CHAPTER
6

Style Sheets

Last chapter, you learned XHTML's dirty little secret—it doesn't have much formatting muscle. If you want your Web pages to look sharp, you need to add style sheets into the mix.

A style sheet is a document filled with formatting rules. Browsers read these rules and apply them when they display Web pages. For example, a style sheet rule might say, "Make all the headings on this site bold and fuchsia, and draw a box around each one."

There are several reasons why you want to put formatting instructions in a style sheet instead of embedding them in a Web page. The most obvious is *reuse*. For example, thanks to style sheets, you can create a single rule to format level-3 headings, and every level-3 heading on every Web page on your site will reflect that rule. The second reason is that style sheets help you write tidy, readable, and manageable XHTML files. Because style sheets handle all your site's formatting, your XHTML document doesn't need to. All it needs to do is organize your pages into logical sections. (For a recap of the difference between structuring and formatting a Web page, refer to page 110.) And finally, style sheets give you more extensive formatting choices than those in XHTML alone. Using style sheets, you can control colors, borders, margins, alignment, and (to a limited degree) fonts.

You'll use style sheets throughout this book. In this chapter, you'll learn the basics of style sheets, and see how you can use them to create a variety of visual effects.

Style Sheet Basics

Style sheets use a standard that's officially known as *CSS* (Cascading Style Sheets). CSS is a system for defining style *rules*. These rules change the appearance of the elements in a Web page, tweaking details like color, font, size, borders, and placement.

When you use CSS in a Web page, a browser reads both the page's XHTML file and the style sheet rules. It then uses those rules to format the page. Figure 6-1 shows the process.

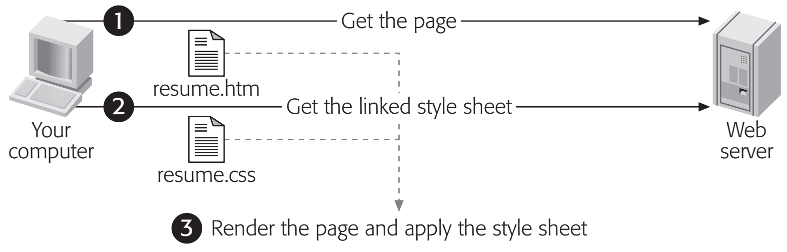

1
Get the page
resume.htm
2
Get the linked style sheet
Your computer
resume.css
Web server
3 Render the page and apply the style sheet

Figure 6-1: *When you go to a Web page that uses a style sheet, the following things happen. 1) Your browser requests the XHTML page from a Web server. 2) The browser finds an instruction in the XHTML page indicating that the page uses a style sheet. The browser then grabs that style sheet with a separate request. 3) The browser chews through the XHTML in the Web page, and uses the rules in the style sheet to adjust the page's appearance.*

This system gives Web weavers the best of both worlds—a rich way to format pages and a way to avoid mucking up your XHTML document beyond recognition. In an ideal world, the XHTML document describes only the structure of your Web page (what's a header, what's a paragraph, what's a list, and so on), and the style sheet formats that Web page to give it its hot look.

The Three Types of Styles

Before you learn how to write CSS rules, you first have to think about *where* you're going to place those instructions. CSS gives you three ways to apply style sheets to a Web page:

- An **external style sheet** is one that's stored in a separate file. This is the most powerful approach, because it completely separates formatting rules from your XHTML pages. It also gives you an easy way to apply the same rules to many pages.
- An **internal style sheet** is embedded inside an XHTML document (it goes right inside the <head> section). You still have the benefit of separating the style information from the XHTML, and if you really want, you can cut and paste the embedded style sheet from one page to another (although it gets difficult to

keep all those copies synchronized if you make changes later on). You use an internal style sheet if you want to give someone a complete Web page in a single file—for example, if you email someone your home page. You might also use an internal style sheet if you know that you aren't going to use any of its style rules on another page.

- An **inline style** is a way to insert style sheet language directly inside the start tag of an XHTML element. At first glance, this sounds suspicious. You already learned that it's a bad idea to embed formatting instructions inside a Web page, because formatting details tend to be long and unwieldy. That's true, but you might occasionally use the inline style approach to apply one-time formatting in a hurry. It's not all that clean or structured, but it does work.

These choices give you the flexibility to either follow the CSS philosophy wholeheartedly (with external style sheets), or to use the occasional compromise (with internal style sheets or inline styles). Because style sheet language is always the same, even if you use a "lazier" approach like internal style sheets, you can always cut-and-paste your way to an external style sheet when you're ready to get more structured.

UP TO SPEED

The "Other Way" to Format a Web Page

Style sheets aren't the only way to format a Web page—they're just the most capable tool. You've also got a few formatting options built right into the XHTML elements you learned about in Chapter 5. For example, you can change a page's background color or center text without touching a style sheet. For the most part, this book doesn't use these formatting options, for several good reasons:

- **They're patchy and incomplete.** Many style features, like paragraph indenting and borders, are missing—there are no XHTML formatting elements to achieve these effects. Even worse, the model isn't consistent—for example, you might be able to line up text in one type of element, but not the text contained in another type of element. This makes the model difficult to learn and remember.
- **They work only in XHTML 1.0 transitional**. As you learned earlier (page 30), transitional XHTML includes features that the official rulemakers of the XHTML standard—that'd be the good people who work at World Wide Web Consortium (W3C)—consider obsolete. If you use these old-school formatting features, your hard-core Web designer friends won't sit with you at restaurants.
- **They don't let you easily reuse formatting changes.** So after you reformat one page, you need to start all over again to fix the next page. And so on, and so on, and so on.
- **Why learn something you don't need?** Considering that style sheets have so much power and flexibility, and now that virtually every browser around—old and new—supports style sheets, it doesn't make sense to waste time with something you'll only outgrow.

Browser Support for CSS

Before you embrace style sheets, you need to make sure they work on all the browsers your site visitors use. That's not as easy to figure out as it should be. The first problem is that there's actually more than one version of the CSS standard—there's the original CSS1, the slightly improved CSS2, the corrected CSS2.1, and the still-developing CSS3. But the real problem is that browsers don't necessarily support the entire CSS standard no matter what version it is. And when they do, they don't always support it in exactly the same way. The discrepancies range from minor to troubling. As a result, in this book you'll focus on CSS1 and CSS2 properties known to be well-supported on all the major browsers (see below). You'll steer clear of CSS3, which is still too new to be useful. That said, don't forget to test your pages in a variety of browsers to be sure they look right.

As a basic rule of thumb, you can count on good all-around CSS1 support in Netscape Navigator 6, Internet Explorer 5, Opera 3.6, and any version of Firefox, Safari, or Google Chrome. In later versions of these browsers, CSS support gets more consistent and more comprehensive, with added features from CSS2 and even CSS3.

People who've used the Web for a few years may still remember an earlier generation of browsers, namely Netscape 4.x and Internet Explorer 4.x. Both of these browsers are unreliable when it comes to the fancier features of CSS. However, you're unlikely to run into them outside of a museum. If you're in doubt, take a look at some recent statistics to see which browsers people use online (page 13).

If you're still concerned about whether a specific browser version supports a specific CSS feature, see the box on page below.

FREQUENTLY ASKED QUESTION

A Browser Compatibility Reference

How can I tell if a particular browser supports a CSS feature?

