Bonus Chapter 2

Formatting with CSS

In This Chapter

- Getting used to CSS
- Managing color with CSS
- Changing text fonts and styles
- Adding borders and backgrounds to page elements
- Building multi-column forms with floating formats
- Creating absolutely positioned elements

XHTML is a powerful technology, but modern Web pages require a combination of XHTML and Cascading Style Sheets (CSS). This technology works alongside XHTML. While XHTML is used to provide the basic framework and content, CSS is used to specify the visual aspects of the page.

Introduction to CSS

Early forms of HTML paid very little attention to the visual aspects of page layout. The original plan was for HTML to be more tied to the *meaning* of page elements rather than their display. In the very early days of the Web, this was fine, but soon people wanted far more sophisticated design elements than HTML was capable of producing. Browser manufacturers responded by adding vendor-specific tags that added new capabilities but greatly complicated development efforts.

XHTML is an attempt to return HTML to its earlier simplicity. In the strict form of XHTML, all the tags that were used to directly manage the appearance of the page (tags like `<font>`, `<center>`, `<b>`, and `<i>`) are removed. Rather than having special tags indicate formatting, a new language has been devised that can provide very powerful formatting features to virtually any HTML or XHTML tag. CSS is this language.

Overview of CSS

CSS works by describing certain parts of the page (one particular tag, all the tags of a specific type, or all tags sharing a particular characteristic). For each of these tag groups, you can then identify a number of *rules*. Each rule is a name/value pair. Take a look at the simple page displayed in Figure BC2-1.

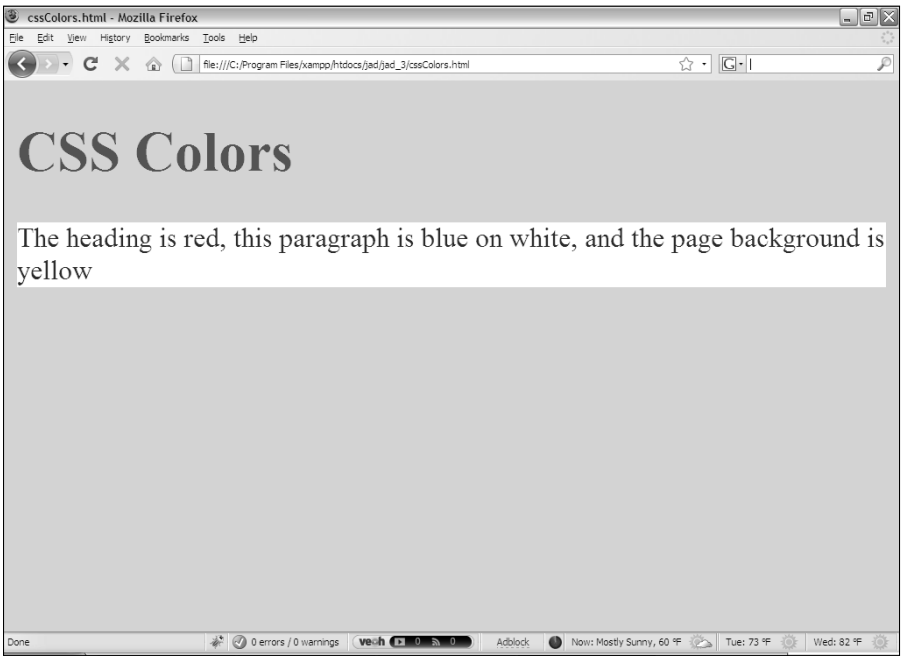

Figure BC2-1: This page uses CSS to change the default colors.

The code to produce this page is shown here:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
<head>
  <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />
  <title>cssColors.html</title>
  <style type = «text/css»>
    body {
      background-color: yellow;
    }

