be able to do continuous integration so that every sync to the master repository should kick off a build (I’ll talk about how we did that in an upcoming part of this series).

### How

There are a handful of DVCS’s out there including GIT, Mercurial (or HG for short) and others. For me I did not want to be the IT department. I wanted to get a company that would simply host our code…privately. They’d do the backups and all would be well. After, very summarily, looking at GIT and Mercurial I took the easy way out and chose Mercurial. My reason?  Tooling.

GIT is a great tool (as I am sure many of you will tell me in comments ;) and is mature enough for prime time. It has broad support and a great eco system. But I chose Mercurial because since we were using Visual Studio the tool story for GIT at the time (and mostly the same now) is immature. With GIT there is an expectation that most users will use a BASH shell to submit their changes. The last thing I wanted to do was to was to learn something new. I wanted a tool that would just slip in where I needed it.

So I decided on Mercurial because of the two integration layers I use:

* [VisualHG](http://visualhg.codeplex.com/)
* [TortoiseHG](http://tortoisehg.bitbucket.org/)

[VisualHG](http://visualhg.codeplex.com/) is a plugin into Visual Studio that handles add/change/delete/renaming of files in a solution for me. For non-Visual Studio files, I also use [TortoiseHG](http://tortoisehg.bitbucket.org/" \t "_blank) to allow me to add/remove files from the file system ([TortoiseHG](http://tortoisehg.bitbucket.org/" \t "_blank) works as an extension to Explorer).

### Hosting

While I could have handled the hosting of Mercurial myself, as I said earlier, I wanted someone to handle that for me. I ended upon [BitBucket.org](https://bitbucket.org/) as they changed based on users, not on the #/repositories. I knew that I would probably need a handful of private repositories to fulfill all my projects (WebSite, Mobile Apps, etc.). In fact, [BitBucket.org](https://bitbucket.org/) allows up to 5 users with unlimited public \*and private\* repositories.  That means it’s perfect (AFAIAC) for startups.

I don’t use [BitBucket.org](https://bitbucket.org/" \t "_blank)’s tooling for bug/feature handling or other features. It is simply DVCS hosting for me.

(For the record, I use [AgileZen](http://www.agilezen.com/" \t "_blank) for project management (kanban board) and[FogBugz](http://www.fogcreek.com/fogbugz/) for bug tracking).

What do you think?

## [Modern Web Development - Part 8](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)

Mar 24, 2012 at 4:47 PM

Tags: [ASP.NET MVC](http://wildermuth.com/tags/ASP.NET%20MVC) [JavaScript](http://wildermuth.com/tags/JavaScript)

[3 Comments](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)

by [Shawn Wildermuth](http://wildermuth.com/me)

This is the eighth of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: Working with Facebook (this article)
* 9: [Mobile Pages](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)
* 10: Deploying to the Cloud (upcoming)

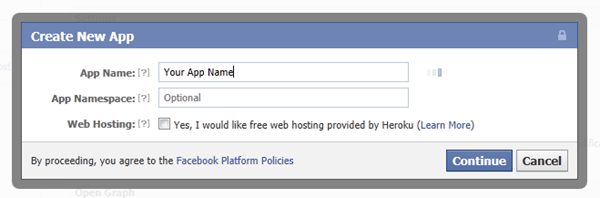
Oh Facebook…how do you becoming so insistent on integrating you into every website? Well anyway, let’s show you how it actually works. In this post, I’ll show you how to authenticate an app using Facebook.

