Section 9: CSS

We can use Cascading Style Sheets (CSS) to define the presentational details of our XHTML pages. This is in line with the basic premise of XHTML, which is that it exists for semantics; just as XML data can represent menu data for a pizza chain, XHTML should stick to representing the structure and content of your pages, rather than telling the browser how to display it.

The advantages of using CSS, as opposed to mixing up presentation and semantics in one XHTML page, include:

- Mixing up presentational details and XHTML can lead to problems later on, leading to maintenance and code readability issues
- CSS takes creativity to a whole new extreme: CSS affords far more control over presentation than XHTML
- Using CSS means your pages will be more accessible, both to people and to devices

In addition, if we specify our CSS in a separate file and link to it (instead of embedding CSS within our pages using the "style" attribute), we can gain more flexibility and end up with pages that are easier to maintain. We can link to CSS files by including the following in the <head> element of our pages:

```
<link rel="stylesheet" href="mystyles.css" type="text/css" />
```

We'll see later on that because our CSS can be linked to in this way, dynamically modifying the stylesheets that get included in our document (either on the client side, using JavaScript, or on the server side, using something like PHP) is trivial and can often be used to produce interesting and useful results.

A Three-Column Layout

Oftentimes, web developers will utilize a three-column layout for their pages: a narrow left column for links, a wide middle column for the content, and a narrow right column for, perhaps, a calendar that links to blog entries from a particular date, or maybe even a tall banner ad.

The lazy way to implement this is by using tables: one table row with three table cells. Take away the borders, tweak the cell padding and spacing a little bit, and you're done! Or so it may seem. But the mess of tables you have just created will cause all sorts of problems later on; what happens if a user on a mobile browser views the page? Will a screen reader be able to correctly figure out what it's seeing is a messy layout, rather than a table of figures?

The recommended way to create a three-column layout, of course, is to use CSS. We start off with the following HTML code:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
      <title>Three-Column Layout</title>
      <link rel="stylesheet" href="columns.css" type="text/css" />
</head>
<body>
    <div id="leftcolumn">
    <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">Dolor</a></li>
    </ul>
    </div>
    <div id="rightcolumn">
        <h2>About this site</h2>
        <p>This site is very nice.</p>
    </div>
    <div id="middlecolumn">
        <h1>Welcome</h1>
        <p>Content goes here</p>
    </div>
</body>
</html>
```

Figure 1: columns1.html

Notice that we have three divs: one for the left column, one for the right column, and one for the middle column, in that order. Notice that even though we intend for the right column to "come after" the middle column, we've put it above the middle column in the code. This, as we'll see later, is due to our CSS markup. Depending on the situation, however, you'll be able to put such divs in any order you want and have them show up exactly where you want them. Such is the power of CSS.

First, let's get started with some very basic CSS that will help us figure out what is going on:

```
body { font-size: 10pt; }
#leftcolumn { border: solid 1px blue; }
#rightcolumn { border: solid 1px red; }
#middlecolumn { border: solid 1px green; }
```

Figure 2: columns1.css

This stylesheet is mostly for debugging purposes; it gives each of the divs we saw above different border colors so we can see where each of the divs is being placed. In addition, it also sets the body (and all its child elements) to have a font size of 10pt.

If you open this page with this particular CSS, you'll see that we see what we would expect: each box, now with a border color, is stacked up on top of each other, in the order that they are presented in the XHTML code.

To this, we can add one effect—making the left column "float" towards the left, and making the right column float towards the right—to get one step closer to what we want. Simply add "float: left" to the left column's CSS declaration and "float: right" to the right column's CSS declaration.

```
#leftcolumn {
    border: solid 1px blue;
    float: left;
}
#rightcolumn {
    border: solid 1px red;
    float: right;
}
```

Figure 3: columns2.css (excerpt)

The effect of this CSS is that the left column will, indeed, be on the left, and the right column will be on the right. So far so good. However, it's not what we intended: the two narrow columns are boxes, not columns, and they are now embedded in the content.

We can solve this problem by doing two things: setting a particular width for the columns, and leaving the same amount of space as whitespace (the margin) of the content column. That is, the content column will vacate *x* pixels on either side of itself, and the two narrow columns will occupy those vacated pixels on either side of the content column.