If you're a hard-core Web maven, you may be interested in one of the Web browser compatibility charts for CSS available on the Web. Two good resources are *www.webdevout.net/browser-support-css* and *www.quirksmode.org/css/contents.html*. These charts specify which browsers support which CSS features.

But chart-reader beware: These tables include many rarely used or new and poorly supported features. For example, you probably don't care that virtually no browser supports the *pitch-range* property, used in conjunction with text-reading devices. Unfortunately, the CSS charts can cause panic in those who don't know the standards. However, they can be handy if you need to check out support for a potentially risky feature.

The Anatomy of a Rule

Style sheets contain just one thing: *rules*. Each rule is a formatting instruction that applies to a part of your Web page. A style sheet can contain a single rule, or it can hold dozens (or even hundreds) of them.

Here's a simple rule that tells a browser to display all <h1> headings in blue:

```
h1 { color: blue }
```

CSS rules don't look like anything you've seen in XHTML markup, but you'll have no trouble with them once you realize that three ingredients make up every rule: *selectors*, *properties*, and *values*. Here's the format that every rule follows:

```
selector { property: value }
```

And here's what each part means:

- The **selector** identifies the type of content you want to format. A browser then hunts down all the parts of a Web page that match the selector. For now, you'll concentrate on selectors that match every occurrence of a specific page element, like a heading. But later in this chapter (page 171), you'll learn to create more sophisticated selectors that act only on specific sections of your page.
- The **property** identifies the type of formatting you want to apply. Here's where you choose whether you want to change colors, fonts, alignment, or something else.
- The **value** sets a value for the property defined above. This is where you bring it all home. For example, if your property is color, the value could be light blue or a queasy green.

Of course, it's rarely enough to format just one property. Usually, you want to format several properties at the same time. You can do this with style sheets by creating a rule like this:

```
h1 { text-align: center;
  color: black; }
```

This example changes the color of *and* centers the text inside an <h1> element. That's why style rules use the funny curly braces { and }—so you can put as many formatting instructions inside them as you want. You separate one property from the next using a semicolon (;). It's up to you whether to include a semicolon at the end of the last property. Although it's not necessary, Web-heads often do so to make it easy to add another property onto the end of a rule when needed.

Note: As in an XHTML file, CSS files let you use spacing and line breaks pretty much wherever you want. However, people often put each formatting instruction on a separate line (as in the example above) to make style sheets easy to read.

Conversely, you might want to create a single formatting instruction that affects several different elements. For example, imagine you want to make sure that the first three heading levels, h1 to h3, all have blue letters. Rather than writing three separate rules, you can create a selector that includes all three elements, separated by commas. Here's an example:

```
h1, h2, h3 { color: blue }
```

Believe it or not, selectors, properties, and values are the essence of CSS. Once you understand these three ingredients, you're on your way to style sheet expertise.

Here are a few side effects of the style sheet system that you might not yet realize:

- A single rule can format a whole bunch of XHTML. When you implement a rule for the kind of selectors listed above (called *type selectors*), that rule applies to every one of those elements. So when you specify blue h1 headings as in the example above, every h1 element in your page becomes blue.
- It's up to you to decide how much of your content you want to format. You can fine-tune every XHTML element on your Web page, or you can write rules that affect only a single element, using the techniques you'll find at the end of this chapter (page 171).
- You can create two different rules for the same element. For example, you could create a rule that changes the font of every heading level (<h1>, <h2>, <h3>, and so on), and then add another rule that changes the color of just <h1> elements. Just make sure you don't try to set the same property multiple times with conflicting values, or the results will be difficult to predict.
- Some elements have built-in style rules. For example, browsers always display text that appears in a <b> element as boldfaced, even when the style sheet doesn't include a rule to do so. Similarly, browsers display text in an <h1> heading in a large font, with no style sheet rule necessary. But you can override any or all of these built-in rules using custom style rules. For example, you could explicitly set the font size of an <h1> heading so that it appears *smaller* than normal text. Similarly, you can take the underline off of a link, make the <b> element italicize text instead of bolding it, and so on.

Don't worry about memorizing the kind of properties and values you can specify. Later in this chapter, after you see how style sheets work, you'll get acquainted with all the formatting instructions you can use.

Applying a Style Sheet

Now it's time to see style sheets in action. Before you go any further, dig up the *resume.htm* file you worked on in Chapter 2 (it's also available from the Missing CD page at *www.missingmanuals.com*). You'll use it to test out a new style sheet. Follow these steps:

1. **First, create the style sheet. You do this by creating a new file in any text editor, like Notepad or TextEdit.**

   Creating a style sheet is no different from creating an XHTML page—it's all text. Many XHTML editors have built-in support for style sheets (see the box on page 145 for more information).

2. **Type the following rule into your style sheet:**

   ```
   h1 { color: fuchsia }
   ```

   This rule instructs your browser to display all <h1> elements in bright fuchsia lettering.

3. **Save the style sheet with the name *resume.css*.**

   Like an XHTML document, a style sheet can have just about any file name. However, as a matter of convention, style sheets almost always use the extension .css. For this example, make sure you save the style sheet in the same folder as your XHTML page.

4. **Next, open the *resume.htm* file.**

   If you don't have the *resume.htm* file handy, you can test this style sheet with any XHTML file that has at least one <h1> element.

5. **Add the <link> element to your XHTML file.**

   The <link> element points your browser to the style sheet you wrote for your pages. You have to place the <link> element in the <head> section of your XHTML page. Here's the revised <head> section of *resume.htm* with the <link> element added:

   ```
   <head>
     <link rel="stylesheet" type="text/css" href="resume.css" />
     <title>Hire Me!</title>
   </head>
   ```

   The link element includes three details. The *rel* attribute indicates that the link points to a style sheet. The *type* attribute describes how you encoded the document. You should copy both these attributes exactly as shown above, as they never change. The *href* attribute is the important bit—it identifies the location of your style sheet ("href" stands for hypertext reference). Assuming you put your style sheet in the same folder as your XHTML file, all you need to supply is the file name. (If you put these two files in different folders, you need to specify the location of the .css file using the file path notation system you'll learn about on page 171.)

6. **Save the XHTML file, and open it in a browser.**

   Here's what happens. Your browser begins processing the XHTML document and finds the <link> element, which tells it to find an associated style sheet and apply all its rules. The browser then reads the first (and only, in this case) rule in the style sheet. To apply this rule, it starts by analyzing the selector, which targets all <h1> elements. Then it finds all the <h1> elements on the XHTML page and applies the fuchsia formatting.

The style sheet in this example isn't terribly impressive. In fact, it probably seems like a lot of work to simply get a pink heading. However, once you've got this basic model in place, you can quickly take advantage of it. For example, you could edit the style sheet to change the font of your *resume.htm* headings. Or, you could add rules to format other parts of the document. Simply save the new style sheet, and then refresh the Web page to see the effect of the changed or added rules.

