

Web Standards and Business Process

Monday, 06 August 2007

If you're wondering why Web standards have anything to do with business, just look around at the myriad companies who now face the prospect of upgrading all their systems to connect to the web. Not keeping up with Web standards will soon cost these companies millions as they either close down business, or spend the money to make their systems talk to computers online in a fashion that is understandable and accepted by other systems.

Introduction

If you're wondering why Web standards have anything to do with business, just look around at the myriad companies who now face the prospect of upgrading all their systems to connect to the web. Not keeping up with Web standards will soon cost these companies millions as they either close down business, or spend the money to make their systems talk to computers online in a fashion that is understandable and accepted by other systems.

This whitepaper contains a review of two books and a discussion of web standards, including CSS and XHTML, and some known problems associated with deploying CSS-based sites. Most already know that the World Wide Web Consortium (W3C) is the international standards organization which recommends how the Web should work and how web pages should be coded for usability and accessibility. These recommendations are then born into standards as the W3C's member organizations take the new suggestions to the rest of the world and also try to apply it internally. The W3C is led by Tim Berners-Lee himself, and contributing organizations number about 428 including giants like Microsoft, Adobe, Macromedia (now owned by Adobe), Apple, Motorola, Oracle, Pitney Bowes, as well as a number of smaller companies.

These days, a discussion of Web standards is almost synonymous with XHTML 1.0 (the pairing of HTML 4.01 with XML standards), CSS (2 and 3), "Web 2.0," and the move toward semantically valid code (separating structure from content), table-free designs, and increased accessibility of content by end-readers other than browsers running on desktops or laptops. There are other criteria that Web standards are judged on, but the aforementioned items seem to be on everyone's list at the moment. Because Web standards constitute so much, use of the term has to be specifically footnoted so as to give meaning to what currently discussing. Since we cannot discuss web standards comprehensively in the space of an article, we'll limit ourselves to a discussion of CSS, some CSS hacks, and rules and guidelines for shifting from HTML to XHTML.

A review of two books

Increasingly, because of the move toward adherence to standards, and requests for companies creating browsers to show the same code the same way regardless of platform or version, it seems we are living the next phase in the evolution of the Web. Therefore there's a plethora of books on the subjects of Web design using CSS and Web standards, and on avoiding the pitfalls associated with this. I've reviewed two books here. They are:

- Bulletproof Web Design, Improving flexibility and protecting against worst-case scenarios with XHTML and CSS, by Dan Cederholm.
- Designing with Web Standards, 2nd ed., by Jeffrey Zeldman.

I bought Dan Cederholm's Bulletproof Web Design because of its promising title which seemed, upon quick flip-through at the bookstore, quite readable. I believed it would show the reader how to protect their front-end code from breaking in different situations, while following the oft-requested adherence to current web standards by today's CSS-centric web community.

Unfortunately, the book was a bit of a disappointment. While nothing presented is incorrect, and Cederholm does offer up some reminders of nifty techniques as well as a few new tricks, it certainly doesn't fulfill the title's mission. The author does a great job presenting code and design he deems not bulletproof, and then showing his bulletproof version. But the definition of bulletproof is arguable. "Bulletproof" in this case does not mean code that doesn't break; instead it means code that is not likely to break if certain conditions exist. Furthermore, he tries to address the conditionals by presenting some CSS hacks. I'll attempt to catalog and discuss the use of some hacks later on.

Cederholm starts off the book with a chapter on text size and how to make designs bulletproof by designating text in relative widths (like ems and the use of keywords such as small, medium and large) as opposed to fixed width pixel designation. The thesis is entirely logical, since using relative-width text does facilitate the ability to change the size of the text directly in the browser as well as with DHTML, allowing for increased readability for older, impaired or disabled visitors, while allowing for the flow of text to be rearranged (hopefully) without breaking the design. However, there's the if... if one designs layouts without relying on tables to position elements, and instead uses CSS layouts.

