CHAPTER 7

Using CSS

Cascading Style Sheets have been around since the end of 1996. Despite the relative longevity of the technology, its use in real-world web design has been limited to managing fonts and color, at least until recently. This limitation was imposed by the lack of consistent browser support. Because not all browsers managed CSS equally (if at all), it has been very difficult for designers to tap into the true power of style sheets. Instead, there's been a reliance on HTML for presentation.

Now we have far better support for CSS, so to tap into its many valuable features, web designers are moving away from HTML as a means of adding style and laying out pages, and into pure CSS design. Why is this so important? The reasons are many:

- Keeping presentation separate from the document means you can style that document for numerous media, including the screen, print, projection, and even handheld devices.
- Separating presentation from the document means a lighter document, which, in turn, means the page loads and renders faster, making for happier visitors.
- CSS offers ways to control one document or millions of documents. Any time you'd like to make a change, you change that style in one location and automatically update to all the documents with which that CSS is connected. In HTML, this couldn't be done.
- CSS documents are cached. This means they are loaded into your browser's memory *one time*. As you move within a site, the browser never has to reinterpret the styles. The results are more fluid movement from page to page and faster-loading pages, which, of course, is always desirable.
- By separating presentation from structure and content, accessibility is easily achieved. Documents that don't have heavy tables and lots of presentational HTML are inherently more accessible than documents that do.

Clearly, CSS offers a lot. In this chapter, you'll learn how to set up CSS to be most efficient and flexible for your designs.

CSS Theory Simplified

Before you can actually put CSS to use, you need to know some important things about the language and how to use it effectively. I'll try to make this quick and painless because I know you want to get down to it.

CSS Rules

CSS rules are made up of a selector and at least one declaration. A selector is the code that selects the HTML to which you want to apply the style rule. We'll be focusing on common selectors in this book, but you can use more than a dozen selector types as you become more proficient at CSS. A declaration is made up of at least one CSS property and related property value. CSS properties define the style:

```
h1 {color: red;}
```

The `h1` is the selector for your `h1` headers, and the declaration is made up of the `color` property with a value of `red`. Simply said, this rule turns all `h1` elements red. You'll note that the syntax of the style rule looks different from what you've seen in HTML. The curly braces contain the declarations, each property is followed by a colon, and a semicolon is used after each property. You can have as many properties as you want in a rule.

Applying CSS

Six types of style sheets exist:

- **Browser style**—This is the default style sheet within a browser. If you declare no style rules, these defaults are applied.
- **User style**—A user can write a style sheet and make it override any styles you create by changing a setting in the browser. These are used infrequently but can be helpful for individuals with special needs, such as low vision. In such a case, the user will create high-contrast, large-font styles that override your own.
- **Inline style**—This is style that is used right in the individual element and applied via the `style` attribute. It can be very useful for one-time styles by element, but it isn't considered ideal.
- **Embedded style**—This is style that controls one document and is placed inside the `style` element within the HTML document.
- **Linked style**—This is a style sheet that is linked to an HTML document using the `link` element in the head of the document. Any document linked to this style sheet gets the styles, and here's where the management power of CSS is found.

- **Imported style**—This is similar to linked styles, but it enables you to import styles into a linked style sheet as well as directly into a document. This is useful in workarounds and when managing many documents.

You'll see examples of these style sheets as we progress throughout the chapter.

The Cascade

People often wonder where the term *cascading* comes from. The cascade is an *application hierarchy*, which is a fancy term for a system of how rules are applied. If you examine the five types of style sheets just introduced, you'll notice that there are numerous means of applying style to the same document.

What if I've got an inline style, an embedded style sheet, and a linked style sheet? The cascade determines how the rules are applied. In the case of style sheet types, user style overrides all other styles; inline style trumps embedded, linked, and imported styles; embedded style takes precedence over inline style; and linked and imported styles are treated equally, applying everywhere any of these other style sheet types are not applied. Browser style comes into play only if no style for a given element is provided; in that case, the browser style is applied.

The cascade also refers to the manner in which multiple style sheets are applied. If you have three linked style sheets, the one *on the bottom* is the one that is interpreted if any conflicts exist among those styles.