    h1 {
      color: red;
```

```
    }

    p {
      color: blue;
      background-color: white;
    }
  </style>

</head>
<body>
  <h1>CSS Colors</h1>

  <p>
    The heading is red, this paragraph is blue on white,
          and the page background is yellow
  </p>
</body>
</html>
```

The new elements are not terribly surprising, but they are quite powerful. Note that the colors are changed without changing anything in the HTML body. All the real action happens in a special part of the header. This is part of the charm of CSS. It makes your HTML a lot cleaner, because much of the formatting can go elsewhere. Here's how to add color formatting (or any CSS, for that matter) to your pages:

1. **Begin with clean, valid HTML code.**

   Be sure your HTML or XHTML code validates before you try to do too much else with it. Improper HTML won't always respond to CSS the way you think it should.

2. **Add a `<style>` tag to the page heading**

   The `<style>` tag allows you to add a style sheet (a list of formatting instructions) directly on the page. See the "Managing levels of CSS" section later in this chapter for how to put style sheets in other places.

3. **Set the style's type to `text/css`.**

   The only value you'll ever use for style type is `text/css`.

4. **Indicate the tag you want to modify.**

   The first tag I change is the `body`. Changes to the `body` tag will affect the entire visible part of the page, so this is a great place to start. Just type the tag name (without the angle braces).

5. **Use squiggly braces (`{}`) to enclose the rules for this style.**

   You might have several rules to describe how the body should be displayed. For each style, you'll need to enclose all the rules in a pair of braces.

6. **Denote the `background-color` attribute.**

   Every rule consists of an *attribute* and a *value*. The attribute is a built-in characteristic of the element, and the value is what value we want to give that attribute. For now, change the background color of the background, so type `background-color:`. Note that you must end the attribute name with a colon (`:`). Capitalization and spelling count, so be careful.

7. **Indicate the value you want to apply to the attribute.**

   For this example, I want a yellow background, so I just type the value `yellow;` after the attribute `background-color:`. Doing so sets the body's background color to yellow.

8. **End each value with a semicolon (`;`).**

   Every value must end with a semicolon. If one tag has a lot of rules (which is common), the semicolons help the browser separate all the various rules from each other.

9. **Change the foreground color with the `color` attribute.**

   Note how I make the level-1 headline red: I set `h1` as the new tag, and set its `color` attribute to the value `red`.

10. **One tag can have multiple rules.**

    Take a look at the rules for the paragraph (`p`) tag. You'll see that I set both the foreground and background colors.

You learn more about which colors can be used in the following section called (cleverly enough) "Working with colors." For now, though, just play around with the various color names. Most of the common color names will work just like you expect. When you want a fancier color, you'll have to learn how to use the fancy hex codes described in that section.

Working with colors

CSS has a rich mechanism for working with colors. Whenever you want to specify a color, you can simply type the color name. Figure BC2-2 demonstrates the 16 color names that CSS understands.

Of course, a page that demonstrates colors won't be very useful in a black-and-white book, so you'll definitely want to view this page (and all the others in this book) in color on the companion Web site at `www.aharrisbooks.net/jad/`.

Figure BC2-2: The color names understood in CSS.

The `namedColors.html` page featured in Figure BC2-2 has another trick up its sleeve. Take a look at the source code and you'll see what I mean:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
  <head>
    <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />
    <title>namedColors.html</title>
  </head>

  <body>
    <h1>Named Colors</h1>
    <table border=»1»>

      <tr>
        <th>color name</th>
        <th>color value</th>
      </tr>

      <tr>
        <td>aqua</td>
        <td style=»background-color: aqua;»><br /></td>
      </tr>

      <tr>
```

```
        <td>black</td>
        <td style=»background-color: black;»><br /></td>
      </tr>

      <tr>
        <td>blue</td>
        <td style=»background-color: blue;»><br /></td>
      </tr>

      <tr>
        <td>fuchsia</td>
        <td style=»background-color: fuchsia;»><br /></td>

      </tr>

      <tr>
        <td>gray</td>
        <td style=»background-color: gray;»><br /></td>
      </tr>

      <tr>
        <td>green</td>
        <td style=»background-color: green;»><br /></td>
      </tr>

      <tr>
        <td>lime</td>
        <td style=»background-color: lime;»><br /></td>
      </tr>

      <tr>
        <td>maroon</td>
        <td style=»background-color: maroon;»><br /></td>
      </tr>

      <tr>
        <td>navy</td>
        <td style=»background-color: navy;»><br /></td>
      </tr>

      <tr>
        <td>olive</td>
        <td style=»background-color: olive;»><br /></td>
      </tr>

      <tr>
        <td>purple</td>
        <td style=»background-color: purple;»><br /></td>
      </tr>

      <tr>
        <td>red</td>
        <td style=»background-color: red;»><br /></td>
```

```
      </tr>

      <tr>
        <td>silver</td>
        <td style=»background-color: silver;»><br /></td>
      </tr>

      <tr>
        <td>teal</td>
        <td style=»background-color: teal;»><br /></td>
      </tr>

      <tr>
        <td>white</td>
        <td style=»background-color: white;»><br /></td>

      </tr>

      <tr>
        <td>yellow</td>
        <td style=»background-color: yellow;»><br /></td>
      </tr>
    </table>
  </body>
</html>
```

This page uses a table to demonstrate all the colors recognized by CSS. Next to the color name is a table cell with the background color set to that color. Note that in this case, rather than having one large style sheet at the top of the document, I added several smaller styles directly inside the body of the HTML page. This technique is called *local styles*.