### Getting Started

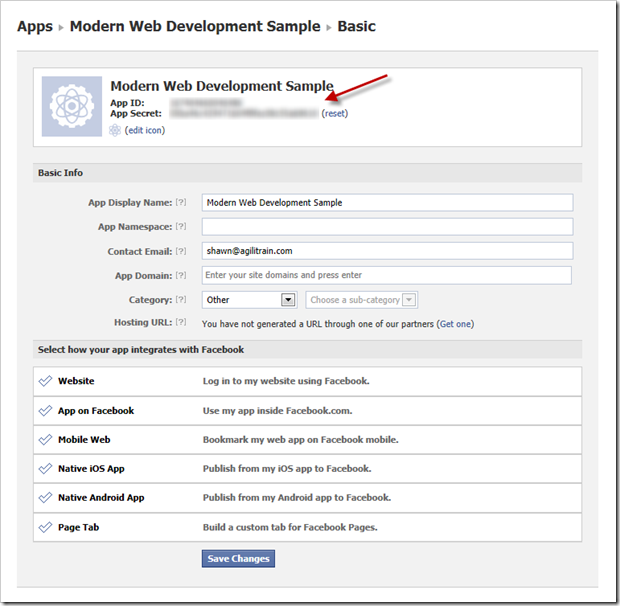
When you want to integrate with Facebook, you’ll need the Facebook SDK. Unlike other JavaScript APIs, the Facebook API isn’t a download. The API has some specific peculiar patterns that it requires. But if you obey Facebook, it will (usually) bend to your will. To get started you’ll want to visit the Facebook developer site:

<https://developers.facebook.com>

Before you get started using the Facebook API, you need to have a Facebook application. This is simply registering with Facebook to get a unique key for your use of the API. To do this, you will want to click on the Apps button (or go to <https://developers.facebook.com/apps>) and choose “New App”:



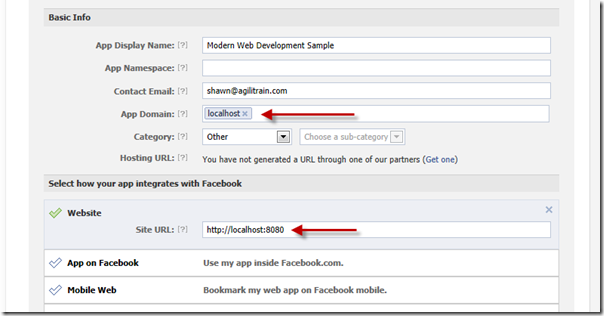
This will create a new application and take you to your application page:

[](http://wildermuth.com/images/3-24-2012%202-30-28%20AM_2.png)

The App ID and App Secret are the magic numbers you’ll need (as shown by the arrows). The only other part of this process that is important is the “App Domain” specified here. The Facebook API only works (with the App ID and App Secret) from a specific domain and website address. For development I suggest you set this to your localhost address (yes, this works fine). For example:

[**http://localhost:8080**](http://localhost:8080)

On the App page this will look like:

[](http://wildermuth.com/images/3-24-2012%203-38-49%20PM_2.png)

This works for a new project, but once you deploy to a live site you no longer can test your development application as you’ll need to change this to the live site’s domain. To address this, I suggest you create two applications – a development application and the application for your deployed application.

NOTE: If you are using Internet Explorer for development, the Facebook SDK doesn’t work on non-standard ports. You should either pick port 80 (or 443) or use another browser for your Facebook work.

So let’s start loading the SDK.

### Loading the SDK

Now that you have your application, you can start using the SDK. The Facebook SDK requires you download the .js file, have an element in your HTML that the SDK uses as well as have an initialization method. So first you’ll need an element (usually a DIV) named “fb-root”:

<!-- The required element that the Facebook SDK uses -->

<div id="fb-root"></div>

It doesn’t matter where the element is in your design, I usually just place it right above my script tags at the bottom of the <body/> tag. It will not be visible to your users.

The Facebook SDK exposes all of it’s functionality (pun intended) on global object called “FB”. Before you can use the SDK you will need to initialize this object to include information like your App ID (and other options). This is usually called in a function that is called by the SDK when it is loaded on your page:

<script type="text/javascript">

  window.fbAsyncInit = function () {

    FB.init({

      appId: '@ConfigurationManager.AppSettings["facebookAppId"]',

      cookie: true,

      status: true,

      frictionless: true,

      xfbml: true

    });

  };

</script>

This function must be added to the ‘window’ object and be called ‘fbAsyncInit’. In the method you should call the init method of the FB object to specify several pieces of information. Note that I am calling this in a Razor page so I can using the @ syntax to load the facebookAppId from my AppSettings in my web.config file. I do this so I can have a debug application and switch it to a release AppId when I deploy it to my live servers. The other options are covered in the Facebook SDK so I won’t repeat that here.