Strictly speaking, resizable text still works in tabled layouts. Of course, the table and cell heights will increase to accommodate any growing text, and after a point the layout will look broken if the text becomes too voluminous to be held within the confines of the cells, and adjoining cells containing image sections and other text blocks break apart in unfortunate ways. Cederholm comments in his introduction and in the middle of the book that bulletproof "never guarantees 100% protection" but as he is defining it, "means being prepared for whatever is thrown at your design." That seems, oddly, like saying "she's a little pregnant." It's either bulletproof or it isn't. Cederholm's book confirms nothing is really bulletproof.

Beyond issues of text size, Cederholm offers up options on how to do table-less layouts using <div> tags and CSS positioning instead of clear gifs to get elements and text blocks into the positions needed for visual design. But just about every tactic requires browser-specific hacks in the style sheets or DOCTYPE (mostly for Internet Explorer) to fix what isn't behaving as expected. It's not so extraordinary to have to use hacks to make CSS layouts work right, but the premise of the book was "bulletproof" techniques, for which I'm willing to hold the author's feet to the fire. For one, simply using hacks at all, though necessary for creating pages that work across multiple browsers and browser versions, should automatically disqualify most designs from being labeled bulletproof; secondly, hacks by their nature aren't bulletproof, even by the author's definition, since newly emerging versions of browsers tend to break CSS-hacked pages and you must modify your CSS and the hacks later on — as we have seen in the case of IE7's emergence onto the scene, even though Microsoft has tried to take into account the previous hacks. Bulletproof cannot mean bulletproof for now.

Another of the book's shortcomings is that the author uses some concepts interchangeably. In the chapter on how to design so that pages "devolve" gracefully when a browser lacks CSS support, Cederholm suggests a simple "10-second usability test" by turning off CSS and viewing the page without any styling.

I disagree. Though it's a good testing technique, viewing a page without styling does not constitute a "usability" test; it's more of an "accessibility" test which, while still important and closely connected to usability, is different.

Usability connotes a reasonable effort to create a site which has an easily understandable and largely navigable site overall (not just menu items) in direct relation to the intended design of the site. That design can be with lots of styling or without any styling at all, but usability depends on a user's understanding and ability to interact with what the designer intended to be viewable to visitors. What Cederholm is saying actually has more to do with accessibility, making allowances for poor browser support, and handicapped or physically impaired users, all very important for those with browser applications outside of the norm (including PDAs, cell phones, and screen readers and TTY devices for the visually impaired).

While people may still surf pages with images off to speed loading times, believing that the majority of people purposely turn off CSS these days is a bit ludicrous, in line with suggesting we still design for 640x480 resolution to help out those with older computers and smaller monitors. Many organizations are just now beginning to redesign their sites with enhanced accessibility in mind, following government requests and regulations in a nod to the ADA principles, whereas almost all sites have been very conscious of the importance of usability since the late nineties, when Jakob Nielsen declared himself as the usability guru of the Web. A site can be imminently usable as designed, while atrociously inaccessible to the impaired and handicapped. The two concepts should not be used interchangeably.

Since Bulletproof let me down in my search for a quick book on Web standards, I extended my search and came across the second book, Designing with Web Standards, by Jeffrey Zeldman — a n excellent book for anyone trying to understand what this "Web standards" thing really

means to them. While the reviews on Amazon.com warned Zeldman's writing suffers from repetitiveness, I find the book does cover a lot of ground and has excellent discussions about coding standards and tricks to bring your code closer to the W3C Web standards being pursued. While both books, more or less, approach the same topic, I'm less enamored with Bulletproof because of the expectations set up by the author, and by a lack of full explanations of less-familiar techniques. To be fair, because Zeldman's book is bigger and contains more relevant content, it's easier to forgive Zeldman's repetitiveness as writes with a much more knowledgeable and in-depth perspective on the matter, as opposed to Cederholm's CSS tricks which can be collected from the net by a bit of searching.

