

Presentation Layer Best Practices

Web Standards Matters!

insight

Frederic Welterlin | October 2006



With the adoption of W3C recommendations by most web browser manufacturers, client-side web development has finally reached "critical mass" in terms of expectations for how web sites should look and function across the many web browsers and browser versions. This has been made possible by what is known as Web Standards.

Simply put, Web Standards is a set of web development methodologies that provide web content to the **maximum number of users** while ensuring forward compatibility and graceful degradation of the "browsing" experience. Web Standards not only lowers the cost of development and makes web content available to numerous Internet-enabled devices, but also promises to remain viable as current browsers evolve and as new browsers enter the market.

Benefits include:

1. "Gracefully degraded" content for browsers that are not able to support certain technologies, and "progressively enhanced" content for browsers that can;
2. One set of code that can be interpreted uniformly and consistently by all standards compliant browsers;
3. Increased SEO (search engine optimization) relevancy and accuracy;
4. Easier site management due to "separation of layers" (HTML, CSS, and JavaScript), and improved performance by using techniques such as SPRITES (more below);
5. Increased Accessibility (Accessibility is not just about disabled people! It is a design approach that allows web content to reach the maximum number of people, regardless of browser types or user viewing preferences).

Problem:

Many web sites being produced today are still using outdated presentation layer development techniques that are inefficient, expensive to maintain, and not universally accessible.

Solution:

Adopting Web Standards-based development methodologies -- separation of semantic, hierarchical HTML mark-up from CSS style sheets and JavaScript behavior.

Benefit:

Web Standards-based development improves web site performance, reduces development time, and increases availability of content to a wider range of Internet enabled devices.

The Wrong Approach

Consider the HTML code below and its associated CSS. Although the author has done well in separating structure from form and avoiding the use of HTML for presentation purposes, this example is still lacking in several areas. The author is suffering from what is commonly known as *divitis*, or, using the HTML "DIV" element to tag all of the content nodes. This creates several problems.

First, each DIV element is given an ID or CLASS attribute, which requires a nomenclature that the author has to first create, and all subsequent developers have to then learn. Secondly, the author has used presentation terminology in naming the CSS selectors (will "form-hdr-green" still make sense if a decision is made to change headers to blue?). Finally, any search spider or screen reader that scans this code will get no semantic meaning from the HTML tags themselves.

```
<div id="my-module">
  <div class="bdr-black">
    <div id="form-hdr-green">Submit Expenses</div>
    <div>
      <form action="doSomething.php" class="margin-lft">
        <div id="name">Your Name:</div>
        <input type="text" class="field" />
        <input type="submit" class="field" />
      </form>
    </div>
  </div>
</div>
```

```
#my-module      {margin-bottom:10px; font-family:arial,sans-serif; font-size:12px}
.bdr-black     {padding-bottom:5px; border:1px solid #000}
#form-hdr-green {color:green; font-weight:bold; font-size:14px; padding:5px}
.margin-lft    {margin:0; margin-left:10px}
#name          {float:left}
.field         {width:100px; margin-left:5px}
```

The Right Approach

The example from above can easily be re-written using hierarchical, semantic HTML and contextual CSS to achieve a more robust presentation:

```
<div id="my-module">
  <fieldset>
    <legend>Expense Report Form</legend>
    <h3>Submit Expenses</h3>
    <form name="myForm" action="doSomething.php">
      <label for="name">Your Name:</label>
      <input type="text" id="name" />
      <input type="submit" />
    </form>
  </fieldset>
</div>
```

```
#my-module          {margin-bottom:10px; font:12px arial,sans-serif;}
#my-module fieldset {padding:5px 0; border:1px solid #000;}
#my-module legend   {display:none;}
#my-module h3       {margin:5px; color:green; font-size:14px;}
#my-module form     {margin:0 0 0 10px;}
#my-module label    {float:left;}
#my-module input    {width:100px; margin-left:5px;}
```

The new code looks like the original when rendered in a browser window:

Submit Expenses

Your Name:

The new code improves the original design in several ways:

1. *Divitis* has been eliminated and replaced with semantic HTML that describes the meaning of content.
2. Style rules have been applied contextually, thereby eliminating the need for unnecessary references. The HTML is now much easier to read and maintain.
3. CSS is nicely encapsulated (i.e. FORM tags which are children of "my-module" will receive the left margin without needing explicit instructions).

