25

CHAPTER

# Introducing Cascading Style Sheets

✦ ✦ ✦ ✦

**In This Chapter**

Why style sheets are needed

What style sheets can do

The cascading model

Examples of style sheets

A discussion of browser-compatibility issues

✦ ✦ ✦ ✦

Perhaps the most exciting part of HTML 4 is its complete support for cascading style sheets. Style sheets completely change the model of HTML for the better. This chapter explains why you need style sheets (even if you are still a little bit afraid of them) and what they can do. It also explains the cascading model of the style sheets and your choices for creating style information to associate with your page. You also get to see a few more examples of the basic meat-and-potatoes kind of style sheet you will get used to creating or modifying. Finally, you learn about browser-compatibility issues related to style sheets.

## Why Style Sheets Are Needed

The earlier chapters in this book alluded to all the great things style sheets can do so many times that, if you haven't flipped to this chapter to read ahead at least once, it's safe to say you are a disciplined individual. Once you understand why you need style sheets, you'll be hard-pressed to believe you ever lived without them.

On the Web, there is a lot of hype about a lot of things. Rarely does any product or technology live up to the hype it engenders. *Cascading style sheets* (CSS) are one of those rare technologies.

HTML 3.2, with all its built-in formatting elements and attributes, made it a nightmare to create pages and even worse to maintain them. Web developers managing sites with hundreds or even thousands of pages had to face this problem daily.

Style sheets directly address the problem of formatting information cluttering up your pages. With style sheets, all formatting information moves from the HTML document to a style sheet (a text file with a `.css` extension). Any page that wants to use that format simply links to the style sheet. Any changes made to the master style sheet are automatically reflected in all pages that link to the style sheet. Now that is real power!

## What Style Sheets Can Do

Style sheets can change the look of any element on your page. Element by element, you can define the way you want things to look in a style sheet. Your paragraphs will all take on the formatting you associate with the `P` element in your style sheet. If you want to define more than one look for an element, you can create classes of an element and assign the classes to the elements when you define them in the pages.

Say you have two types of paragraphs in your pages: normal and newspaper. The normal paragraphs should use a half-inch indent on the first line and leave quarter-inch of white space (padding) above them. The newspaper paragraphs should indent a quarter inch on the first line and leave two inches of white space on both the right and left sides. Style sheets can accomplish this easily. You, as the developer, simply define a regular `P` element in your style sheet to match the formatting you want your normal text paragraphs to have. Then you define another `P.newspaper` element (that is a `P` element with a `class` of newspaper) with the formatting you want your newspaper paragraphs to have. Whenever you come to a normal text paragraph, you use the regular `P` element. When you come to a newspaper paragraph, you use the `P` element with the `class` (an attribute) specified as "newspaper." What could be easier?

### Grouping elements

Remember in Chapters 16 and 17, when you learned about grouping elements with the `DIV` (for block elements) and the `SPAN` (for inline elements) elements? Style sheets can also do something HTML 3.2 can't do, even with all those ugly, deprecated presentation elements. If you wanted to group two paragraphs, give them a background color in common, but different from the rest of the page, and put a nice thick border around them, you'd have to use tables in the way tables were never intended to be used. With style sheets, you can define a `DIV` class in your style sheet, and by including the paragraphs in the `DIV` element with the class set to whatever you called it in your style sheet, all this formatting will appear on your page. If you ever want to change the background color for that class of the `DIV` element, you can do this one time in one place, in your style sheet, and the changes will be reflected everywhere you used that `DIV` class.

### Site face-lift

If nothing else you've read about style sheets thus far impresses you, this will. Imagine giving your site a complete face-lift in a morning. This might not sound impressive if your site is still only a handful of pages, but if your site has matured into the dozens, hundreds, or thousands of pages that many Webmasters have to maintain, this is truly revolutionary.

With style sheets, all the formatting information resides in one place. If you change the definitions in that one place, you can affect literally thousands of pages — however many link to the style sheet — in one fell swoop.

One day visitors come to your site and find a white background, left-justified text, blue headings, black text in Times Roman font. The next day they find a light blue background with the embossed logo of your company subtly woven into the background, navy headings in Verdana font, dark grey text in Verdana font, and the entire page is fully justified! That'll grab them!

### Delegating page assembly without sacrificing design control

Wouldn't you love to implement a distributed team model to facilitate Web development in your organization? After all, the information you use in the pages comes from all over the organization. Why not just have them put all the material into Web pages directly? Before style sheets, about a million good reasons existed. Every which way a person can design a Web page is another reason not to let people who aren't part of the Web design team assemble their own pages. Who knows what they would look like?

But with style sheets, if you can train people to use the `H1`–`H6` elements and to markup their paragraphs with the `P` element, then most of the work can be done by a distributed team of people who are not necessarily Web designers. You can go in later, check their work, and add any tables. Most important of all, if they link to your style sheet, even if they don't do things exactly as you would have, their pages will look pretty much like the rest of the site.

## The Cascading Model

So, what is this cascading business about? The cascading model depends on the idea that you can specify style sheets at more than one level. The lowest-level style sheet takes precedence. For example, say your company has a corporate style sheet, called `corporate.css`. Your department might have its own style sheet called `hr.css`. Finally, you might want to create a specific style for a class of the `DIV` element, right in the HTML document. You could even define styles at the element level in the `BODY` of your page, but that isn't included in this example. How does this work?

In your HTML `HEAD` element, you would create a link to the corporate style sheet, then another link to the department style sheet. Following those, also in the `HEAD` element, you would define a style for the `DIV` element, using the `STYLE` element. Now, say the corporate style sheet had the following styles defined (to name a few):

- background tan
- text color brown
- font face Helvetica
- H1 is 22pt, H2 is 18pt, H3 is 16pt

Say the department had the following style defined (to name one):

- H2 is 20pt, H3 is 18pt, H4 is 16pt

And in the `STYLE` element in the `HEAD` element of your page, you define a `DIV` element with a class of highlight with a background of white and text color of black, with a black border.