Finally you need to load the Facebook SDK. The SDK is located at “connect.facebook.net/en\_US/all.js”. But because of all the initialization that it performs, it is best not to just use a script tag. Instead they suggest you load it asynchronously like so:

// Load the SDK Asynchronously

(function (d) {

  var js, id = 'facebook-jssdk'; if (d.getElementById(id)) { return; }

  js = d.createElement('script'); js.id = id; js.async = true;

  js.src = "//connect.facebook.net/en\_US/all.js";

  d.getElementsByTagName('head')[0].appendChild(js);

} (document));

This creates a new **script** element after the pages has loaded and loads the script. Notice the **src** of the **script** element: “//connect.facebook.net/en\_US/all.js”. This URI is used so it will get it via https or http based on how your application is hosted. The workflow here is:

* Load the SDK asynchronously
* Execute the fbAsyncInit method if it exists on the window object
* FB.Init is called with the AppID (et al.)

At this point you’ve done all the work to be able to use the SDK.

### Logging into Facebook

For this example, I am going to keep it simple and just show login via clicking on an image and then showing the name of the user that was logged in. So I’ve added a new image in my HTML page that I’ve specified a class called ‘fb-login’:

<img class="fb-login" src="/images/FBLogin.png" alt="Facebook Connect" />

This could be an actual button or even a link, but for my needs I just need some UI element for the users to click. Then I can just add a click handler to the image like so:

// Handle anything that can be clicked as

$('.fb-login').click(function () {

  FB.login(function (response) {

    if (response.authResponse) {

      mwd.showCurrentUser();

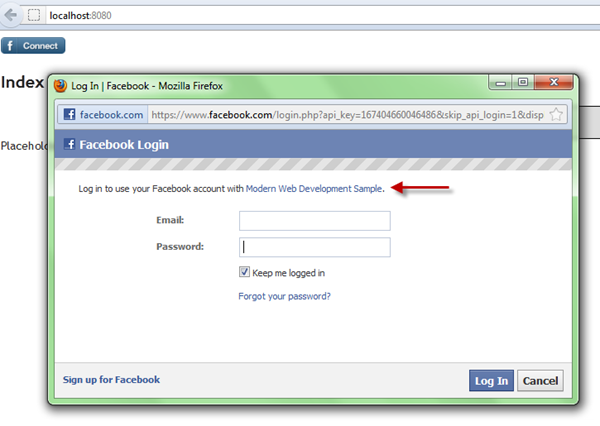
    }

  }, { scope: 'email,user\_about\_me' }

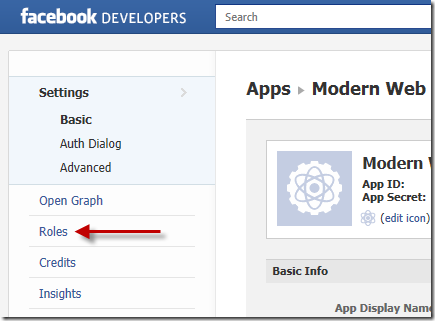
});

In this case, I am just calling the Facebook SDK’s **login** function to perform the login. The second parameter of the **login** function is an object that can be used to specify the ‘scope’. This ‘scope’ is the list of permissions. In this case, we’re just asking for permission to see the email address and the basic user information for the person. You could ask for more permissions (or more appropriately you can expand these permissions as you need them).

When the login function is called it will look for the AppID that was specified and go to Facebook.com and popup a new login dialog for you:



This dialog is shown if the user is not already logged in. If the user hasn’t used Facebook to login before it will also show a permissions dialog. If you login with \*your\* login that you created the App with, this dialog won’t appear as you’ve already approved the application since you’re the owner.  But if you try and login with other users it will simply fail.  Why? Because your application isn’t public yet. So either you can make it public or usually you’ll just add testers to your application instead (for early development). To add testers to your application, you can visit the application page again and pick “Roles”:



On the Roles page, you can add Developers and Testers. These must be Facebook users and they will need to approve their role before they can login. Once you do this, you will get the permissions dialog (based on the permissions you asked for in the **login** method):



This dialog will be shown only when you ask for more permissions than they’ve approved before. For example if you ask for email and user info now, but later expand that to looking at their pictures, this dialog will show asking approval again. In this way you can expand the permissions as a user uses your site without scaring them off with lots of permissions later.