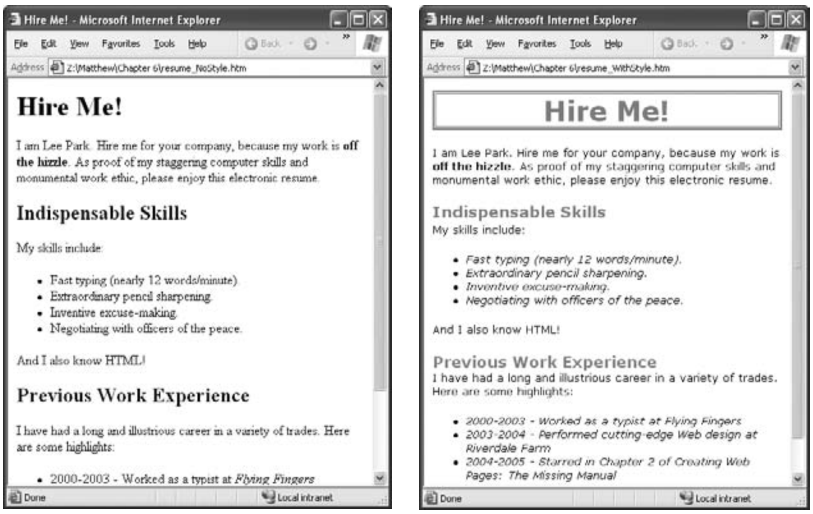
To see this at work, change the *resume.css* style sheet so that it has these rules:

```
body {
  font-family: Verdana,Arial,sans-serif;
  font-size: 83%;
}

h1 {
  border-style: double;
  color: fuchsia;
  text-align: center;
}

h2 {
  color: fuchsia;
  margin-bottom: 0px;
  font-size: 100%;
}

li {
  font-style: italic;
}

p {
   margin-top: 2px;
}
```

These rules change the font for the entire document (through the <body> element rule), tweak the two heading levels, italicize the list items, and shave off some of the spacing between paragraphs. Although you won't recognize all these rules at first, the overall model (content in the Web page, formatting in the style sheet) is the same as in the earlier resume example. Figure 6-2 shows the result.

Internal style sheets

The *resume.css* example demonstrates an external style sheet. External style sheets are everybody's favorite way to use CSS because they let you link a single lovingly crafted style sheet to as many Web pages as you want. However, sometimes you're not working on an entire Web site, and you'd be happy with a solution that's a little less ambitious.


Hire Me! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address Z:\Matthew\Chapter 6\resume_NoStyle.htm
Done
Local intranet

Hire Me!

I am Lee Park. Hire me for your company, because my work is **off the hizzle**. As proof of my staggering computer skills and monumental work ethic, please enjoy this electronic resume.

Indispensable Skills

My skills include:

- Fast typing (nearly 12 words/minute).
- Extraordinary pencil sharpening.
- Inventive excuse-making.
- Negotiating with officers of the peace.

And I also know HTML!

Previous Work Experience

I have had a long and illustrious career in a variety of trades. Here are some highlights:

- 2000-2003 - Worked as a typist at *Flying Fingers*

Hire Me! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address Z:\Matthew\Chapter 6\resume_WithStyle.htm
Done
Local intranet

Hire Me!

I am Lee Park. Hire me for your company, because my work is **off the hizzle**. As proof of my staggering computer skills and monumental work ethic, please enjoy this electronic resume.

Indispensable Skills

My skills include:

- *Fast typing (nearly 12 words/minute).*
- *Extraordinary pencil sharpening.*
- *Inventive excuse-making.*
- *Negotiating with officers of the peace.*

And I also know HTML!

Previous Work Experience

I have had a long and illustrious career in a variety of trades. Here are some highlights:

- *2000-2003 - Worked as a typist at Flying Fingers*
- *2003-2004 - Performed cutting-edge Web design at Riverdale Farm*
- *2004-2005 - Starred in Chapter 2 of Creating Web Pages: The Missing Manual*

Figure 6-2:
Left: By now, you can recognize a plain-vanilla Web page.

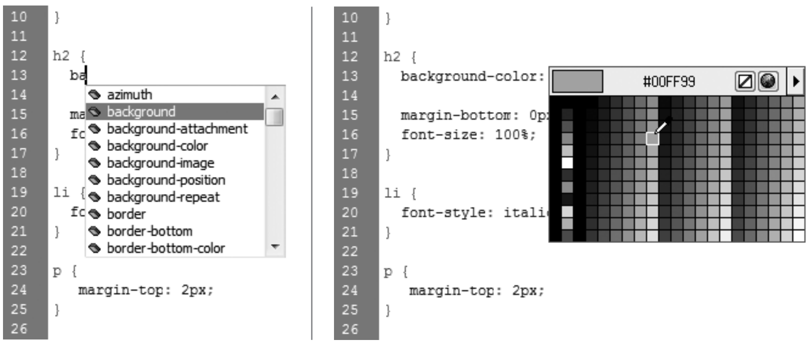
Right: A style sheet revamps the entire page.

GEM IN THE ROUGH

Creating Style Sheets with Web Page Editors

Some Web Page editors, like Dreamweaver and Expression Web, have handy features for editing style sheets. To try them out, start by opening an existing style sheet or creating a new one. To create a style sheet in Expression Web, choose File → New → CSS. To create a style sheet in Dreamweaver, choose File → New, pick CSS in the Page Type list, and then click Create.

So far, you won't see anything to get excited about. But life gets interesting when you start to *edit* your style sheet. As you type, your Web page editor pops up a list of possible style properties and values (see Figure 6-3). If you dig deeper, you'll find that both Web editors have windows that let you build styles by pointing and clicking, as well as convenient shortcuts for applying styles to your Web page elements.



```
}

h2 {
  ba
  ma
  fc
}

li {
  fc
}

p {
   margin-top: 2px;
}
```

```
}

h2 {
  background-color:

  margin-bottom: 0p
  font-size: 100%;
}

li {
  font-style: itali
}

p {
   margin-top: 2px;
}
```

Figure 6-3:
As you edit a style sheet in Dreamweaver, it pops up lists of possible style properties (left) and property values. If you're dealing with colors, you even get this handy color picker (right), which translates colors codes in your style sheet into the actual color and displays the results. It's a great help for foggy memories, and saves more than a few keystrokes.

An internal style sheet is one that, rather than being linked, is *embedded* in the <head> area of your Web page. Yes, it bulks up your pages and forces you to give each page a separate style sheet. But sometimes the convenience of having just one file that contains your page *and* its style rules makes this approach worthwhile.

To change the earlier example so that it uses an internal style sheet, remove the <link> element from your XHTML markup and add the style rules in a <style> element inside the <head> section of the page, as shown here:

```
<head>
  <title>Hire Me!</title>
  <style type="text/css">
     h1 { color: fuchsia }
  </style>
</head>
```

Inline styles

If you want to avoid writing a style sheet altogether, you can use yet another approach. Inline styles let you insert the property and value portion of a style sheet rule right into the start tag for an XHTML element. You don't need to specify the selector because browsers understand that you want to format only the element where you add the rule.

Here's how you use an inline style to format a single heading:

```
<h1 style="color: fuchsia">Hire Me!</h1>
```

The rule above affects only the h1 element where you added it; any other <h1> headings in the page are unchanged.

Unlike internal and external style sheets, inline styles don't require the *type="text/css"* attribute, which tells browsers that you're using CSS as your style language. For the most part, this missing detail is harmless, because every browser ever created assumes you're using CSS. However, to meet the strict rules of XHTML, you really should specify your choice of style language by adding the following <meta> element to the <head> section of your Web page:

```
<meta http-equiv="Content-Style-Type" content="text/css" />
```

You can pop this element into place right after the <title> element.