Zeldman, in one section, makes quick work of exemplifying the purpose of using standards and CSS layout while he undresses Microsoft by showing the bloated code used to render Microsoft's own header (which would take over 20 pages if he were to produce it fully). He then promptly unveils about 10 or 12 lines of code that could do the same job using CSS and HTML. Furthermore, he points out that although Microsoft's code validates in the W3C validator bot, it's still bloated and doesn't really follow the spirit of the W3C.

Lesson: just because it validates does not mean it is standards-compliant in the truest sense.

Zeldman's Web Standards also has a great discussion of XML and how it came to be the world's answer to data interchange not only on the web but for data everywhere. As he writes it, "the world took to XML like ... a boy dressed in his Sunday best takes to a pool of mud." This is a good precursor to talking about the XML prolog which sometimes resides above the DOCTYPE and namespace declarations in an XHTML page.

Web Standards is filled with great examples, and very detailed yet understandable, discussions of code, design, usability, and accessibility; furthermore, Zeldman keeps his similar but different terms straight. There is an entire section on the differences between HTML and XHTML and how to transition to the new standards and convert existing code to XHTML, examples of which will follow later.

CSS Hacks

Next, we'll visit CSS hacks. A quick Google of this topic will bring up numerous up-to-date pages on the subject, but we'll include a brief list of some interesting hacks here.

CSS hacks are usually utilized as workarounds to non-compliant browser behavior with style sheet selectors, as with Netscape 4.0, and various versions of Internet Explorer, the guiltiest of all. However, other browsers have been found to exhibit broken behavior as a result of applying the first hacks and therefore a second set of hacks usually follow to round out all the fixes. We won't cover every single fix here, but a comprehensive (not full) list of hacks can be found at <http://css-discuss.incutio.com/?page=CssHack>.

Let's start with the popular "box model hack," which may actually

become unnecessary soon, but has plagued Web designers a long time. The W3C "box model" standard states that the CSS DOM's content container, or "box," will take the width and height of the box as the starting dimensions and add padding, margin, and border widths to the box to come up with the total width. Win/IE 5.5, however, doesn't follow this rule and instead puts padding and border within the box, effectively subtracting the padding and border from the original. Therefore, a 100px wide box with padding: 5 and border: 5 will have a final width of 120 pixels in a compliant browser, while IE 5.5 will keep the final box width at 100px, but the content width reduces to 80px.

There are a few different fixes for this, but the most famous hack is credited to Tantek Çelik (therefore called "the Tantek method"). It's rather strange, and involves declaring the correct width first (120px in our case), following it with two instances of "voice-family" selectors that Win/IE doesn't understand, and ending with the actual width of the box seen by IE. Like so:

```
div.content {  
    width: 120px;  
    voice-family: "\}{}";  
    voice-family: inherit;  
    width: 100px;  
}
```

There are times when you actually want a browser to misbehave, because it may be easier to write the CSS targeted to that incorrect behavior. For instance, Win/IE Quirks mode: IE6 can be made to render a page as IE5.5 might (incorrectly) if you use the XML prolog normally necessary for XHTML headers. The line `<?xml version="1.0" encoding="iso-8859-1"?>` above the DOCTYPE declaration puts it into quirks mode, and causes it to create the box model improperly. Generally, anything above the DOCTYPE declaration will put IE into quirks mode. It's interesting to note that this behavior has been removed from IE7.

You can use the above two methods in conjunction with conditional comments that IE alone can see. For instance, throw IE into quirks mode, write the CSS as you might for Firefox and Safari to work properly (generally same CSS for both), and then use something like `<!--[if IE]> <link rel="stylesheet" href="../ie-only.css" type="text/css" /><![endif]-->`

To give IE its own stylesheet to follow. This involves actually creating a second set of styles for IE, but that may be easier than trying to manage all browsers' CSS expectations and interpretations in one CSS declaration or stylesheet.