The login function’s first parameter is a callback function that supplies a response of the login. If the response is true, login was successful…otherwise it fails. If you look back at my login callback you’ll see I am calling a method called **showCurrentUser** that I created. This is where we can actually use the Facebook SDK to get some data.

### Using the SDK

You’re logged in, so now you may want to get information from Facebook. The Facebook SDK supports a method called **api** that is used to get objects from the Facebook Social Graph. For this simple example, we’ll just get the users’ object so we can get their full name. The Facebook SDK has a special object identifier called “me” that represents the logged in user. SO we can just use the “me” URI to get the user object:

// mwd.js

(function (m, $) {

  m.showCurrentUser = function () {

    FB.api("me", function (result) {

      if (result.error) {

        alert("Failed to read 'me' from facebook");

      } else {

        alert(result.name);

      }

    });

  };

} (window.mwd = window.mwd || {}, jQuery));

The FB.api call requests information from the social graph and returns an object. In this case I am just testing the result.error to see if it worked.  If it did, I am just showing the name of the user. You can request more objects using a social graph ID. For example if you had permission to the users pictures you could request FB.api(“me/albums”, ...) and it would return a list of the albums for the users. From which you could request the pictures in each album. I find the API very simple to use (once you have all the basics setup).

You can get a copy of the code with my sample facebook application (though you won’t be able to login into the test project as I won’t make it public). You can change it to your own Facebook Application API in the web.config file and go to town!

[ModernWebDev\_8.zip](http://wildermuth.com/downloads/ModernWebDev_8.zip)

Questions?

## [Modern Web Development - Part 9](http://wildermuth.com/2012/4/29/Modern_Web_Development_-_Part_9)

Apr 29, 2012 at 8:30 PM

Tags: [Web Development](http://wildermuth.com/tags/Web%20Development) [Mobile](http://wildermuth.com/tags/Mobile) [CSS](http://wildermuth.com/tags/CSS)

No Comments

by [Shawn Wildermuth](http://wildermuth.com/me)

This is the ninth of ten parts of this blog post. The topics will be:

* 1: [A New World](http://wildermuth.com/2012/1/18/Modern_Web_Development_-_Part_1)
* 2: [Architecting JavaScript](http://wildermuth.com/2012/1/20/Modern_Web_Development_-_Part_2)
* 3: [A Better CSS](http://wildermuth.com/2012/1/31/Modern_Web_Development_-_Part_3)
* 4: [Debugging](http://wildermuth.com/2012/2/8/Modern_Web_Development_-_Part_4)
* 5: [Joy and Pain of jQuery Plugins](http://wildermuth.com/2012/2/15/Modern_Web_Development_-_Part_5)
* 6: [Packaging Assets](http://wildermuth.com/2012/2/20/Modern_Web_Development_-_Part_6)
* 7: [Distributed Version Control](http://wildermuth.com/2012/3/6/Modern_Web_Development_-_Part_7)
* 8: [Working with Facebook](http://wildermuth.com/2012/3/24/Modern_Web_Development_-_Part_8)
* 9: Mobile Pages (this post)
* 10: Deploying to the Cloud (upcoming)

Why do users complicate our lives by trying to view our content on their phone and tablets?  It's even an issue for smaller sized desktop screens too. Since this is simply a reality for today's web developer I was glad to see there were real strides in working with mobile.

### Ways to Handle the Mobile Web

In general there are three techniques for dealing with mobile device browsers:

* **Mobile Site**: Write a version of your site specifically for mobile browsers/phones.
* **Mobile Pages**: Write mobile specific pages for individual pages in your site.
* **Responsive Design**: Adapt pages to work with mobile devices.

Which should you choose? Well, that depends...

While building a mobile site is been a standard way of doing business for a while (ahem m.foo.com is the pattern, right?), it involves more work and occasionally separate engineering efforts. This is still an option, but I think it should become more uncommon.

Building mobile pages is closer to what an software guy might suggest, but that can be a lot of engineering effort too. Imagine building every page twice (or more). But you don't have to. The last technique is to use "Responsive Design". Let's dig into how that works, then we'll add Mobile Pages to fill in the gaps (this is my take on the right approach).