Inline styles may seem appealing at first, because they're clear and straightforward. You define the formatting information exactly where you want to use it. But if you try to format a whole Web page this way, you'll realize why Web developers go easy on this technique. Quite simply, the average CSS formatting rule is long. If you need to put it alongside your content and copy it each time you use the element, you quickly end up with a Web page that's mangled beyond all recognition. For example, consider a more complex heading that needs several style rules:

```
<h1 style="border-style: double; color: fuchsia; text-align: center">Hire
Me!</h1>
```

Even if this happens only once in a document, it's already becoming a loose and baggy monstrosity. So try to avoid inline styles if you can.

WORD TO THE WISE

Boosting Style Sheet Speed

External style sheets are more efficient on Web sites because of the way browsers use *caching*. Caching is a performance-improving technique where browsers store a copy of some downloaded information on your computer so they don't need to download it again.

When a browser loads a Web page that links to a style sheet, it makes a separate request for that style sheet, as shown back in Figure 6-1. However, if the browser loads another page that uses the *same* style sheet, it's intelligent enough to realize that it already has the right .css file on hand. As a result, it doesn't make the request. Instead, it uses the cached copy of the style sheet, which makes the Web page load a little bit faster. (Of course, browsers only cache things for so long. If you go to the same site tomorrow, the browser will have to re-request the style sheet.)

If you embed the style sheet in each of your Web pages, the browser always downloads the full page with the duplicate copy of the style sheet. It has no way of knowing that you're using the same set of rules over and over again. Although this probably won't make a huge difference in page-download time, it could start to add up for a Web site with lots of pages. Speed is just one more reason Web veterans prefer external style sheets.

The Cascade

By now, you might be wondering what the "cascading" part of "cascading style sheets" means. It refers to the way browsers decide which rules take precedence when you've got multiple sets of rules.

For example, if an external style sheet indicates that <h1> headings should have blue letters, and then you apply bold formatting with an inline style, you'll end up with the sum of both changes: a blue-lettered, bold-faced heading. It's not as clear what happens if the rules conflict, however. For example, if one rule specifies blue text while another mandates red, which color setting wins?

To determine the answer, you need to consult the following list to find out which rule has highest priority. This list indicates the steps a browser follows when applying styles. The steps toward the bottom are the most powerful: the browser implements them after it applies the steps at the top, and they override any earlier formatting.

1. Browser's standard settings
2. External style sheet
3. Internal style sheet (inside the <head> element)
4. Inline style (inside an XHTML element)

So, if an external style sheet conflicts with an internal style sheet, the internal style sheet wins.

Based on this, you might think that you can use this cascading behavior to your advantage by defining general rules in external style sheets, and then overriding them with the occasional exception using inline styles. In fact, you can, but there's a much better option. Rather than format individual elements with inline style properties, you can use class selectors to format specific parts of a page (see page 171 for details), as you'll see later in this chapter.

Note: The "cascading" in cascading style sheets is a little misleading, because in most cases you won't use more than one type of style sheet (for the simple reason that it can quickly get confusing). Most Web *artistes* favor external style sheets primarily and exclusively.

Inheritance

Along with the idea of cascading styles, there's another closely related concept—style *inheritance*. To understand style inheritance, you need to remember that in XHTML documents, one element can contain other elements. Remember the unordered list element (<ul>)? It contained list item elements (<li>). Similarly, a <p> paragraph element can contain character formatting elements like <b> and <i>, and the <body> element contains the whole document.

Thanks to inheritance, when you apply formatting instructions to an element that contains *other* elements, that formatting rule applies to *every one of those other elements*. For example, if you set a <body> element to the font Verdana (as in the résumé style sheet shown earlier), every element inside that <body> element, including all the headings, paragraphs, lists, and so on, gets the Verdana font.

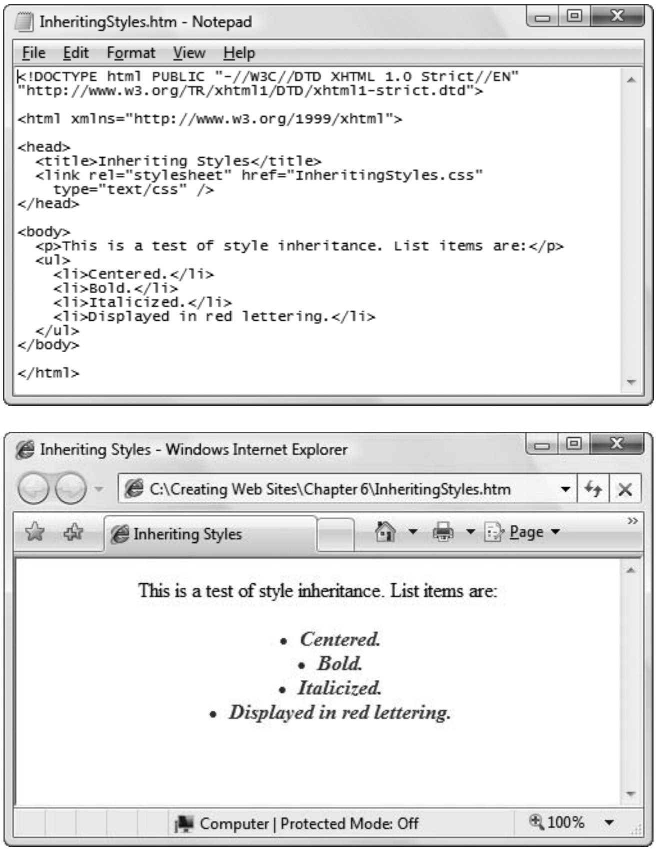
Note: Elements inherit most, but not all, style properties. For example, elements never inherit margin settings from another element. Look for the "Can Be Inherited" column in each table in this chapter to see whether a property setting can be passed from one element to another through inheritance.

However, there's a trick. Sometimes, formatting rules may overlap. In such a case, the most specific rule—that is, the one hierarchically closest to the element—wins. For example, settings you specify for an <h1> element will override settings you specified for a <body> element for all level-1 headings. Or consider this style sheet:

```
body {
  color: black;
  text-align: center;
}

ul {
  color: fuschia;
  font-style: italic;
}
```

```
li {
  color: red;
  font-weight: bold;
}
```

These rules overlap. In a typical document (see Figure 6-4) you put an <li> (list item) inside a list element like <ul>, which in turn exists inside the <body> element. In this case, the text for each item in the list will be red, because the <li> rule overrides the <ul> and <body> rules that kick in first.


InheritingStyles.htm - Notepad
File Edit Format View Help

```
<!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">

<head>
  <title>Inheriting Styles</title>
  <link rel="stylesheet" href="InheritingStyles.css"
    type="text/css" />
</head>

<body>
  <p>This is a test of style inheritance. List items are:</p>
  <ul>
    <li>Centered.</li>
    <li>Bold.</li>
    <li>Italicized.</li>
    <li>Displayed in red lettering.</li>
  </ul>
</body>

</html>
```

Inheriting Styles - Windows Internet Explorer
C:\Creating Web Sites\Chapter 6\InheritingStyles.htm
Inheriting Styles
Page
Computer | Protected Mode: Off
100%

This is a test of style inheritance. List items are:

- *Centered.*
- *Bold.*
- *Italicized.*
- *Displayed in red lettering.*

Figure 6-4:
When rules collide, the most specific element wins. In this example, that means your browser will display these list items in red, because the rule for the <li> element overrides the inherited properties from the <ul> and <body> elements. However, elements retain the style of an inherited rule if it doesn't conflict with another rule. In this example, that means the <li> element gets italics and center alignment through inheritance.