Inheritance

Inheritance means that styles are inherited from their parent elements. Consider the following:

```
<body>
<h1>My header</h1>
<p>Subsequent Text</p>
</body>
```

Both the `h1` and `p` elements are considered children of the `body` element. The styles you give to the body will be inherited by the children until you make another rule that overrides the inherited style. Not all properties, such as margins and padding, are inherited in CSS but almost all others are.

Specificity

Finally, if there are conflicts within any of your style sheets that aren't resolved by the cascade, CSS has an algorithm that resolves the conflict. This algorithm is based on how specific a rule is. It's a bit heavy for this discussion but worthy of mention.

Obviously, two pages can't really do justice to any of these topics, so if you're interested in learning more, be sure to look at the *Additional Resources* section.

Adding Style Inline

Okay, enough with the theory—let's get down to work! Here you'll learn to apply inline style. You'll use inline style infrequently because it styles only the element to which it is applied. This defeats the management power of CSS.

What's more, inline style can be equated with presentational HTML because it goes right into the document instead of being separated from it, defeating the primary benefits of CSS. I use inline style mostly for situations in which a quick fix for a single element is called for, or in rare cases when it's the only style for one unique element in an entire site.

Consider the following element:

```
<h1>Welcome!</h1>
```

If this header were part of a complete HTML document and you viewed it in a browser, the results would be equivalent to Figure 7-1.

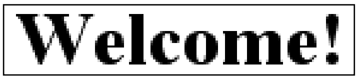


FIGURE 7-1 Default size of an `h1` as defined by browser styles.

Say you don't like the default color and size. You can add CSS rules directly to the element using the `style` attribute:

```
<h1 style="color: gray; font-size: 24px;">Welcome!</h1>
```

Now you've got a gray header sized at 24 pixels (see Figure 7-2).

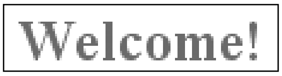


FIGURE 7-2 Redefining color and size using inline style.

Using Embedded Style

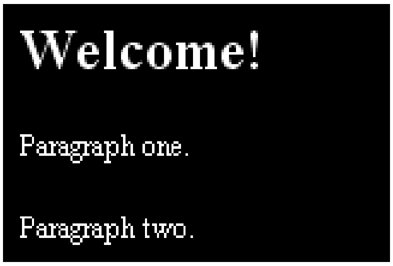
Embedded style controls only the document in which it is embedded. As with inline style, this defeats the purpose of being able to apply styles site-wide. However, there are good uses for embedded style. One would be if that document is the only document in the site that takes those specific styles. Another is workflow related. I like to use embedded style while working on a design because it's all in the same document. This way, I don't have to switch between applications or windows to accomplish my tasks. Because the style rules are the same, I can simply cut out the final styles from the embedded sheet and link them, which you'll see how to do in just a bit.

Embedded style is added to the `head` portion of the document, within the `style` element, and it uses the required `type` attribute (see Example 7-1).

EXAMPLE 7.1 An HTML document snippet describing embedded style

```
<head>
<title>working with style</title>
          <style type="text/css">
          body {background-color: black; color: white;}
          h1 {font-size: 24px;}
          p {font-size: 12px;}
          </style>
</head>
<body>
<h1>Welcome!</h1>
<p>Paragraph one.</p>
<p>Paragraph two.</p>
</body>
```

Figure 7-3 shows the results.


Welcome!

Paragraph one.

Paragraph two.

FIGURE 7-3 Notice how the color from the body is inherited by all its children.

Creating a Linked Style Sheet

To truly tap into the power of CSS, you'll be using linked style sheets the majority of the time. A linked style sheet is a separate text file into which you place all your CSS rules (but *not* any HTML) and is named using the `.css` suffix. You then link any HTML file you want to have affected by that style sheet to the sheet using the link element in the head portion of the document.

Example 7-2 shows a style sheet ready for linking. In it, I've provided a range of style rules and then saved the file to my local folder, naming the file `styles.css`.

EXAMPLE 7-2 A style sheet ready for linking

```
body {
          background-color: #999;
          color: black;
          }
h1 {
          font-family: Verdana;
          font-size: 24px;
          color: #ccc;
          }
p {
          font-family: Georgia;
          font-size: 12px;
          color: white;
}
```

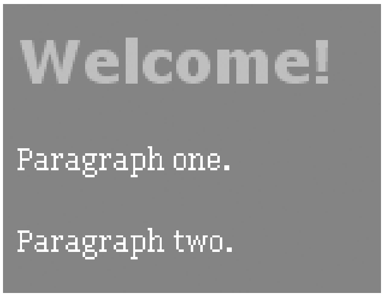
In Example 7-3, you'll find the complete HTML along with the required link to the style sheet within the same directory.