### Responsive Design

The basic tact of responsive design is to design your pages (and site) to react to changes in browser size. This means on different devices, you can have your site \*respond\* to the change. Instead of complex logic to sniff the type of device and a database screen sizes, you can use CSS specify changes to your design based on screen size.

A CSS media query is the key to getting this to work. A media query device type (in our case, screen) and size help you create exceptional case CSS for specific device types. A typical example in responsive design is defining that you want a 'screen' and a specific range of widths:

@media only screen and (min-width: 960px) {

}

This allows you to have rules that apply to only certain widths of an app. This works well to allow you to change formatting for desktop, then tablet, then phone.

/\* fallback \*/

#main

{

  width:  989px;

}

/\* Tablet Portrait size to standard 960 (devices and browsers) \*/

@media only screen and (min-width: 768px) and (max-width: 959px)

{

  #main

  {

    width:  760px;

  }

}

/\* Mobile Landscape Size to Tablet Portrait (devices and browsers) \*/

@media only screen and (min-width: 480px) and (max-width: 767px)

{

  #main

  {

    width:  472px;

  }

}

/\* Mobile Portrait Size to Mobile Landscape Size (devices and browsers) \*/

@media only screen and (max-width: 479px) {

  #main

  {

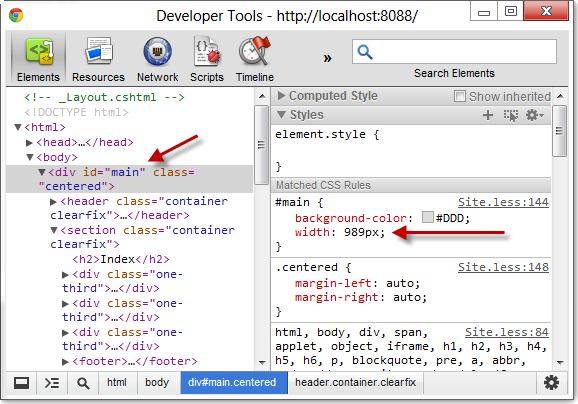
    width:  400px;

  }

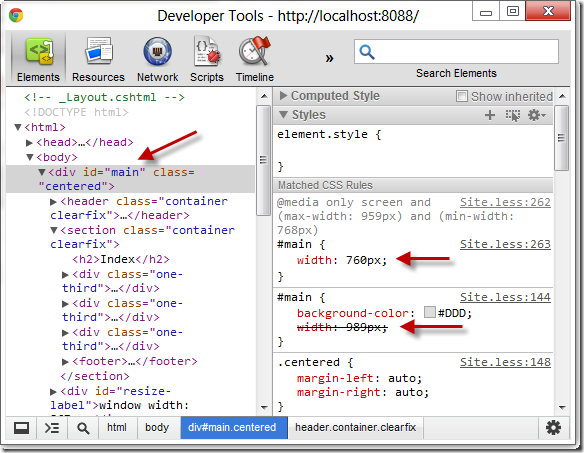
}

The idea behind this is to define rules that apply everywhere (and most of your rules will not be inside of media rules) and then for specific screen sizes, create rules that override the default rules. For example, the 1024px width for **#main**works in the browser, but when you resize your window to be < 960px(or open it on a device with the right resolution), the size of **#main** changes to 755px. This is to ensure it fix on smaller resolution devices. This continues for the different size screens all the way down.

Note that in the browser tools, you won't see the media rules, they'll just be applied. So on my laptop, when I look at the matching rules for **#main** I get:



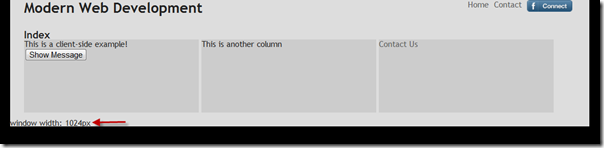
Because the page size is > 960px at this time, I only see the first #main definition that defines the width. But if I reduce the size of the window to be smaller than 960px, the rules change as the media rule kicks in:

[](http://wildermuth.com/images/4-29-2012%207-24-22%20PM_2.png)

In this case we see both the original #main rule and then the #main added in the media query section and since the media query happened after the first declaration, it takes precedence.

If there are elements of the page (e.g. ".centered") that aren't different for the screen size variations, I don't need to duplicate them since their in the main CSS scope.