Notice that even without style sheets, the new code is semantically meaningful and will provide any user with access to content:

Expense Report Form

Submit Expenses

Your Name:

Obviously, this is a simplistic example, but it does illustrate how using a Web Standards approach to client-side development can improve performance by reducing the amount of code generated; streamline development time by creating an organized, structured presentation layer; and provide content to more Internet-enabled devices, regardless of what presentation technologies each device can support.

Adding Unobtrusive JavaScript

"Unobtrusive JavaScript" refers to the implementation of JavaScript functionality in a way that enhances the user experience for those who have that capability, while not adversely affecting **core functionality** for those that do not. Well written, unobtrusive JavaScript uses standard design patterns to test for the existence of a node, attach an event to that node, and then apply the enhancement based on the user's event trigger:

Identify the attach point in the DOM	Create new markup to add to the page	Apply new markup relative to attach point
getElementById(); getElementsByTagName(); etc.	createElement(); setAttribute(); innerHTML etc.	insertBefore(); appendChild(); etc.

Ideally, JavaScript should locate the HTML element upon which to attach an event, create all of the content/behavior to be delivered, and then inserts the new content/behavior into the web page. This approach to adding functionality to a web page will ensure browsers that do not support JavaScript can still function properly. In the case of the form example from above, adding unobtrusive JavaScript is easy:

```
function enhanceForm() {
    // do something cool, then...
    document.myForm.submit();
}
```

```
<div id="my-module">
  <fieldset>
    <legend>Expense Report Form</legend>
    <h3>Submit Expenses</h3>
    <form name="myForm" action="doSomething.php" onsubmit="return false;">
      <label for="name">Your Name:</label>
      <input type="text" id="name" />
      <input type="submit" onclick="enhanceForm();" />
    </form>
  </fieldset>
</div>
```

The nice thing about this solution is the "onclick" event, located on the submit button, will override the default form submission for browsers that support this behavior. If the browser is not JavaScript capable, then the form is still submitted in the traditional manner. In other words, JavaScript is used to enhance the user experience, not replace it.

From here, we can actually take it one step further: removing the "onclick" event from the markup itself -- after all, it is best to have behavioral events instantiated via JavaScript, not by the markup. To do this, an event handler must be called when the page loads:

```
window.onload = function() {
  var doIt = document.getElementsByTagName("INPUT");
  doIt[doIt.length-1].onclick = function() {
    enhanceForm();
  }
}
```

Conclusion

Web Standards based development should not be regarded as additional tasks to be implemented -- it is a design approach that should be implemented by default. Presentation layer developers should be brought into the design phase early on to work with visual and interaction designers in helping to create a user experience that is compelling and dynamic for those browsers that can handle it (progressive enhancement), while providing a meaningful experience for those browsers that cannot (graceful degradation).

About the Author



As a senior presentation layer developer, graphical alchemist, and Web Standards advocate for Avenue A | Razorfish, Fred Welterlin's experience and areas of focus include designing, architecting, and programming client-side templates, providing project management and technical recommendations, and developing multimedia-based widgets.

Fred has ten years experience working as a user interface designer and developer -- responsible for conceptual and practical design development for a wide range of business and industry needs.

You may send your suggestions and questions to Fred at fred.welterlin@avenuea-razorfish.com

"When I'm working on a problem I never think about beauty, I only think about how to solve
- R. Buckminster Fuller

References

- *Designing with Web Standards*, by Jeffrey Zeldman
- *Unobtrusive JavaScript*, by Christian Heilmann
- *DOM Scripting*, by Jeremy Keith
- *Graded Browser Support*, by Nate Koechley (Yahoo! Inc.)
- The Web Standards Project
- The Web Standards Group
- The W3C Web Accessibility Initiative (WAI)

About Avenue A | Razorfish

Avenue A | Razorfish (www.avenuea-razorfish.com) solutions are entrenched in deep technology, rigorous analytics and a rich understanding of customer needs, including award-winning web media and creative, search marketing services, email marketing/eCRM, and world-class creative, design and implementation of customer websites, intranets and extranets. Avenue A | Razorfish operates three U.S. regions – East, West and Central – with offices located in major U.S. markets, the U.K., Australia, Germany and headquarters in Seattle. Clients include AstraZeneca, Best Buy, Disney, Kraft, Microsoft and Starwood Hotels & Resorts. aQuantive, Inc. and all of its operating units are committed to Internet privacy.

Avenue A | Razorfish
821 2nd Avenue, Suite 1800
Seattle, WA 98104
Phone: 206.816.8800 Fax: 206.816.8808
For more information please visit: avenuea-razorfish.com