```
#leftcolumn { float: left; width: 150px; }
#rightcolumn { float: right; width: 200px; }
#middlecolumn { margin-left: 150px; margin-right: 200px; }
```

Figure 4: columns3.css (excerpt)

You'll see that we now have the intended effect; we have three columns: the left column is 150px wide, the right column is 200px wide, and the middle column takes up everything that's left. If we're not perfectly happy with the space between the columns, we can adjust by using the "padding" of the columns (the *margin* is what lies between other block elements and the "box" surrounding an object; the *padding* is the space between the object itself and the "box" that surrounds it).

For example, we can add "padding: 0 10px;" to the declaration of the content column, causing the div to have no top or bottom padding, and 10px padding on the left and right. This works because the "padding" shorthand property takes either of three forms:

- padding: [padding for all four sides];
- padding: [top and bottom padding] [left and right padding];
- padding: [top padding] [right padding] [bottom padding] [left padding];

These forms are the same for the "margin" and "border" shorthand properties.

Unorthodox lists

The <ul> element (unordered list) can be used to serve a great number of purposes with CSS. One favorite among Web 2.0 developers is to use them as navigation bars, either in a horizontal or vertical configuration. You'll see in our original XHTML code we employed an unordered list to create a list of hyperlinks. At the moment, with the default CSS, they're being rendered as a simple, unattractive-looking links with bullets.

We can enhance their presentation by, for example, removing the bullets, giving each item some extra vertical padding, and showing the user some feedback when they hover their mouse over the elements:

```
#leftcolumn > ul {
    padding-left: 10px;
    list-style-type: none;
}
#leftcolumn li {
    padding: 3px 0;
}
#leftcolumn li:hover {
    background-color: #BDBDBD;
}
```

Figure 5: columns4.css (excerpt)

These CSS declarations use more advanced *selectors*, or different ways to select which rules to apply to which XHTML elements. The first, "#leftcolumn > ul", builds on the "#leftcolumn" selector we were using in the previous examples by qualifying it with "> ul", which means "any <ul> element that is *directly* the child of the object with an ID of leftcolumn". You'll see that we've set the list-style-type attribute of the <ul> to "none", meaning that it will have no bullet points.

The second selector used in the example, "#leftcolumn li", on the other hand, specifies "*any* element that falls under an object with an ID of leftcolumn". Therefore, a <li> element could be a direct child of the div, a grandchild of the div, or any other child of the div, and the rule specified would apply. You'll see that in the example XHTML, the <li> elements in fact fall two levels underneath the div with the ID of leftcolumn.

The third selector used here, "#idcolumn li:hover", means "the rules to apply when a <li> element that falls anywhere underneath an object of idcolumn *is hovered over*". More often seen in the context of an anchor tag ("a:hover"), the ":hover" *pseudo-class* applies only when the element specified is hovered over. (Similarly, ":link", ":visited", ":active", and ":focus" are also pseudo-classes.) Because we've set the background color only when the element is hovered over, the element will have a background color when the mouse is over it, but not when it isn't.

Dynamically modifying CSS: Using PHP

As we've seen, changing the CSS stylesheet for a page (even when the underlying page remains the same) can drastically change the appearance of our pages. One of the great things about dynamic web programming is that we can change the stylesheet that our users get to change their aesthetic experience.

The simplest way to present different CSS stylesheets to users is to simply modify what stylesheet they get linked to. Because the <link> tag is XHTML, just like any other tag, its output can be controlled by PHP. In fact, the example three-column layout page on the course website is generated by a PHP script that changes the CSS files on the fly:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title>Three-Column Layout</title>
    <link rel="stylesheet" href="columns<?php
    if (is_numeric($_GET['css']))
        echo $_GET['css'];
    else
        echo 1; ?>.css" type="text/css" />
</head>
```

Figure 6: columns.php (excerpt)

This script has a very simple purpose; when given a GET argument, "css", it modifies the output of the <link> tag. For example, when $_GET['css'] is "2", it will link to columns2.css; when the argument is not specified or is not numeric, it will link to columns1.css.

You can do better, for example, by randomly assigning users a different CSS file, or even by saving user preferences in a database or session, and linking to the users' favorite CSS file depending on those preferences.

Dynamically modifying CSS: Using JavaScript

Now that we've used JavaScript for a whole array of things, we can leverage its power by modifying CSS on a page without requiring a page reload, in pseudo-Ajax style.

This we can do by programmatically adding a new “link” element to the <head> tag of the document:

```
function changeStyle(styleName) {
    var headID = document.getElementsByTagName("head")[0];
    var newNode = document.createElement("link");
    newNode.type = "text/css";
    newNode.rel = "stylesheet";
    newNode.href = "/styles/"+styleName+".css"
    headID.appendChild(newNode);
}
```

Figure 7: changestyle.html (excerpt)

This JavaScript function, when given one argument (the style name, like “blue” or “spotted”), has the effect of artificially creating a new <link> element with the appropriate “type”, “rel”, and “href” attributes, and then adding that <link> element to the <head> element of the current document. If we call this JavaScript using a selector box’s onChange event, for example, we can have a drop-down box that allows users to change the style of the current page on the fly.

Of course, all older stylesheets will remain in effect on the page; however, because CSS styles that come later on will override previously defined ones (with some exceptions), we can continue to add <link> tags to our <head> element to change our style, provided that we override all necessary CSS rules with new ones.

This kind of CSS manipulation is useful when dealing with, again, user preferences; also, you may want to use it if you want to increase the size of the text on a page using a dynamic script, for example.