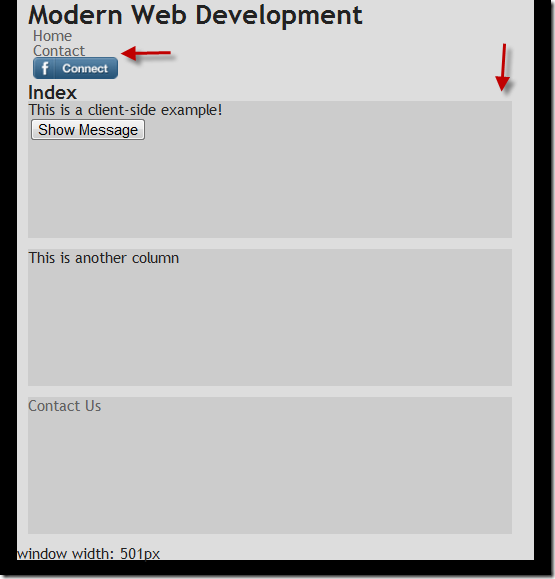
For example, I've created a simple design with a container and a way to float boxes next to each other when the screen is wide:

[](http://wildermuth.com/images/4-29-2012%207-32-56%20PM_2.png)

When the screen is large (e.g. 1024px wide as shown), it looks great. The menu items are floated to the right top, and the three columns are just floated sections. When I go down below 960px I simply change the sizes of the columns:

[](http://wildermuth.com/images/4-29-2012%207-33-27%20PM_2.png)

But you can completely change the rules to change the look, like when we're below 768px:

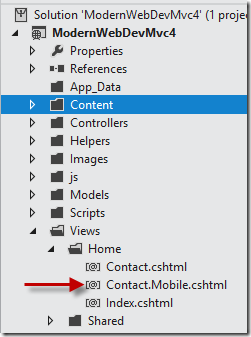


In this case, I am keeping the columns to be farily wide, but no longer floating them as on a small screen (like a phone) floating makes them too small to be useful. In addition, you can change other elements (like the menu is stacked below the logo instead of floated to the right. You can make design decisions about what looks best on the right size device.

In all of these cases, these changes are made in the design artifacts (e.g. CSS) and not in code. So that you still have a single code block for a particular page, even very complex ones. But sometimes even that isn't enough.

### Mobile Pages

There are times when you want a completely different experience for mobile users than typical web users. This may not be a solution for the entirety of a site, but only applicable on special pages (e.g. data entry screens). For those times when you need special mobile pages, ASP.NET MVC4 has a great way to handle this (though there are hacks to work with MVC3 and get this behavior). The trick is to simply create a new view and name it {page}.mobile.??html (e.g. Contact.Mobile.cshtml). By default, it will find the mobile page (if it exists) for mobile devices. For example:



Why a specific page for mobile? One big reason is to use mobile frameworks (e.g. KendoUI Mobile or jQuery Mobile). My non-mobile version of my contact page looks like you might expect (using a \_layout page):

@\* Contact.cshmtl \*@

@model ModernWebDev.Models.SomeFormModel

@{

  ViewBag.Title = "Contact Us!";

}

@section Stylesheets

{

  <link rel="stylesheet" href="~/Content/Home.Contact.less" />

}

<h2>

  Contact</h2>

<section class="big-form">

  @using (Html.BeginForm())

  {

    @Html.ValidationSummary()

    @Html.EditorFor(m => m)

    <input type="submit" value="Send" />

  }

</section>

But for the mobile page, I could use jQuery Mobile to show off the page:

@\* Contact.mobile.cshmtl \*@

@model ModernWebDev.Models.SomeFormModel

@{

  Layout = null;

  ViewBag.Title = "Contact Us!";

}

<!DOCTYPE html>

<html>

<head>

  <title>Modern Web Dev - Contact</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="stylesheet" href="~/Content/jquery.mobile-1.1.0.min.css" />

</head>

<body>

  <div data-role="page">

    <div data-role="header">

      <h1>Modern Web Dev</h1>

    </div>

    <!-- /header -->

    <div data-role="content">

      <h2>Contact (Mobile Version)</h2>

      <section class="big-form">

        @using (Html.BeginForm())