Crafty style sheet designers can use this behavior to their advantage. For example, you might apply a font to the <body> element so that everything in your Web page—headings, paragraph text, lists, and so on—has the same font. Then, you can judiciously override this font for a few elements by applying element-specific formatting rules.

Tip: Although you probably won't see cascading styles in action very often, you'll almost certainly use style inheritance.

Now that you've learned how style sheets work, you're ready to start with the hard part—learning about the dozens of different formatting properties you can change. The following sections group the key style properties into categories.

Note: In this chapter, you won't learn about every CSS property. For example, there are some properties that apply primarily to pictures and layout. You'll learn about those properties in later chapters.

Colors

It isn't difficult to inject some color into your Web pages. Style sheet rules have two color-related properties, listed in Table 6-1. You'll learn about the types of values you can use when setting colors (color names, color codes, and RGB values) in the following sections.

Table 6-1. *Color properties*

Property	Description	Common Values	Can Be Inherited?
color	The color of the text. This is a handy way to make headings or emphasized text stand out.	A color name, color code, or RGB color value.	Yes
background-color	The color behind the text, for just that element.	A color name, color code, or RGB color value. You can also use the word "transparent".	No[a]

[a] The background-color style property doesn't use inheritance (page 148). This means that if you give the <body> section of a page a blue background color and you place a heading on the page, the heading doesn't inherit the blue background. However, there's a trick. If you don't explicitly assign a background color to an element, its color is transparent. This means the color of the containing element shows through, which has the same effect as inheritance. So the heading in this example still ends up with the appearance of a blue background.

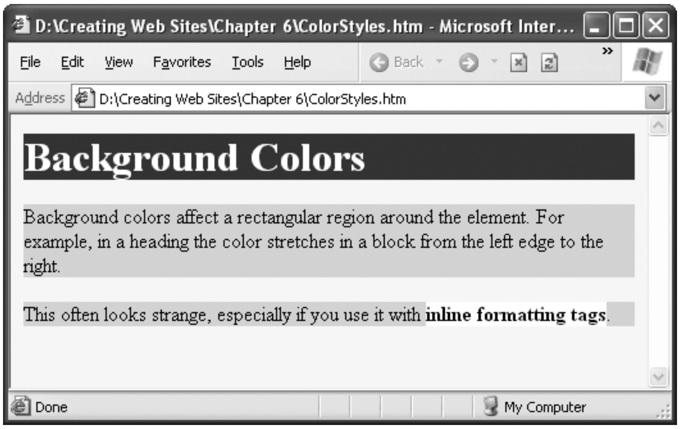
The *color* property is easy to understand—it's the color of your text. The *background-color* property is a little more unusual.

If you apply a background color to the <body> element of a Web page, the whole page adopts that color, as you might expect. However, if you specify a background color for an individual element, like a heading, the results are a bit stranger. That's because CSS treats each element as though it's enclosed in an invisible rectangle. When you apply a background color to an element, CSS applies that color to just that rectangle.

For example, the following style sheet applies different background colors to the page, headings, paragraphs, and any bold text:

```
body {
  background-color: yellow;
}

h1 {
  color: white;
  background-color: blue;
}

p {
  background-color: lime;
}

b {
  background-color: white;
}
```

Figure 6-5 shows the result.


D:\Creating Web Sites\Chapter 6\ColorStyles.htm - Microsoft Inter...
File Edit View Favorites Tools Help
Back
Address D:\Creating Web Sites\Chapter 6\ColorStyles.htm
Done
My Computer

Background Colors

Background colors affect a rectangular region around the element. For example, in a heading the color stretches in a block from the left edge to the right.

This often looks strange, especially if you use it with **inline formatting tags**.

Figure 6-5:
If you apply a background color to an element like <h1>, it colors just that line. If you use it on an inline element like <b> or <span>, it affects only the words in that element. Both results look odd—it's a little like someone went wild with a highlighter. A better choice is to apply a background color to the whole page by specifying it in the <body> element, or to tint just a large box-like portion of the page, using a container element like <div>.

Specifying a Color

The trick to using color is finding the code that indicates the exact shade of electric blue you love. You can go about this in several ways. First of all, you can indicate your color choice with a plain English name, as you've seen in the examples so far. Unfortunately, this system only works with a small set of 16 colors (aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow). Some browsers accept other names, but none of these names are guaranteed to be widely supported, so it's best to use another approach. CSS gives you two more options: hexadecimal color values and RGB (or *red-green-blue*) values.

Hexadecimal color values

With *hexadecimal color values* you use a strange-looking code that starts with a number sign (#). Technically, hexadecimal color values are made up of three numbers that represent the amount of red, green, and blue that go into creating a color. (You can create any color by combining various amounts of these three primary colors.) However, the hexadecimal color value combines these three ingredients into an arcane code that's perfectly understandable to computers, but utterly baroque to normal people.

You'll find hexadecimal color notation kicking around the Web a lot, because it's the original format for specifying colors under HTML. However, it's about as intuitive as reading the 0s and 1s that power your computer.

Here's an example:

```
body {
  background-color: #E0E0E0
}
```

Even a computer nerd can't tell that #E0E0E0 applies a light gray background. To figure out the code you need for your favorite color, check out the "Finding the Right Color" section further down this page.

RGB color values

The other approach to specifying color values is to use RGB values. According to this more logical approach, you simply specify how much red, green, and blue you want to "mix in" to create your final color. Each component takes a number from 0 to 255. For example, a color that's composed of red, green, and blue, each set to 255, appears white; on the other hand, all those values set to 0 generates black.

Here's an example of a nice lime color:

```
body {
  background-color: rgb(177,255,20)
}
```

Finding the Right Color

Style sheets can handle absolutely any color you can imagine. But how do you find the color code for the perfect shade of sunset orange (or dead salmon) you need?

Sadly, there's no way this black-and-white book can show you your choices. But there *are* a lot of excellent color-picking programs online. For example, try *www.colorpicker.com*, where all you need to do is click a picture to preview the color you want (and to see its hexadecimal code). Or try *www.colorschemer.com/online.html*, where you can find groups of colors that complement each other, which is helpful

FREQUENTLY ASKED QUESTION

Web-Safe Colors

Will the colors I pick show up on other computers?

Decades ago, when color became the latest fad in Web pages, the computing world was very different. The average PC couldn't handle a wide variety of colors. Many computers could display a relatively small set of 256 colors, and had to deal with other colors by *dithering*, a dubious process that combines little dots of several colors to simulate a different color, leading to an unattractive speckled effect. To avoid dithering, Web designers came up with a standard called *Web-safe colors*, which identifies 216 colors that any PC can reliably replicate. Even better, they look almost exactly the same on every computer.

But the world has changed, and you'd be hard pressed to find a computer that can't display at least 65,000 colors (a standard called 16-bit color, or *high color*). Most support a staggering 16.7 million colors (a standard called 24-bit color, or *true color*). In fact, even very small devices (like cell phones and palmtop computers) support every color a Web designer will ever need.