What would show up in your document? Everything from the corporate style sheet that is not also defined in the department style sheet, and everything in the department style sheet that is not defined in the `STYLE` element, and everything in the `STYLE` element that is not defined at the element level. The result would be:

- background tan
- text color brown
- font face Helvetica
- H2 is 20pt, H3 is 18pt, H4 is 16pt
- DIV `class of highlight with a background color of white, text color of black, with a black border`

## Style Sheet Examples

Even though you may find a style sheet definition intimidating at first, don't fret; lots of great tools are available to help you create a syntactically correct style sheet. Refer to Chapter 8 for a list. A number of Web-based resources also exist for creating style sheets. Unlike pages, you won't be creating a lot of style sheets. You'll create one you like, use it for the majority of your pages, and modify it occasionally.

If worse comes to worse, you could always start with one of the style sheets on the CD-ROM and modify it to meet your needs.

Here is an example style sheet. Explanations of things to notice are listed after the entire style sheet definition. Your style sheet doesn't need to contain all these rules (definitions of all these elements). You might just define a style for the body (one rule) and be content with the way your browser renders everything else.

```
BODY {
       font-family: "Book Antiqua", "Times New Roman", serif;
       color: #000040;
       background: #FFFF9F;
       padding: 1in;
}
A:LINK {
       color: #FF00FF;
}
A:VISITED {
       color: #808080;
}
BLOCKQUOTE {
       margin-left: 1in;
       margin-right: 1in;
}
HR {
       height: 2pt;
}
P {
       text-indent: .5in;
}
P.double {
       text-indent: .5in;
       line-height: 24pt;
}
SPAN.highlight {
       color: #000080;
       background: #FFFF00;
}
```

The only difficult part about creating a style sheet is remembering the property names. For example, to indent a paragraph a half inch, you need to know that the property is `text-indent`, not `paragraph-indent`, or `indent-text`. If you can find a reasonably priced tool you like for creating style sheets (the previous one took advantage of the style sheet wizard in HomeSite 4.0), then you don't have to worry about the syntax; you can put your energy into creating the style sheet you want.

In the previous example, notice each element name (called a *selector* in style sheets) is followed by a curly brace ({), and then comes a property followed by a colon, and then a value (called a *declaration*). With attributes, the syntax is

```
attribute = "value"
```

With style sheets, the syntax for a declaration is

```
property: value;
```

or

```
property: value, value, value;
```

Notice no quotes are around the value of the property in the style sheet (unless a single value with more than one word in it, such as a font). Don't worry about learning all these rules right now. You get a more thorough explanation in the next chapter. Right now you should notice that for each element, you can define several different formatting features. The last two elements on the list also have class definitions: `P.double` and `SPAN.highlight`. Those would be used in your document as follows:

```
<P class="double">
```

and

```
<SPAN class="highlight">
```

What does this style sheet do?

1. It formats all the `BODY` elements with a light yellow (#000040) background and dark gray (#FFFF9F) text using a font of Book Antiqua or, if that font is not available, Times New Roman or, if that is not available, any serif font, and it sets padding on the page of one inch.
2. It formats not-yet-visited links (`A:LINK`) to fuchsia (#FF00FF).
3. It formats visited links (`A:VISITED`) to gray (#808080).
4. It formats blockquotes to have one-inch left and right margins, which the browser would add to the one-inch padding already defined in the `BODY` element.
5. It formats all horizontal rules to be two pixels tall.
6. It gives paragraphs a half-inch indentation on the first line.
7. It creates a paragraph class called double (`P.double`), which has the same half-inch indentation on the first line, but also includes double-spacing of text (line height: 24pt).
8. It also creates a `SPAN` class called highlight (`SPAN.highlight`), which will have a text color of navy blue (#000080) and a background color of bright yellow (#FFFF00).

## Browser Compatibility Issues

What about actually using style sheets in browsers? Is all this stuff supported? Unfortunately, no. Internet Explorer 5 does the best job of implementing style sheets. Netscape 4.x is significantly less supportive of style sheets, but does support many features of style sheets. Internet Explorer 3 and 4 also do a respectable job of supporting style sheets.

What does this mean for you? If you are creating pages for an intranet and you know people using your pages will have IE 5, or Navigator 4, then you can test your pages on the platform you know people will be using, and only use the styles supported on that platform. However, if you are creating pages for the Internet, where you can't predict what kind of browser visitors will be using, you have three choices, recapped from Chapter 2:

- Don't use style sheets. Continue to design your pages with the deprecated HTML 3.2 elements and attributes and look forward to the day when browsers that support style sheets are more widely used.
- Maintain two versions of your site. Believe it or not, this is what many large, high-profile sites do. In Chapter 48, you will learn how to test to see which browser a visitor is using so that you can automatically load the appropriate version of the page.
- Use style sheets and to heck with the luddites who haven't upgraded. They will, after all, be able to see your site; they just won't get the same design look that they should.

## From Here

Cross-Reference

Jump to Chapter 48 and learn about using JavaScript to test to see which version of which browser a visitor is using.

Proceed to Chapter 26 and learn the syntax of style sheets.

## Summary

In this chapter you learned just why style sheets are so valuable. You learned what they can do that couldn't previously be done. You learned about the cascading model. You got a chance to analyze an example of a style sheet. You also learned enough about browser compatibility issues to understand that making a decision about using style sheets requires some analysis.

✦ ✦ ✦