        {

          @Html.ValidationSummary()

          @Html.EditorFor(m => m)

          <input type="submit" value="Send" />

        }

      </section>

    </div>

    <!-- /content -->

  </div>

  <!-- /page -->

  <script src="~/Scripts/jquery-1.7.2.min.js"></script>

  <script src="~/Scripts/jquery.mobile-1.1.0.min.js"></script>

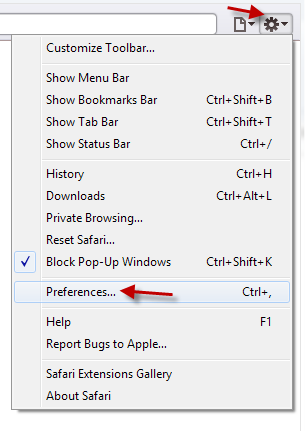
</body>

</html>

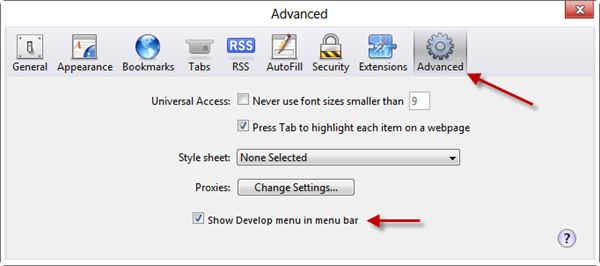
Note that I am setting the Layout to null to prevent using the Layout page as my 'master page' but doing it all in this page. But since this page is relying on sniffing the browser to detect a site, how do I trigger it? Magic? Nope, Safari!

The trick is to use Safari as it has the ability to send iPhone, iPad and iPod request headers. How does it work?  Let me show you.

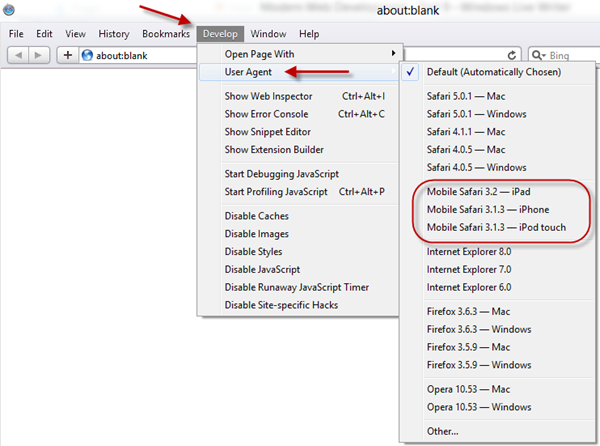
If you have Safari installed, you'll need to enable the developer menus by opening preferences:



Then on the Advanced Tab, show the Developer menu:

[](http://wildermuth.com/images/4-29-2012%208-14-08%20PM_2.png)

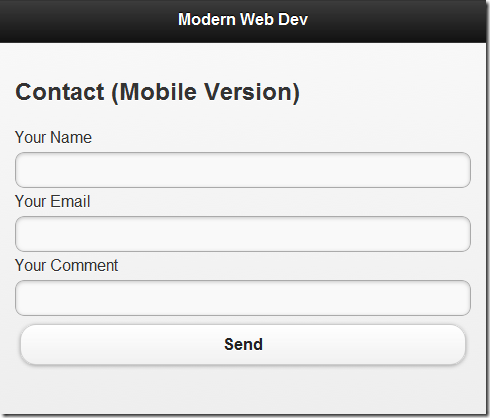
Once you have that you can enable them in the User Agent menu:

[](http://wildermuth.com/images/4-29-2012%208-14-30%20PM_2.png)

This will trigger the mobile versions.  The contact page with the default User Agent:



And the jQuery Mobile version once I change the User Agent to "iPHone":



Using Mobile Pages allows you to have specific parts of your site use these mobile frameworks but not requiring whole separate sites.

Combining the two efforts (Responsive Design and Mobile Pages) I think is the best of both worlds. I don't see a realistic place for creating separate sites.

Here's an example of using these two techniques together:

* <http://wildermuth.com/downloads/modernwebdev_9.zip>

What do you think?