

# Presentation Layer Best Practices

## Why Web Standards and Accessibility Matter

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 will 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 it also promises to remain viable as current browsers evolve and as new browsers enter the market.

Benefits include:

1. One set of code that can be interpreted uniformly and consistently by all standards-compliant browsers (as an added benefit, modularization integrates easier with content management systems and allows for multiple presentations of the same content);
2. Semantic markup increases Search Engine Optimization (SEO);
3. Graceful degradation, the logical, hierarchical presentation of content is still available for browsers not able to support certain technologies, and progressive enhancement, which allows modern, capable browsers to show more content and advanced interactions;
4. Separation of the three main client-side layers (HTML mark-up, CSS presentation, and JavaScript behavior) allows easier site management, better uniformity and consistency, and reduces redundancy and bandwidth requirements;

### **Problem:**

Many web sites being produced today are still using out-dated 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 (for presentation) and JavaScript/DOM (for behaviors).

### **Benefit:**

Web Standards-based, accessible web design will improve web site performance, reduce development time, and increase availability of content to a wider range of Internet enabled devices.

5. Increased web site loading performance by shifting traditionally graphic-based interface elements to DHTML, using encapsulation/inheritance techniques in CSS and JavaScript, separating the "layers" to reduce redundancy, etc;
6. Accessibility of content. **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 (desk top web browsers, PDAs, text to speech readers) or user viewing preferences (text zooming, etc).

## The Wrong Approach

Consider the “snippet” of HTML below and its associated CSS. Although the author has done well in separating structure from form and avoiding the use of HTML to create the presentation, 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. This creates several problems.

First, each DIV element is given an ID or CLASS attribute, which requires a nomenclature the author has to first invent and all subsequent developers will need to learn in order to modify this code. Secondly, the author has used presentation terminology in naming the CSS selectors. For instance: will “form-hdr-green” still make sense if a decision is made to have headers in blue? And finally, anyone reading this snippet must rely on the ID or CLASS attributes themselves in order to understand what this code represents. Without style sheets, this markup has virtually no semantic meaning, and, is therefore dependent on styles to give it meaning.

```
#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}
```

```
<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>
```

## 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:

```
#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}
```

```
<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>
```

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

### Submit Expenses

Your Name:

The new code improves the original design by several factors:

1. The use of semantic HTML markup (such as FIELDSET and LABEL, etc) ensures graceful degradation in terms of providing meaningful content to non CSS capable browsers.

2. *Divitis* has been eliminated and replaced with more compacted semantic HTML that more closely tags the content for what it actually represents.
3. Style rules have been applied contextually, thereby eliminating the need for references. Notice that the CSS is nicely encapsulated, ensuring that only FORM tags which are children of “my-module” will receive a left margin of 10px. The markup is now much easier to read.

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 behaviors in a way that enhances the user experience for those who have JavaScript capable browsers, but does not affect the **core functionality** for browsers that do not. Well written, unobtrusive scripts tend to use the same methods and attributes to find, add, and/or modify content on a web page:

Identify the attach point on the web page	Create new markup/content to add to web page	Place new markup/content relative to attach point
<pre>getElementById(); getElementsByName(); etc.</pre>	<pre>createElement(); setAttribute(); innerHTML etc.</pre>	<pre>insertBefore(); appendChild(); etc.</pre>

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 behaviors 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 (in part by using “return false” on the FORM element). If the browser is not DOM capable, then the form is still submitted in the traditional manner. In other words, JavaScript is used to enhance the user experience; it is not used to replace core functionality.

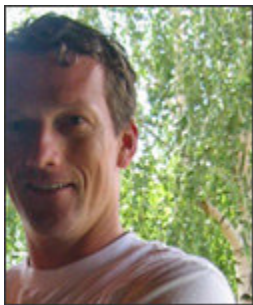
From here, we can actually take it one additional step further: removing the “onclick” event from the markup itself -- after all, it is best to have behavioral events instantiated via JavaScript, not by the markup. To do this, an event handler must be called when the page loads, awaiting the user to click on the last INPUT element for “my-module:”

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

## Conclusion

Web Standards and Accessibility-based development should not be regarded as just additional tasks to be implemented -- it is really a design approach that starts right from the beginning. Ideally, 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 still providing logical content structures 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](mailto:fred.welterlin@avenuea-razorfish.com)

**"When I'm working on a problem I never think about beauty, I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it's wrong."**

*- 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](http://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](http://avenuea-razorfish.com)