EXAMPLE 7-3 The HTML for the style sheet

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>working with style</title>

<link rel="stylesheet" type="text/css" href="styles.css" media="all" />

</head>
<body>

<h1>Welcome!</h1>
```

```
<p>Paragraph one.</p>

<p>Paragraph two.</p>

</body>
</html>
```

The results can be seen in Figure 7-4.


Welcome!

Paragraph one.

Paragraph two.

FIGURE 7-4 Results from a linked style sheet.

Of course, you can link as many documents you want to this style sheet using the `link` element.

You'll note several attributes in use with the `link` element, as follows:

- `rel`—This is the relationship attribute, and it describes the relationship of the link. In this case, the relationship is with a primary style sheet, so the `stylesheet` value is given.
- `type`—As with the `style` element in embedded styles, you must define the type of language and format in use—in this case, `text/css`.
- `href`—This is the familiar reference attribute. In this case, I've identified only the file because both documents are in the same directory. You might want to put your style sheets into a different directory; just be sure your `href` info is accurate. You can also use absolute linking to directly link to a style sheet.
- `media`—The `media` attribute enables you to define different styles for different media. If you wanted to create a separate style sheet for this document that's for handheld devices only, you would link to that and use the `media="handheld"` attribute. Similarly, a `media="print"` attribute would send that style sheet only to print. In this case, the media is defined as `screen`. The default is `all`, so if you want the same styles to apply to all media, you can use that or simply leave out the `media` attribute.

As mentioned, you can link as many style sheets to the same document as you want.

Importing Style Sheets

Imported style sheets are a lot like linked style sheets, in that you create a separate file for the styles you want to import. Then you can either import those sheets into a primary style sheet that is then linked to the document, or import directly into the document.

Importing Directly into a Document

Importing into a document actually involves two types of style sheets: the separate style sheet that's to be imported (I'll call that `import.css`) and an embedded style sheet in the document. This is because importing isn't done with an element such as `link`; instead, the CSS directive `@import` is used (see Example 7-4).

EXAMPLE 7-4 Importing style with an embedded sheet

```
<head>
<head>
<title>working with style</title>
<style type="text/css">

@import url(import.css);

</style>
</head>
```

The style sheet `@import.css` will be imported directly into the document. Imagine the `style` element being filled with all the style rules within the `import.css` file—that's exactly what happens. So now the style is actually embedded in this file.

You can use this technique for as many documents as you want, but typically this technique is used primarily in workarounds. A number of browsers, particularly Netscape 4 versions, do not support the `@import` directive, yet they do support the `link` element. Because Netscape 4.*x* has limited support for CSS and you have to take care to send styles to it, separating out those styles that you don't want it to misinterpret and those styles you know it can support into linked and imported allows Netscape users to see some, but not all, styles. This is very effective as a workaround when you must support Netscape 4 versions.

Another workaround using the `@import` directive is to simply place all styles into the imported sheet. Then any browser that doesn't support the `@import` simply won't read the styles, and a plain, unstyled document gets sent to the browser instead.

QUANTUM LEAP: FLASH OF UNSTYLED CONTENT (FOUC)

If you are using the `@import` technique and have no `link` or `script` element in the head of your document, Internet Explorer will often display the unstyled content first and then redraw the page with the style. It's an annoying bug but is easily avoided by adding a `link` or `script` element to the head of the document. For more about FOUC, see http://www.bluerobot.com/web/css/fouc.asp.

Most of the time, you won't be using the `@import` in an embedded sheet unless you have a very specific reason to do so.

Importing Style into a Linked Style Sheet

Another use for the `@import` directive, and the real reason `@import` exists, is to be able to modularize your styles and then import them into the primary style sheet. Consider Figure 7-5.

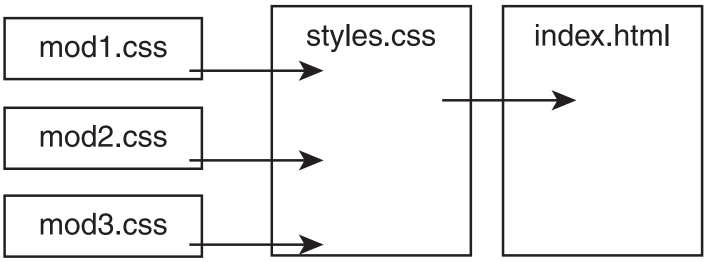


FIGURE 7-5 Importing styles into a main sheet.

Imagine that each module file (`mod1.css`, `mod2.css`, and `mod3.css`) contains styles specific to a feature or function within your site. As an example, you might have styles set to manage ads, styles specific to tables, and styles specific to forms. You could place these in separate module files and then import them into the `styles.css` file, which is then linked to `index.html`. The rationale behind this approach is that you could make modifications to the modules independently or cut them out easily when they are no longer needed. This technique is most effective when you have very large sites with lots of styles to manage.

QUANTUM LEAP: HACKING CSS

Although CSS is now in widespread use, hacks sometimes must be employed to achieve cross-browser consistency. You can use the modular import method described for your hacks. This way, as soon as you no longer need the hack, you can simply delete the imported file and the `@import` directive, removing the hack completely and getting your CSS as clean as possible. For more about hacking CSS, see my InformIT article "Strategies for Long-Term CSS Hack Management," at http://www.informit.com/articles/article.asp?p=170511.

Commenting and Formatting CSS

Just as you can add comments to your HTML files to describe sections, hide markup and content from the browser, or add directives to fellow document authors, you can comment your CSS documents. And just as HTML can be written with indentations or other personal formatting preferences, so can CSS.

Commenting CSS

CSS comments are different than HTML comments. CSS comments open with a forward slash and an asterisk, and close with an asterisk followed by a forward slash. Any content within that area is *not interpreted* by the browser (see Example 7-5).

EXAMPLE 7-5 Commenting CSS