So here's the bottom line: in the 21st century, it's safe for Web designers to finally forget about Web-safe colors. However, it's still a good idea to check out your Web pages and color preferences on a variety of computers. That's because different monitors don't always reproduce the colors exactly–some tend to tint colors unexpectedly, and Windows computers tend to produce darker colors than their Macintosh counterparts (even when using the same monitor). Pick colors carefully, because a color combination that looks great on your PC can look nauseating (or worse, be illegible) on someone else's.

for creating Web sites that look professionally designed. If you use a Web design tool like Expression Web or Dreamweaver, you have an even easier choice—the program's built-in color-picking smarts, as shown back in Figure 6-3.

Note: The RGB system lets you pick any of 16.7 million colors, which means that no color-picking Web site will actually show you every single possible RGB color code (if they do, make sure you don't hit the Print button; even with 10 colors per line, you'd wind up with thousands of pages). Instead, most sites limit you to a representative sampling of colors. This works, because many colors are so similar that they're nearly impossible to distinguish by eye.

The RGB color standard is also alive and well in many computer programs. For example, if you see a color you love in a professional graphics program like Photoshop (or even in a not-so-professional graphics program, like Windows Paint), odds are there's a way to get the red, green, and blue values for that color. This gives you a great way to match the text in your Web page with a color in a picture. Now that's a trick that will please even the strictest interior designer.

Fonts

Using the CSS font properties, you can choose a font family, font weight (its boldness setting), and font size (see Table 6-2). Be prepared, however, for a bit of Web-style uncertainty, as this is one case where life isn't as easy as it seems.

DESIGN TIME

Making Color Look Good

Nothing beats black text on a white background for creating crisp, clean, easy-to-read Web pages with real presence. This black-and-white combination also works best for pages that have a lot of colorful pictures. It's no accident that almost every top Web site, from news sites (*www.cnn.com*) to search engines (*www.google.com*) to e-commerce shops (*www.amazon.com*) and auction houses (*www.ebay.com*), use the winning combination of black on white.

But what if you're just too colorful a person to leave your Web page in plain black and white? The best advice is to follow the golden rule of color: *use restraint*. Unless you're creating a sixties revival site or a Led Zeppelin tribute page, you don't want your pages to run wild with color. Here are some ways to inject a splash of color without letting it take over your Web page:

- **Go monochrome**. That means use black, white, and one other dark color. Use the new color to emphasize an important design element, like subheadings in an article. For example, the *Time* magazine Web site once used its trademark red for headlines (although now it favors a sleeker black-and-white combination).
- **Use lightly shaded backgrounds**. Sometimes, a faint wash of color in the background is all you need to perk up a site. For example, a gentle tan or gold can suggest elegance or sophistication (see the Harvard library site at *http://lib.harvard.edu*). Or light pinks and yellows can get shoppers ready to buy sleepwear and other feminine accoutrements at Victoria's Secret (*www.victoriassecret.com*).
- **Use color in a box**. Web designers frequently use shaded boxes to highlight important areas of a Web page (see Wikipedia at *http://en.wikipedia.org*). You'll learn how to create boxes later in this chapter.
- **Be careful about using white text**. White text on a black or dark blue background can be striking–and strikingly hard to read. The rule of thumb is to avoid it unless you're trying to make your Web site seem futuristic, alternative, or gloomy. (And even if you do fall into one of these categories, you might still get a stronger effect with a white background and a few well-chosen graphics with splashy electric colors.)

***Table 6-2.** Font properties*

Property	Description	Common Values	Can Be Inherited?
font-family	A list of font names. The browser scans through the list until it finds a font that's on your visitor's PC. If doesn't find a supported font, it uses the standard font it always uses.	A font name (like Verdana, Times, or Arial) or a generic font-family name: serif, sans-serif, monospace.	Yes
font-size	Sets the size of the font.	A specific size, or one of these values: xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger.	Yes
font-weight	Sets the weight of the font (how bold it appears).	normal, bold, bolder, lighter	Yes
font-style	Lets you apply italic formatting.	normal, italic	Yes

Table 6-2. *Font properties (continued)*

Property	Description	Common Values	Can Be Inherited?
font-variant	Lets you apply small caps, which turns lowercase letters into smaller capitals (LIKE THIS).	normal, small-caps	Yes
text-decoration	Applies a few miscellaneous text changes, like underlining and strikeout. Technically speaking, these aren't part of the font (the browser adds these).	none, underline, overline, line-through	Yes
text-transform	Transforms text so that it's all capitals or all lowercase.	none, uppercase, lowercase	Yes

Although most CSS font properties are straightforward, the *font-family* property has a nasty surprise—it doesn't always work. The inescapable problem you face is that no two computers have the same set of fonts installed, so the fonts you use to design your Web page won't necessarily be the fonts your visitors have installed on their PCs. A simple way to solve this problem is to create browsers that automatically download fonts they don't have, but this would be a Web nightmare. First, automatic downloads could swamp the average computer's hard drive with thousands of (potentially low-quality) fonts. Second, it would infuriate the software companies who sell fonts. (Fonts aren't free, and so wantonly copying them from one computer to another isn't kosher.)

There may be practical solutions to these problems, but, unfortunately, browser companies and the people who set Web standards have never agreed on any. As a result, any font settings you specify are just recommendations. If a PC doesn't have the font you request, the browser reverts to the standard font it uses whenever it's on a site that doesn't have special font instructions.

Given that caveat, you're probably wondering why you should bother configuring font choices at all. Well, here's one bit of good news. Instead of requesting a font and blindly hoping that it's available to a browser, you can create a list of *font preferences*. That way, the browser tries to match your first choice and, if it fails, your second choice, and so on. At the end of this list, you should use one of the few standard fonts that almost all PCs support in some variation. You'll see this technique at work in the next section.

Specifying a Font

To select a font, you use the *font-family* attribute. Here's an example that changes the font of an entire page:

```
body {
  font-family: Arial;
}
```

DESIGN TIME

Graphical Text

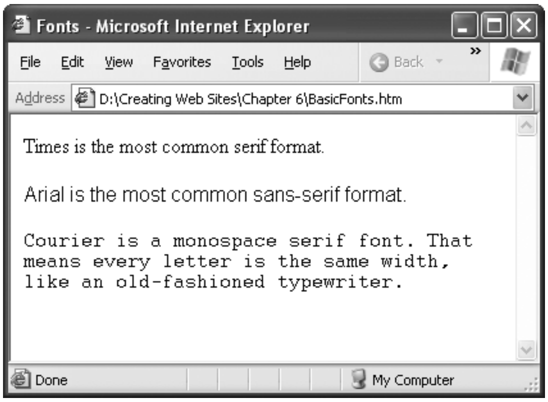
The only guaranteed cure for font woes is *graphical text*. With graphical text, you don't type your content into an XHTML file. Instead, you perfect it in a drawing program, save it as a picture, and then display the *picture* of that text on your page using the <img> element.

Graphical text is clearly unsuitable for large amounts of text. First of all, it's an image and images require more bytes of storage space than text, so it bloats the size of your Web page horribly. It's also much less flexible. For example, graphical text can't adjust itself to fit the width of a browser window or take into account your visitors' browser preference settings. There's also no way for a visitor to search an image for specific words (or for a Web search engine to figure out what's on your site).