Another rather annoying problem Web designers face these days with CSS and boxes is the “float bug.” This is another IE/Win bug in which long text in a floated <div> element gets chopped off, and the scroll bar disappears. To get the text back, you must refresh the page or hit F11 twice in quick succession. This is similar to yet another bug called “the guillotine” in which bottoms of floated elements get chopped off if you hover over links. While the exact percentage of people afflicted by these problems is unknown, they do happen. The float bug is known to happen as a result of IE’s incorrect caching of values it received from another page which previously called a selector of the same name. For example, when IE encounters a “text” div on page 1 to be 100px tall, it retains that value when it goes to page 4 with a “text” div 200px tall. Therefore the words in the “text” div on page 4 will get truncated as per the bug. The fix for these bugs can involve javascripting, or more simply by using the “height:1%” hack otherwise known as “Holly Hack.”

IE is not the only browser to require hacks. For example, for those (who?) still concerned with Netscape Navigator 4.0, is that NN4 wants to put make anything that has padding around it, into a block-level element. Therefore, having padding around a link, turns it into a block level element and causes whatever element they’re found in to break up and possibly overlap other content. The solution is to exploit the @import method of calling styles, and put any NN4 hating styles into the separate style sheet; NN4 ignores the @import command for CSS, and therefore never sees the offending code, while other browsers just add it to the main style sheet. Seeing as how NN4 is practically dead now, any pages that do use this hack are future-safe.

So with IE7 coming out, we are beginning to see reports that pages built using hacks for earlier versions are now showing their age and breaking when viewed in IE7. This is more evidence of the poor compliance of IE6 than anything else, but now we’ve got hacks to clean up. Googling “difference between IE6 and IE7” or a similar search produces a number of pages talking about this phenomenon. Some new CSS-isms are presented on this page, with a discussion of how to “unhack” pages for IE7:
<http://www.positioniseverything.net/articles/ie7-dehacker.html> .

A new hack that is most interesting from a design perspective is the “adjacent sibling selector” which says that using “+” in between two elements defines the phrase as “the sequence of...” For instance, writing “tr+tr” means “when you have one table row immediately following another table row.” In this manner you can style first table rows, or last table rows, or other combinations, with different styles to make your design more elegant.

But of course, all page would be better off without hacks. If Microsoft and the other browser companies would actually follow the W3C’s recommendations and turn them into real standards, then we wouldn’t have to waste so much time discussing “workarounds” for things that should work correctly in the first place. The W3C put down a set of rules and recommendations. The “internet law” should be, if you’re a contributing member of the W3C, you must get their blessing and get them to OK any browser releases. That would settle most disputes and would actually allow the W3C better control over their own standards, for what good is a standard that you can’t get your own member organizations to follow? Of course, this will not happen, at least not in the near future, but we can still hope.

Coding XHTML

Next, we're going to move to the subject of getting your code XHTML compliant. The natural question is, "Who Cares? Why does it matter?" Well, there are a number of reasons to write valid, well-formed XHTML, but the few that stand out are:

-
- Consistent. XHTML is more consistent than HTML, so it's less likely to cause problems
-
- Predictable. Newer browsers love XHTML code more than HTML4, and combined with the consistency mentioned above, make XHTML-written pages more predictable.
-
- Wireless. XHTML works in wireless devices (it's made to be so), and if you've noticed, wireless devices are demanding more of consumer dollars these days and is more likely to warrant the attention of companies wanting to develop for these devices. In short, that's where much of the money will be soon, getting web apps to work in wireless.
-
- Semantic. XHTML can help you avoid display inconsistencies and accessibility problems by putting you in the habit of separating content from design.
-

How do you get your code to be XHTML 1.0 compliant? Follow some rules and guidelines like:

- Start with a valid DOCTYPE, namespace, and content-type. There are 3 to choose from, Transitional, Strict, and Frameset. I would not use Frameset, as frames have been frowned upon since as long as I can remember Web history. It's available for those page that must still use framesets for business reasons, as in the case of About.com. Transitional seems to be the easiest to use currently for any designers trying to ... transition. If you've been introduced to XHTML for quite some time already, consider Strict.Namespace resides in a modified HTML declaration in which XML DTD elements are declared. It helps you identify the specific namespace you're using by pointing to its online location.Content-type indicates the kind of character set you're using. Declaring your content type, whether Unicode, or Latin-1 is necessary to pass validation tests. You must declare it so the validator knows what "language" you're speaking in.
- Write all tags in lowercase. Unlike HTML, XHTML is case conscious, since it is based on XML which is, again, case conscious. This also makes it easier to write code, since you'll be less concerned with making the code "look pretty." Element and attribute names should be lowercase, but the attribute values and content does not need to be. If you're charged with converting HTML to XHTML, and cringe at the

prospect of changing your tags, look at HTML Tidy, a freeware open source program at Source Forge.

- Quote all attribute values, as `inheight= "55"`
- Give all attributes a value, they require it even if you're declaring it as an HTML element. For instance, `<td nowrap="nowrap">` and `<hr noshade="noshade">`
- Close all tags, even empty ones like the break and horizontal rule. Use a space before the end slash: `<p> ... </p>
<hr />`
- Remove double dashes from within comment tags. Double dashes may only exist at the start and end of the comment tag. If you need to use a vertical separator, use equal signs instead, or use a space between dashes inside comments.
- Encode the `&` and `<` characters, replacing them with `&` and `<`. The W3C validator will give you a warning if you don't encode them, but a strict XML validator will give you fatal errors.

Those are the rules, and they're not very hard to follow. They just need a little getting used to, and if you use a coding program which helps write the tags for you, make sure you switch its workings in the options menu to write them in lowercase.

Conclusion

We'll end the discussion here. There's enough here for anyone who's starting to look at front-end Web standards to launch into their own research on the Web or to grab either of the two books I reviewed earlier. At this point, we'll wrap up with a brief "state of the union" future commentary, or "how it might work."

Coming from the print side of the publishing industry, and then having spent some time on the web front-end, backend, and management side in sequence, my view on the world of information design is changing from "bleak" to "hopeful."

It used to be bleak because regardless of how entrenched Web technologies had become, publishers — those responsible for disseminating the world's knowledge in various forms — could not see the importance of "doing things right" on the web, or of the importance of lean code, or of following any kind of Web standards. This kind of thinking has darkened the prospect of transitioning to a new revenue model for many years and has lost businesses millions, though the light is finally coming through I think. Some publishers are now moving in the right direction, recognizing the importance of Web standards and other issues, albeit some still going there kicking and screaming. At the center of the problem was the lack of understanding or support for reviewing the entire production system starting with the writers and editors.

The production people would put a print product together, without any regard for where it had to go from there. Unfortunately, they soon learned that they would also be the ones charged to lead the transition when CEOs asked the question, "What more can we do to earn money from our information? Where else can all our information go?" Well, if you wanted to sell information on the web, it had to go online. And then it had to go somewhere else, either offline again as another product, or to yet another online venue (as in selling articles to content aggregators for resale on the web). Unfortunately, they lost millions, and probably continue to do so because they didn't tool up in the early days for the paradigm shift that has come now (and is in full swing), especially being spotlighted by companies' desires to bring Web information to cell phones and Blackberry-type devices. What they needed from the beginning was the understanding that content should be separated from context and design, and that starting the process at that time would have brought them up to par with today's companies. Instead they are now playing catch up with all their systems, including bookkeeping, payables and receivables, circulation and more.

The insistence of designers and knowledgeable managers, and the diminishing of browser wars has brought Web standards into the full view of all professionals and industries whose work requires quick or almost instantaneous interchange of information, including financial institutions, publishers, logistics professionals, accounting departments, and even salespeople, to name a few. The advent of XML brought the promise of a "write once, distribute many" mantra, but could not be fulfilled until the other pieces fell into place.

The technology is now finally falling into place, though some may argue that it will be quite a while longer before we see the whole system work like a well-oiled machine. With CSS and XHTML, and the separation of content from design, we are seeing the nascent stages of a rebirth of commerce, and judging from the speed of implementation and change, most companies should be able to realize the full potential of EDI with Web standards in about 5-7 years.