

Creating a Theme from Scratch – Lesson 3

Debugging, Validation and Troubleshooting

As you discover in creating your own themes that life will go much smoother if you debug and validate at each step of your theme development process.

The full process will pretty much go like this: Add some code, check to see the page looks good in FireFox, validate, then check it in IE and any other browsers you and your site's audience use, validate again if necessary, add the next bit of code... repeat as necessary until your theme is complete.

Generally, if you write valid markup and code that looks good in Firefox, it will look good in all the other browsers (including IE). Markup and code that goes awry in IE is usually easy to fix with a work-around.

Now that we completed a fully functional theme in the last lesson, I probably should revise the workflow diagram more accurately. I am sure you understand a little better that it should be more like this (Figure 1):

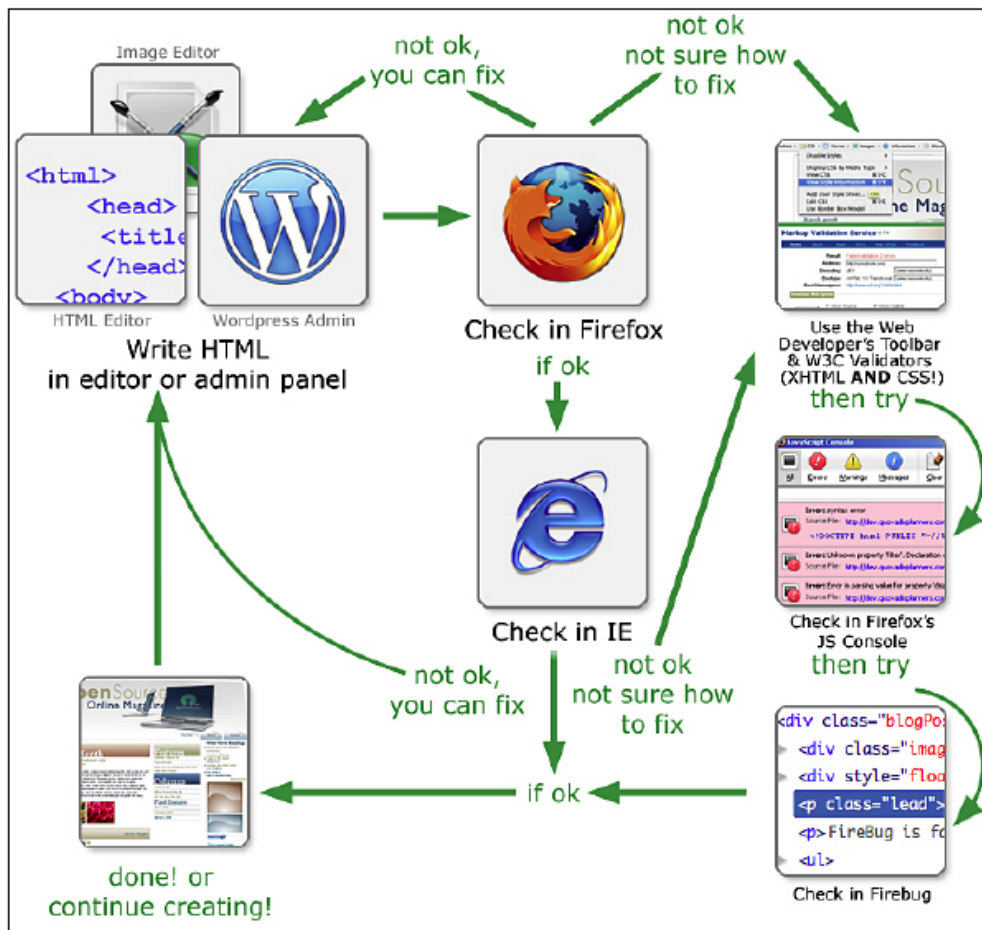


Figure 1 – A more accurate workflow of creating themes

You want to work with nice, small pieces or 'chunks' of code. I tend to define a chunk in XHTML markup as no more than one div section, the internal markup, and any WordPress template tags it contains. When working with CSS, try to only work with one id or class rule at a time. Sometimes, while working with CSS, you can break this down even further and test after every property, until the rule looks as intended and validates.

As soon as you see something that doesn't look right in your browser, you can check for validation and then fix it. The advantage of this work-flow is you know exactly what needs to be fixed and what XHTML markup or PHP code is to blame. All the code that was looking fine and validating before, you can ignore. The recently added markup and code is also the freshest in your mind, so you're more likely to realize the solution needed to fix the problem.

You should be regularly saving backups of your theme at good stable stopping points. If you do discover that you just can't figure out where the issue is, rolling back to your last stable stopping point and starting over might be your best bet to getting back on track.

Troubleshooting Basics

Suffice to say, it will usually be obvious when something is wrong with your WordPress theme. The most common reasons for things being 'off' are:

- Mis-named, mis-targeted, or inappropriately-sized images.
- Markup text or PHP code that affects or breaks the Document Object Model (DOM) due to being inappropriately placed or having syntax errors in it.
- WordPress PHP code copied over incorrectly, producing PHP error displays in your template, rather than content.