However, graphical text is commonly used for Web page menus, buttons, and headings, where these issues aren't nearly as important. Sometimes, graphical text isn't obvious. For example, you may never have noticed that the section headings on your favorite online newspaper are actually images. To figure out if a Web site uses graphical text or the real deal, try to select the text with your mouse. If you can't, the text is really a picture.

You'll learn how to use graphics (including graphical text) in Chapter 7.

Arial is a *sans-serif* font found on just about every modern PC, including those running Windows, Mac OS, Unix, and Linux operating systems. (See Figure 6-6 for more about the difference between serif and sans-serif fonts.)


Fonts - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address D:\Creating Web Sites\Chapter 6\BasicFonts.htm
Done
My Computer

Times is the most common serif format.

Arial is the most common sans-serif format.

Courier is a monospace serif font. That means every letter is the same width, like an old-fashioned typewriter.

Figure 6-6:
Serif fonts use adornments, or serifs, that make them easier to read in print. This book uses a serif font, for example. If you look closely at the letter "S" in the body text, you'll see tiny curlicues in the top-right and bottom-left corners. On the other hand, sans-serif fonts have a spare, streamlined look. They can make pages seem less bookish, less formal, more modern, and colder.

To be safe, when you create a font list, always end it with a generic font-family name. Every PC supports generic fonts under the font-family names serif, sans-serif, and monospace.

Here's the modified rule:

```
body {
  font-family: Arial, sans-serif;
}
```

At this point, you might be tempted to get a little creative with this rule by adding support for a less common sans-serif format. Here's an example:

```
body {
  font-family: Eras, Arial, sans-serif;
}
```

If Eras is relatively similar to Arial, this technique might not be a problem. But if it's significantly different, it's a bad idea.

The first problem is that by using a nonstandard font, you're creating a Web page whose appearance may vary dramatically depending on the fonts installed on your visitor's PC. Whenever pages vary, it becomes more difficult to really tweak them to perfection because you don't know exactly how they'll appear elsewhere. Different fonts take up different amounts of space, and if text grows or shrinks, the layout of other elements (like pictures) changes, too. Besides, is it really that pleasant to read KidzzFunScript or SnoopDawg font for long periods of time?

A more insidious problem happens if a visitor's computer has a font with the same name that looks completely different. Even worse, browsers may access an online database of fonts to try and find a similar font that's already installed. This approach can quickly get ugly. At worst, either of these problems can lead to illegible text.

Tip: Most Web page editors won't warn you when you apply a nonstandard font, so be on your guard. If your font isn't one of a small set of widely distributed Web fonts (more on those in a moment), you shouldn't use it.

Finding the Right Font

To make sure your Web page displays correctly, you should use a standard font that's widely available. But just what are these standard fonts? Unfortunately, Web experts aren't always in consensus.

If you want to be really conservative, you won't go wrong with any of these fonts:

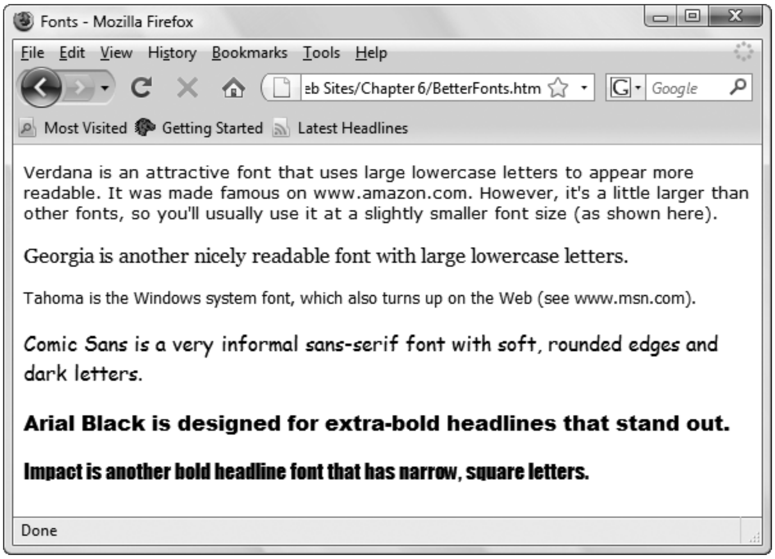
- Times
- Arial
- Helvetica
- Courier

Of course, all these fonts are insanely boring. If you want to take more risk, you can use one of the following fonts, found on almost all Windows and Mac computers (but not necessarily on other operating systems, like Unix):

- Verdana
- Georgia
- Tahoma

- Comic Sans MS
- Arial Black
- Impact

To compare these different fonts, see Figure 6-7.


Fonts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
eb Sites/Chapter 6/BetterFonts.htm
Google
Most Visited Getting Started Latest Headlines
Done

Verdana is an attractive font that uses large lowercase letters to appear more readable. It was made famous on www.amazon.com. However, it's a little larger than other fonts, so you'll usually use it at a slightly smaller font size (as shown here).

Georgia is another nicely readable font with large lowercase letters.

Tahoma is the Windows system font, which also turns up on the Web (see www.msn.com).

Comic Sans is a very informal sans-serif font with soft, rounded edges and dark letters.

Arial Black is designed for extra-bold headlines that stand out.

Impact is another bold headline font that has narrow, square letters.

Figure 6-7:
Have you spotted these fonts at large on the Web?

Verdana, Georgia, and Tahoma can all help give your Web pages a more up-to-date look. However, the characters in Verdana and Tahoma start off a bit large, so you usually need to ratchet them down a notch in size (a technique described in the next section).

For a good discussion of fonts, the platforms that reliably support them, and the pros and cons of each font family (some fonts look nice onscreen, for example, but they generate lousy printouts) see *http://web.mit.edu/jmorzins/www/fonts.html* and *www.upsdell.com/BrowserNews/res_fontsamp.htm*.

Font Sizes

Once you sort out the thorny issue of choosing a font, you may also want to change its size. It's important that you select a text size that's readable and looks good. Resist the urge to shrink or enlarge text to suit your personal preferences. Instead, aim to match the standard text size you see on other popular Web sites.

Despite what you might expect, you don't have complete control over the size of the fonts on your Web pages. Most site visitors use browsers that let them scale font sizes up or down, either to fit more text on-screen or, more commonly, to make text easy to read on a high-resolution monitor. In Internet Explorer and Firefox, you find these options in the View → Text Size menu.

A browser's font-size settings don't completely override the size you've set in your Web page, however. Instead, they tweak it up or down. For example, if you choose to use a large font size on your Web page (which corresponds to a setting of about 15 points in a word processor) and a visitor using Internet Explorer selects View → Text Size → Larger, the text size grows about 20 percent, to 18 points.

The fact that your visitors have this kind of control is another reason you shouldn't use particularly small or large fonts on your pages. When you combine them with browser preferences, a size that's a little on the large size could become gargantuan, and text that's slightly small could turn unreadable. The best defense for these problems is to test your Web page with different browsers *and* different font size preferences.

As you'll discover in the following sections, you can set font sizes in several ways.

