Separate Positioning from Presentation

By Kevin P. Wojdak

The growth of browser-based application development combined with the flexibility of the browser gives interface designers the opportunity to create very sophisticated visual applications and websites. In doing so, the browser interface itself can become very complex to build. The old way of building websites involved using HTML table elements to structure page layouts but, that technique quickly makes the source code messy, confusing, and difficult to manage. There's now a better way to structure your HTML and CSS to address this complexity, CSSP or Cascading Style Sheet Positioning. However, CSSP alone won't solve all of your problems. You need to have a tighter, more controlled relationship between the different technologies that define the browser-based interface. That's what this article is all about… giving yourself more control over your layouts while also giving yourself highly maintainable and manageable source code.

Be Object-Oriented

CSSP allows you to position each element of your website or interface on the page by embedding the positioning information in your style sheets. The real trick of this technique is in how you manage the information. You need to go beyond designing your interface to architecting your style sheets as well. The most effective way to architect your style sheets to keep them maintainable and manageable is to think of the elements on your pages as objects and to design them as reusable components.

The idea of reusable components is a concept coming from object-oriented programming (OOP). CSS has many similarities to OOP and is a great counterpart to a browser interface sitting on top of an object-oriented programming backend. The key to making reusable components is to understand and exploit the structuring power of CSS.

Begin by examining your page design and breaking it down into objects and object groups, and then begin to map out your basic CSS. For example, if your design is for a shopping cart interface, you probably have buttons to process different actions. Look at all of these buttons together and find their similarities and differences. Then, start building your style sheet from the ground up by defining the basic attributes of the buttons in CSS Type selectors. After that, group similar presentation attributes of the buttons into Class selectors such as "Enabled" and "Disabled" buttons. Finally, as you start to build the actual pages, any unique attributes of an individual object should be defined in a unique CSS Id selector just for that object.

By defining stylesheets using a structured and controlled method, just as in object-oriented programming, you create reusable Type and Class selectors. Reusing styles reduces the size of your stylesheets and makes it easier to manage and maintain them.

Remove Positioning from the Presentation

Be object-oriented in your approach to HTML development and combine it with CSSP and a little thought as to the architecture and your style sheets evolve. The most important aspect of this approach is to separate the Positioning of an object from the Presentation styles. By embedding the majority of the Presentation style attributes of an object into the CSS Type selectors and Classes, reusable object Classes are created reducing the size of style sheets.

Positioning of the objects is achieved by embedding positioning information in CSS Id selectors matched to individual objects on a page. The position coordinates of each object on a page are unique attributes of an object; the positioning information is embedded in a uniquely identified Id selector.

The approach is a variation of CSSP that can be thought of as the "2P" approach to CSSP. The "2P" stands for the two distinct referencing points of every object that should be understood and stay separated for the technique to remain effective. The two P's are Presentation and Positioning. As you create objects and position them on screen, the 2P's become a most effective and efficient tool.

The first 'P' is Presentation and accounts for all of the style attributes of an object that make up its visual display on screen. For object presentation, the generalized attributes are stored in CSS Type selectors and reusable Classes.

CSS Type selectors contain all of the most basic, reusable aspects of an object shared by every object of that type in the interface.

Classes, on the other hand, allow the common object attributes to be generalized and grouped together under a single reusable name. That Class name is applied to any object that shares common functionality with one or more other objects.

For example, to have two identical buttons positioned on two different areas of the screen, you generalize the common attributes for both buttons into a Class selector and then create unique Id selectors for their positioning. Some common attributes to place in the Class are the height and width of the object, which remain constant; border sizes and colors; the cursor style; and anything visual about that object.

The second 'P' is the Positioning of the element on screen. Take all of the unique object attributes that cannot apply to any other object or Class and store that information in a CSS Id selector. The Id selector is unique because it applies to only a single object with a matching Id on your page. Since it is unique, you may not have more than one object sharing the same id on a page. Unique attributes that typically go into an Id selector are "position", "top", and "left" or "right" coordinates. Other attributes placed within Id selectors are the unique variations of a generalized Class that apply to only that object on the page.

By separating the Positioning from the Presentation, the author has control over the HTML markup and less style redundancy in style sheets. Using a generalized Presentation Class removes excess style definitions from the CSS. The same style attributes aren't repeated across multiple similar Classes but are instead contained within a single reusable Class.

Example One

To illustrate the 2P approach, imagine the author is placing two identically shaped and styled gray rectangles on an interface. They share the same width, height, color, and border. The only attributes they don't share are position on screen.

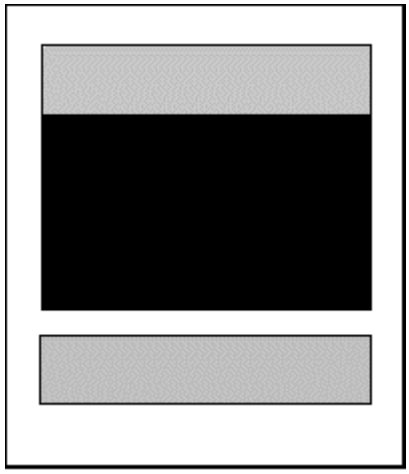

Figure 1

The gray areas use objects with unique id's of "area1" and "area2" and a Class named "graybox". The black area behind is used as an enclosing space or "container" with an id of "blackbox". The entire page is placed within an empty object identified as "pagebase" providing a container for all objects on the page.