The first point is pretty obvious when it happens. You see no images, or worse, you might get those little ugly 'x'd' boxes in IE if they're called directly from the WordPress posts or pages. Fortunately, the solution is also obvious: you have to go in and make sure your images are named correctly if you're overwriting standard icons or images from another theme. You also might need to go through your CSS file and make sure the relative paths to the images are correct.

For images that are not appearing correctly because they were mis-sized, you can go back to your image editor, fix them, and then re-export them, or you might be able to make adjustments in your CSS file to display a height and/or width that is more appropriate to the image you designed.

For the latter two points, one of the best ways to debug syntax errors that cause visual 'wonks' is not to have syntax errors in the first place (don't roll your eyes just yet).

This is why, in the expanded work-flow chart, we advocate you to not only visually check your design as it progresses in FireFox and IE, but also test for validation.

Why Validate?

The problem with debugging purely based on visual output is, all browsers (some more grievously than others) will try their best to help you out and properly interpret less than ideal markup. One piece of invalid markup might very well look OK initially, until you add more markups and then the browser can't interpret your intentions between the two types of markup anymore. The browser will pick its own best option and display something guaranteed to be ugly.

You'll then go back and futz around with the last bit of code you added (because everything was fine until you added that last bit, so that must be the offending code) which may or may not fix the problem. The next bits of code might create other problems and what's worse that you'll recognize a code chunk that you know should be valid! You're then frustrated, scratching your head as to why the last bit of code you added is making your theme 'wonky' when you know, without a doubt, it's perfectly fine code!

Avoid all that frustration! Even if it looks great in both browsers, run the code through the [W3C's XHTML](#) and [CSS validators](#).

If something turns up invalid, no matter how small or pedantic the validator's suggestion might be (and they do seem pedantic at times), incorporate the suggested fix into your markup *now*, before you continue working. This will keep any small syntax errors from compounding future bits of markup and code into big visual 'uglies' that are hard to track down and troubleshoot.

PHP Template Tags

The next issue you'll most commonly run into is mistakes and typos that are created by 'copying and pasting' your WordPress template tags and other PHP code incorrectly. The most common result you'll get from invalid PHP syntax is a 'Fatal Error.' Fortunately, PHP does a decent job of trying to let you know what file name and line of code in the file the offending syntax lives

If you get a 'Fatal Error' in your template, your best bet is to open the file name that is listed and go to the line in your editor. Once there, search for missing `<?php ?>` tags. Your template tags should also be followed with parenthesis followed by a semicolon like `();`. If the template tag has parameters passed in it, make sure *each* parameter is surrounded by single quote marks, that is, `template_tag_name('parameter name', 'next_parameter');`.

Advanced Troubleshooting

Take some time to understand the XHTML hierarchy. You'll start running into validation errors and CSS styling issues if you wrap a 'normal' (also known as a 'block') element inside an 'in-line' only element, such as putting a header tag inside an anchor tag (`<a href`, `<a name`, etc.) or wrapping a div tag inside a span tag.

Avoid triggering quirks mode in IE! This, if nothing else, is one of the most important reasons for using the W3C HTML validator. There's no real way to tell if IE is running in quirks mode. It doesn't seem to output that information anywhere (that I've found). However, if any part of your page or CSS isn't validating, it's a good way to trigger quirks mode in IE.

The first way to avoid quirks mode is to make sure your DOCTYPE is valid and correct. If IE doesn't recognize the DOCTYPE (or if you have huge conflicts, like an XHTML DOCTYPE, but then you use all-cap, HTML 4.0 tags in your markup), IE will default into quirks mode and from there on out, who knows what you'll get in IE.

Note: Another item to keep track of is to make sure you don't have *anything* that will generate any text or code above your DOCTYPE. FireFox will read your page until it hits a valid DOCTYPE and then proceed from there, but IE will just break and go into quirks mode.

To Hack or Not to Hack

If for some reason, you feel you know what you're getting into and have intentionally used markup syntax that's triggering quirks mode in IE (or you just can't figure out why, or maybe your client insists on designing for IE6 for Windows), then it's time for some hacks.

The cleanest hack is the !important hack. I like it because it lets CSS still render as valid. However, you should note that the !important value is the valid syntax and meant to be used as an accessibility feature of CSS. It's not a value that was ever meant to affect the design.

The fact that IE does not recognize it is a bug and though it's very simple and easy to implement, it's not recommended to be used liberally as a design fix. The understanding is, eventually IE will fix this bug so that it adheres to accessibility standards and then your hack will no longer work (especially if IE doesn't change anything about how it runs in quirks mode).

To implement the !important hack, take the width, height, margin, or padding property that has the discrepancy in it and double it. Place the value that looks best in FireFox first and add the !important value after it. Then, place the value in the duplicate property that looks best in IE *below* the first property.

Advanced Validation

Thanks for your interest in the Intermediate WordPress: Creating a Theme from Scratch lessons by Frank Stepanski. To purchase the complete full lessons [click here](#).

Copyright 2009 © Frank Stepanski

Lessons, files and content of these classes cannot be reproduced and/or published with out the express written consent of the author.