The Joys of Developing for Internet Explorer

Posted At : February 9, 2012 10:51 AM | Posted By : Cutter
Related Categories: jqGrid, Sencha, Ajax, Development, JQuery, Usability, ExtJS

Note: Follow the madness here to it's conclusion, to discover yet another "Really? I didn't know that..." IE moment, that may save you heartache and pain.

Is the sarcasm evident in my title? It should be. While Internet Explorer may have been "groundbreaking" when it was released, it has ever been the bain of the web developer's existance. Why, you ask? Because it refuses to adhere to standards, and just operates differently than everyone else. To be fair, Internet Explorer 10 is in the works, and supposedly closes the gap a fair amount (and even wins out in some of the html 5/css 3 support), and 7, 8 and 9 did progressively improve (if slowly). Unfortunately the fact remains that some companies/organizations/governments are stuck on Internet Explorer 6, or maybe even 7. If you are a web developer, and write cross-browser web applications, chances are Internet Explorer has hit you more than once in your career.

It's the dumbest things that hit you too. The most obscure "WTF!?!" moments that drive you crazy. That is a daily experience for me now.

I've been a server-side developer for over two decades now, but in the last several years I have dealt more and more with client-side development. Why? Well, I started with HTML, and got into Javascript when it was introduced, and then got into server-side programming, so I already had some roots. With all of the work that I've done, working with **Ext JS** and **JQueryUI**, my current position threw me to the wolves to work on standardizing our application's interface development. Fun, right? It is (a lot), but the client is one of those mixed environments that is entrenched in IE, typically at 7 or 8. They can't upgrade, so it's our responsibility to make sure that our interfaces can render and function properly in these older, buggy, non-compliant POS browsers. (Note the frustration here.) Let me layout one of those typical, what-the-hell-is-goin-on-here, Internet Explorer moments for you.

The Application

Like most good development companies, we've made the shift to more modular, reusable code. It's a legacy codebase, serving the client well for years, and goes through constant review, revision, update and improvement. Many of us (developers in general) would love to just scrap our current codebase and start from scratch, but that's not really practical in most situations. What happens is you begin to slowly move away from **The Big Ball of Mud** design pattern, by replacing small pieces of your application as updates/changes are required, with more modern code. Rather than one huge application, what you end up with are small, mini-applications, that can

be plugged in anywhere within an application. Each of these 'applications' then have their own supporting files: html, javascript, and css. Welcome to the fun.

Collision Control

At this point, the issue isn't necessarily browser specific, but rather logistics. When you start adding in multiple mini-applications, things can (and will) butt heads. You will have issues with race conditions, as multiple scripts use variables of the same names, have functions with the same name that override each other due to execution order, id's repeated in a page (IE hates this a lot), and multiple event handlers accidentally added to the same DOM controls because selectors aren't specific enough. You tighten up your DOM selectors, come up with some variable and DOM naming standards, and learn about namespacing. Each a slow, progressive step towards sanity.

Then there's the issue of accidentally loading the same files multiple times. Our application uses JQueryUI, **cfUniform**, **TinyMCE**, and **jqGrid** extensively. Each of these mini-apps loads it's own support files. Suddenly, you're faced with coming up with systems of control, to ensure that you aren't loading scripts and CSS multiple times.

Back to the Browser

There's a ton of debate out there about loading multiple scripts and css files. Loading multiple files has performance issues for initial download, but maintenance is far easier with these bit modularized like your display and model. In many applications things aren't complex enough for multiple files to truly be an issue. But, when you are writing a complex application, like scheduling software, customer resource management, content management, etc., the number of 'modules' you might include in a page can grow and grow and grow, especially if your interface makes heavy use of ajax. This is where Internet Explorer can actually force your hand into creating a better experience, unintentionally and generally when it's inconvenient.

What happens when you load up your app, and notice that your forms (cfUniform) don't appear to be styled, and your highly configured and customized WYSIWYG editor (TinyMCE) appears to be half loaded? But, only in IE? Well, you start troubleshooting. Maybe cfUniform didn't load? No, the stylesheet is what's important here, and Developer Tools say it's there. The editor? Did it initialize? Well, the style and font dropdown info is there, even if they aren't in dropdowns. What's going on?

You start removing script files, one at a time, to make sure there isn't a conflict. But wait, it works in everything but IE? Well, you try it anyway. You step debug things,

trying to see a break. And you Google (and Google, and Google). These are my days. And, after three days of this single issue, you finally stumble on **an obscure post** (page 15 of a Google search). Please, tell me you're kidding me...

Yes, Internet Explorer (at least 6 through 9) only apply up to 31 stylesheets and/or style blocks. So, when your page (1 specific stylesheet) uses JQueryUI (1 plugin stylesheet), and a menu (3 plugin stylesheets), and a nav widget (1 plugin stylesheet), plus cfUniform (3 plugin + 2 override stylesheets), and jqGrid (1 plugin stylesheet), and TinyMCE (multiple dynamically loaded stylesheets, depending on plugin configuration), and...

You get the picture. Oh, and did I mention that some of those stylesheets were 10 line, IE only stylesheets to hack IE's different handling of CSS? Yeah, salt in the wound. To test this, I went looking for something that might be loaded by default, but wasn't needed on this page of my app (or at least, not for what I was testing). I found a few stylesheets I could disable in local development, and reloaded my page. Imagine my surprise when everything rendered as it should. I had to toggle it back and forth a few times just to verify. I think I spewed profanity for several minutes.

What To Do?

OK, so we had already been looking to form a strategy for combining files and minification. Notice I said "looking to". That is to say, we wanted one, and knew we needed one, and even done a little research, but we hadn't **solidified** one. If we had, we'd already have been working on it. Now, under the gun, with deadlines looming, we have to put it in gear. What to do? Well, in this case, improvise. Time not being on our side, we have to do this manually. Identify multiple CSS files that are used either constantly (every single page load) or extensively (85%+ page view). Combine and compress, and remove single *link* references from the code. Now, I didn't do this completely manually. I did write a script where I could define which files, in what order, and have the script build the file, while correcting internal url references so that image paths wouldn't break. Then I used an online compressor to minify the file. After all of this work (took the good part of an afternoon), I reload my pages and all of them function as intended, in all browsers. And, compressing 10 files down to 1, I now get my form styling, and my editor displays correctly, etc.

OK, is this the ideal solution? No, we'll still develop a comprehensive (and automated) strategy. But, this gets us over the deadline hump, and proves that the multiple stylesheets were the issue, and that this type of action can correct the issue. Proof of concept. You have to start somewhere I guess. But, how much easier web development would be if we didn't have to support Microsoft's mistakes.