Keyword sizing

The simplest way to specify the size of your text is to use one of the size values listed previously in Table 6-2. For example, to create a really big heading and ridiculously small text, you can use these two rules:

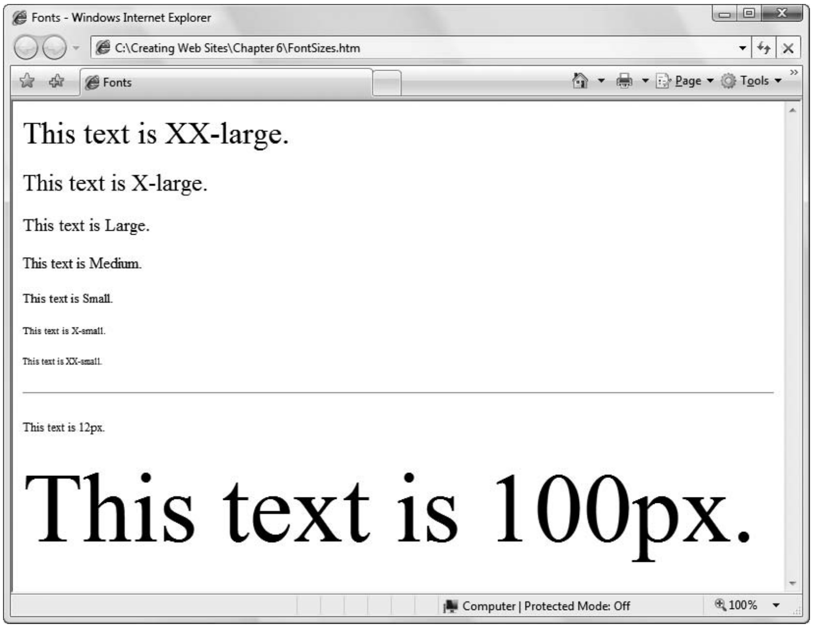
```
body {
  font-size: xx-small;
}
h1 {
  font-size: xx-large;
}
```

These size keywords are often called *absolute sizes*, because they apply an exact size to text. Exactly what size, you ask? Well, that's where it gets a bit complicated. These size details aren't set in stone—different browsers are free to interpret them in different ways. The basic rule of thumb is that the font size *medium* corresponds to a browser's standard text size, which is the size it uses (12 points) if a Web site doesn't specify a text size. Every time you go up a level, you add about 20 percent in size. (For math geeks, that means that every time you go down a level, you lose about 17 percent.)

The standard font size for most browsers is 12 points (although text at this size typically appears smaller on Macs than on Windows PCs). That means *large* text measures approximately 15 points, *x-large* text is 18 points, and *xx-large* text is 27 points.

Figure 6-8 shows the basic sizes you can choose from.

Note: When using size keywords, make sure your Web page specifies an XHTML doctype. If you don't, Internet Explorer renders your page in the dreaded "quirks" mode, which makes your text one size larger than it should be. As a result, your page won't look the same in Internet Explorer as it does in other browsers, like Firefox.


Fonts - Windows Internet Explorer
C:\Creating Web Sites\Chapter 6\FontSizes.htm
Fonts
Page
Tools
Computer | Protected Mode: Off
100%

This text is XX-large.

This text is X-large.

This text is Large.

This text is Medium.

This text is Small.

This text is X-small.

This text is XX-small.

This text is 12px.

This text is 100px.

Figure 6-8: *There are seven standard text sizes, ranging from xx-large to xx-small. You can dictate font sizes, too, by specifying a pixel measurement.*

Percentage sizing

Another font-size option is to use percentage sizes instead of size keywords. For example, if you want to make sure your text appears normal size, use this rule:

```
body {
  font-size: 100%;
}
```

And if you want to make your text smaller, use something like this:

```
body {
  font-family: Verdana,Arial,sans-serif;
  font-size: 83%;
}
```

This displays text at 83 percent of its standard size. It doesn't matter whether the standard size is considered *small* (Internet Explorer) or *medium* (most other browsers). This particular example creates nicely readable text with the Verdana font.

It's just as easy to upsize text:

```
h1 {
  font-size: 120%;
}
```

But keep in mind that 100 percent always refers to the standard size of normal *paragraph* text, not the standard size of the element you're styling, like an h1 heading. So if you create a heading with text sized at 120 percent, your heading is going to be only a little bigger than normal paragraph text, which is actually quite a bit smaller than the normal size of an <h1> heading.

Using percentage sizes is the safest, most reliable way to size text. Not only does it provide consistent results across all browsers, it also works in conjunction with the browser size preferences described earlier.

Relative sizing

Another approach for setting font size is to use one of two *relative size* values—"larger" or "smaller". This is a bit confusing, because as you learned in the last section, absolute sizes are already relative—they're all based on the browser setting for standard text.

The difference is that relative size settings are influenced by the font of the element that contains them. The easiest way to understand how this works is to consider the following style sheet, which has two rules:

```
body {
  font-size: xx-small;
}

b {
  font-size: larger;
}
```

The first rule applies an absolute *xx-small* size to the whole page. If your page includes a <b> element, the text inside *inherits* the xx-small size (see page 148 for a recap of inheritance), and then the second style rule steps it up one notch, to x-small.

Now consider what happens if you edit the body text style above to use a larger font, like this:

```
body {
  font-size: x-small;
}
```

Now all bold text will be one level up from *x-small*, which is *small*.

The only limit to the two relative sizes is that they can step up or down only one level. However, you can get around this limitation by using font numbers. For example, a size of +2 is a relative size that increments a font two levels. Here's an example:

```
body {
  font-size: x-small;
}
```

```
b {
  font-size: +2;
}
```

Now the bold text becomes *medium* text, because *medium* is two levels up from *x-small*.

Relative sizes are a little trickier to get used to than absolute sizes. You're most likely to use them if you have a style sheet with a lot of different sizes. For example, you might use a relative size for bold text if you want to make sure bold text is always a little bit bigger than the text around it. If you were to use an absolute size instead, the bold text would appear large in relation to small-sized paragraph text, but it wouldn't stand out in a large-sized heading.

Note: When you use absolute or relative sizes, you create flexible pages. If a visitor ratchets up the text size using his browser's preferences, the browser resizes all your other fonts proportionately.

Pixel sizing

Most of the time, you should rely on absolute and relative sizing for text. However, if you want to have more control, you can customize your font size precisely by specifying a *pixel size*. Pixel sizes can range wildly, with 12 pixels and 14 pixels being about normal for body text. To specify a pixel size, use the number immediately followed by the letters *px*, as shown here:

```
body {
  font-size: 11px;
}
h1 {
  font-size: 24px;
}
```

Note: Don't put a space between the number and the letters "px". If you do, your rule may work in Internet Explorer but it will thoroughly confuse other browsers.

As always, you need to test, refine, and retest your font choice to get the sizes right. Some fonts look bigger than others, and require smaller sizes. Other fonts work well at larger sizes, but become less legible as you scale them down in size.

Web purists avoid using exact sizes because they're horribly inflexible in Internet Explorer. For example, if a near-sighted visitor has upped the text size settings in Internet Explorer, this adjustment won't have any effect on a page that uses pixel sizing. (For some reason, other browsers don't suffer from this problem—they're able to resize pages even if you use pixel sizes.) As a result, using pixel sizes is a quick shortcut to inconsistent results. By using them, you could inadvertently lock out certain audiences or create pages that visitors find difficult to read or navigate on certain types of browsers. It just goes to show that in the Web world there's a price to be paid for getting complete control over formatting.