The style sheet entries are:

```
/* Content Base */

#pagebase {
    position: absolute;
    top: 0px;
    left: 0px;
}


/* Generalized Classes */

.graybox {
    color: gray;
    border: 1px solid #000;
    width: 300px;
    height: 50px;
}

.blackbox {
    color: #000;
    border: 1px solid #000;
```

```
    width: 300px;
    height: 480px;
}


/* Unique objects */

#blackbox {
    position: absolute;
    top: 100px;
    left: 100px;
}

#area1 {
    position: absolute;
    top: 0px;
    left: 0px;
}

#area2 {
    position: absolute;
    top: 600px;
    left: 100px;
}
```

Since “area1” is positioned within “blackbox”, it is positioned in relation to and becomes a part of “blackbox”, its container or parent object. It appears at coordinates “0,0”, or “top: 0px” and “left: 0px”, of “blackbox”. The “blackbox” object can then be positioned anywhere on the screen and “area1” will move with it without losing its position within “blackbox”.

The HTML markup is simple, easy-to-read, clean, and easily modified. Clear source code such as this makes it easy to follow XHTML standards when writing your markup.

```
<html>
<head></head>
<body>
<div id=”pagebase”>
        <div id=”blackbox” class=”blackbox”>
                <div id=”area1” class=”graybox”>
        </div>

        <div id=”area2” class=”graybox”></div>
</div>
</body>
</html>
```

The author changes the look of the page by changing the position of objects on the page entirely through the CSS. This is very powerful, as the HTML source code doesn’t need to change.

Example Two

The author decides to change the page in Figure 1 by moving the bottom gray box, “area2”, up to the top of the screen to overlap “area1” offsetting it about 50 pixels down and to the right of “area1”. The new page looks like Figure 2.

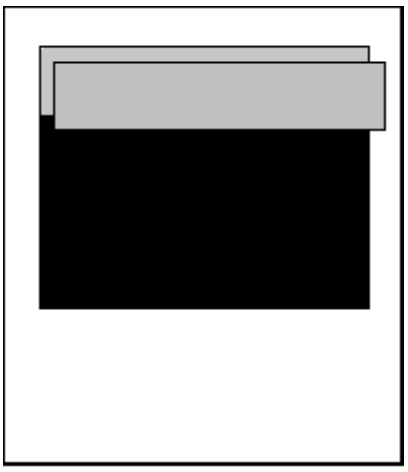

Figure 2

Simply change the “area2” Id selector by adjusting the “top” and “left” style attributes to new coordinates. Since “area2” will be 50 pixels below and to the right of “area1” and “area1” is within the “blackbox” container, we take the coordinates of “blackbox” which are “top: 100px;” and “left: 100px;” and add 50 pixels to both values.

The updated CSS for “area2” is:

```
#area2 {
    position: absolute;
    top: 150px;
    left: 150px;
}
```

That’s the only change to make to this object to shift it on the page. Since it is a positional change, the change is to the Positioning selector (Id) and the Presentation selector (Class) is left untouched.

Any object’s position on an interface can be moved simply by changing its “top” and “left” coordinates in a linked Id selector. The generalized object Classes do not need to be modified unless a global change is to be made to the Presentation of the entire group of objects on the interface.

The power of this technique may not immediately be apparent until changes are made to a large or complex application interface with several different screens that contain many common objects. Application architecture needs to go beyond the interface and application code down to the level of planning useful and maintainable style sheets. This technique is most effective and provides the most value when building complex applications and large Web sites.

Why Use the Technique?

- Reusable – Well thought out CSS reduces the size of your style sheets through reusable object Classes and Types.
- Standards – XHTML coding standards are easily followed within this type of framework because the code is simplified. Applying XHTML standards to the code requires all object styles to be embedded in external style sheets making HTML markup very compact and easy to read.
- Clean Code – Types, Classes, and Id selectors in external style sheets control all Presentation and Positioning of displayed objects. External style sheets remove embedded code congestion.
- Structure – CSSP using the 2P approach is the combination of several technical aspects of building an HTML interface. Applying these techniques and standards together puts logical structure around interface markup without using tables to hold a page together.
- Control – The technique, as a variation of CSSP, helps the author mimic designs flawlessly in the HTML.
- Compliance – HTML markup is compliant with many different browsers that support the latest CSS versions.
- Granular positioning – Precision positioning adjustments of objects down to the single pixel level.
- Manageable – In constantly changing environments, positions of objects are modified through minor CSS Id selector adjustments.
- Integration – Clean and easy to read code is more easily integrated with dynamic backend code such as JSPs (Java Server Pages).
- Localization – The separation of Presentation from Positioning makes this technique very efficient when localizing Web sites or application interfaces to different languages.

Kevin P. Wojdak is a freelance web and IT professional in the Chicago area. His skills and experience have allowed him to pioneer new techniques and tricks in Rich Interface Development. He enjoys transforming other people's creative inspiration into technical reality. Visit his personal website, Woj's Twisted World (http://www.wojzworld.com) to see his resume or to find other articles and development techniques.
