

Why is Code Important?

Richard Merrill
Development Director, Autograff, Inc.

autograff inc

Why Is Code Important?

Richard Merrill, Development Director, Autograff Inc.

Why should anyone but techno-geeks care about the code behind content management systems? The answer is that the code carries a great deal more than what you see in the browser. It's common knowledge among web developers that the HTML standards of the 90s are no longer appropriate to today's conditions. Database connectivity, interactive scripting and XML applications require code that is consistent, clean and robust.

Consistency means, among other things, closing all tags. In HTML, the purpose of code is to tell the web browser what to show. Tags enclose some portion of the web page like bookends. For instance, the `<p>` tag indicates a paragraph.

```
<p>This is my paragraph text.</p>
```

The `<p>` symbol tells the browser where the paragraph begins. The closing tag, with a slash before the `p`, indicates the paragraph ends. Most browsers are lenient and display the paragraph correctly, even if the closing tag is missing. XML, a communication language that drives communication with mobile devices, new feeds, and many web functions, is not so forgiving. In order to unleash its power, it needs to work with consistent information. XHTML is the term used for HTML that is consistent enough to be used with XML.

Old-School HTML

Old-school HTML programmers solved many complex display challenges by overcoming native characteristics of HTML and building pages with tables. Tables were originally meant for displaying tabular data with no visual styling. The power of tables to control presentation was discovered bit by bit by the gallant programmers of the 90's. Their pages were dense with code specifying the complex relationships of rows and cells.

The New School

The new school of XHTML is driven by several factors: one is the growing group of designers and programmers who wanted to separate form and content on general principle. They reasoned that this would make control of web appearance much easier, with less code and therefore shorter loading times. Another factor is the rise of mobile devices. PDAs, cell phones that browse the internet, web tv, and other non-computer devices needed other means to get web content onto their small screens and tenuous wireless connections. A third factor is the growing awareness of the members of the web audience with physical and visual challenges. Web pages can be navigated by special assistive software; with the universal presence of the web, accessibility for web pages mirrors the social mandates for accessible spaces. Separating form from content is the major way to make web pages accessible. A visually impaired person might miss the cool rollover effect in the menu, but is probably more interested in the actual content of the page.

If appearance is controlled outside the web page, the page content is stripped of its overburden of presentation code, making it accessible to screen readers and other assistive devices. In fact, separating form from content makes it easier to design and program assistive devices, because they don't have to wade through a swamp of table and font tags.

This gives web developers a reason to learn to build pages without tables. The replacement for the table is the <div> element. A div is an empty box without features; the perfect container for whatever you want to put in it. By giving it a label, either a "class" or an "id" you can tell it how big to make itself, where to appear on the page, what color, what kind of border if any, and what font, size and color the text inside it will be. It's a whole different approach for web developers raised in the land of tables, but it's vastly rewarding in its flexibility.

Controlling the appearance of a page by means of a separate file meant that a given web page could be served in different forms just by substituting different control files. These files are called Cascading Style Sheets. The term "cascading" has to do with how style sheets are implemented. The term Style Sheets says it all.

An amazing example of the power of Cascading Style Sheets (CSS) can be found at Zen Garden, <http://www.csszengarden.com>. Here you can see one set of web content presented in many different ways just by the application of different style sheets. It's hard to believe that they are all using the same HTML code.

Global control

The style sheet suddenly gave designers and programmers awesome power. One could define the appearance of a paragraph just once, and style could be propagated across a site with thousands of pages. All they had to do was give every page in the website a single link to the style sheet, and suddenly the world would never be the same.

Old school HTML required the individual formatting of every paragraph in the paragraph itself (<p>) That meant that every paragraph on every page had formatting code in it. If you wanted to change the font on the website, it could mean hours of find-and-replace, with vast possibilities for errors and omissions.

Now, with each page linked to the style sheet, just change the paragraph style in the style sheet to display "Verdana", and every paragraph on every page in the entire site suddenly displays in Verdana instead of Arial!

Original style sheet definition:

```
P { font-family: Arial; font-size: 10pt; }
```

Change the word Arial to Verdana:

```
P { font-family: Verdana; font-size: 10pt; }
```

The programmer types seven characters, and fifty thousand paragraphs change! How can a web developer not get excited?

Font sizes are under greater control with style sheets. Formerly, sizes were given as "2" or "3" or "4". Size 4 was pretty large, and 1 was almost unreadably small, so the only practical choices were 2 or 3. With style sheets, percentages, points, pixels, and the mysterious "em" are all valid.

So we got greater visual control and lower cost for visual updates with the marvelous Cascading Style Sheet.

Enter the mobile device. Tables, used to construct page layouts for years, suddenly were unworkable in this more fluid environment with small screen displays.

There were ways to deal with these issues for years before they became popular, but that's the way with all technology. Why try something new until it's absolutely necessary? For all our fascination with the latest cutting-edge gizmo, we're conservative at heart. XML is implemented because people feel they need it, or will make money from it, or need to distribute information with it.

It's Only Right

Building pages with <div> elements instead of tables, writing consistent code, paying attention to accessibility, and conquering CSS are what current web developers are doing. It's good for many reasons beyond what I have outlined above. Applications that automatically create web pages, such as content management systems (CMS) and web page builders, still generally generate bloated table-heavy code. Those that use <div> elements, such as Quark Xpress and InDesign, create a plethora of uniquely labeled div elements absolutely positioned on a page, and producing an unwieldy style sheet. Such an approach defeats the whole purpose of XHTML, which is to make things easier.

CMS redefined

Content management systems can be built to be accessible, both in the pages they create and in the editing interfaces. Responsible application development standards mandate accessibility; in the interest of developing good relationships with developers who will implement the CMS, XHTML should become standard. Web projects funded in part by federal funds (and in many cases by state funds) are mandated by law to be accessible to people with functional impairments. According to the Centers for Disease Control, a 2002 study found 3.4 million Americans with visual impairments (<http://www.cdc.gov/ncbddd/dd/vision3.htm#common>). Four years later, the population of visually impaired web surfers is a significant audience for anyone's site. It is a cliché of the accessibility community that an accessible website is better for all visitors, because the information is well-structured and the code is lean. A cliché becomes a cliché because it's so often true.

Web code is thus implicated in fiscal and social policy decisions, in the same way that architectural design is implicated in decisions requiring accessible buildings. At the present, accessibility is a kind of banner one can wave as part of the *avant garde* of the new order; soon it will be both ubiquitous and required. Code and computer processes are at the heart of more and more of our experience; to the extent that web developers participate in creating this condition, awareness of the results of their work, and willingness to embrace new directions are a requirement of this age.

It used to be that stability was the hallmark of a good business; now agility within stability is required. XHTML embodies exactly that; a stable platform of code on which to build agile, accessible and powerful programs.

Richard Merrill is the Development Director of Autograff, Inc.

www.autograff.com