```
/* global styles */

body {
           background-color: orange;
           font-family: Arial, Helvetica, sans-serif;
           color: white;
           }

/* layout styles */

#nav {
            position: absolute;
           top: 0;
           left: 0;
           width: 150px;
           }

/* hide this style and comment temporarily

.warning {
           color: red;
           }

John: please unhide the warning style when you're ready to launch */
```

Everything in bold will not be interpreted by the browser, but all the styles outside of comments will. As you can see, this can help you chunk your style sheets into logical groups, to aid both you and others to find given styles quickly. Additionally, you can hide styles you don't want for use later, and you can leave commentary for other people working with the style sheet.

You will sometimes see HTML comments surrounding CSS within an embedded sheet (see Example 7-6).

EXAMPLE 7-6 HTML comments to hide CSS

```
<head>
<title>working with style</title>
<style type="text/css">
<!--
          body {
                       background-color: #999;
                       color: black;
                       }
          h1 {
                       font-family: Verdana;
                       font-size: 24px;
                       color: #ccc;
                       }
          p {
                       font-family: Georgia;
                       font-size: 12px;
                       color: white;
                       }
 -->
</style>
</head>
```

In this case, the HTML comments are being used to hide the CSS from older browsers that do not interpret CSS. Many of those browsers would try to display the CSS rules in the browser window. Using HTML comments in this manner is still in widespread use today, although for contemporary browsers the technique is unnecessary.

Formatting CSS

You might have noticed that I've used two formatting approaches in this chapter (sneaky, aren't I?). The first is to follow the selector with the declaration, all on the same line:

```
body {background-color: #999; color: black;}
```

The other is to break up the rule:

```
body {
             background-color: #999;
             color: black;
             }
```

Either approach is correct; it's just a matter of personal preference. Many CSS designers are of the mindset that every bit and byte counts, so they opt for the first approach. Others argue that breaking up the rule makes it easier to find the styles you want to modify. Either way, as long as all the required syntax is intact, the formatting of your style sheet is a personal choice.

Time to Put Your Imagination to Work!

If you're thinking at this point that working with markup and CSS is no play, well, your frustration is well founded. It's imperative that you get the complexities down, and I assure you that if you've made it through thus far, you're grasping complex ideas.

But no doubt you want to put those ideas to work and really get a feel for how to use CSS to make things look good. After all, that's what I keep promising, right?

Fortunately, the next chapter sets you up for a little fun: putting your imagination to work.

You'll be using images and color to spruce up your documents, and exploring the fine control that CSS offers you when it comes to working with imagery and color in your designs.