Most HTML tags have an attribute called `style`. You can add CSS rules directly to this style if you want. To make the aqua-colored cell, for example, look at the following code:

```
<td style="background-color: aqua;"><br /></td>
```

Local styles are easy to use, but they aren't perfect. They tend to clutter up the HTML code, which was exactly what CSS was trying to avoid. Still, the technique is useful in a few circumstances, like this example. Local styles are used quite a bit in animation, so you see them again in Chapter 8.

Using ids and classes

CSS is pretty useful because it allows you to quickly add a style to all the elements of a particular type. For example, you can very easily make all the paragraphs on a page have the same color. But what if you only want to apply a style to a single element? And what if you have two kinds of paragraphs that should have different styles?

Figure BC2-3 illustrates the CSS way to solve exactly these problems.

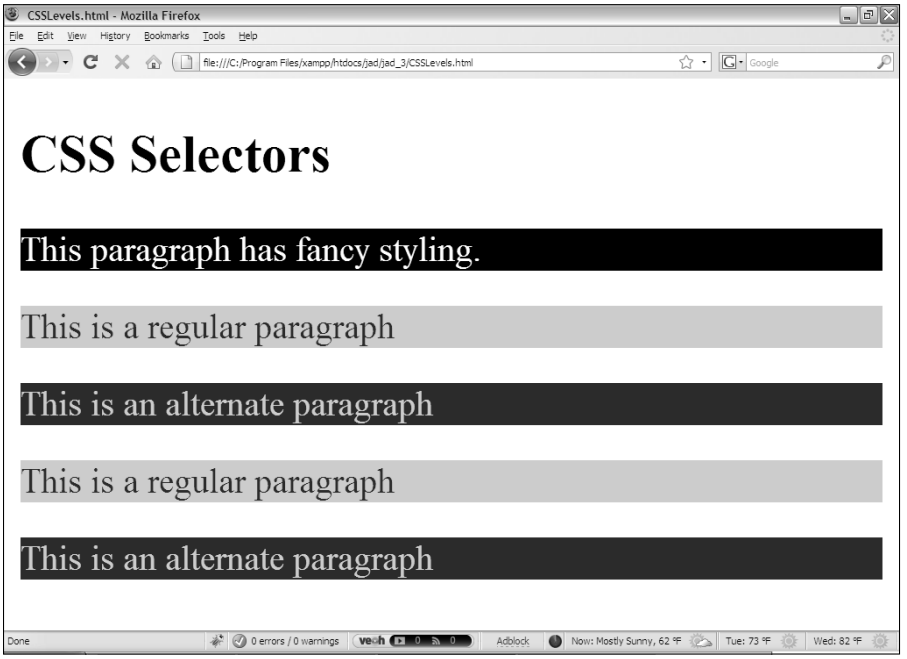

Figure BC2-3: This page has three different paragraph styles!

The `CSSLevels.html` page mainly consists of paragraphs, but there are three different paragraph styles. Ordinary paragraphs are light blue with dark blue letters. There are two special kinds of paragraphs. One paragraph has a special name: `fancy`. The `fancy` paragraph has its own styling. There is only one `fancy` paragraph, but there are two paragraphs using the `alternate` style. Take a look at the code and then I explain how it all works:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
  <head>
    <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />
    <title>CSSLevels.html</title>
    <style type = «text/css»>
    p {
      background-color: lightblue;
      color: blue
    }

    #fancy {
      background-color: black;
```

```
      color: white;
    }

    .alternate {
      background-color: blue;
      color: lightblue;
    }

    </style>
</head>

<body>
  <h2>CSS Selectors</h2>
  <p id = «fancy»>
    This paragraph has fancy styling.
  </p>

  <p>
    This is a regular paragraph
  </p>

  <p class = «alternate»>
    This is an alternate paragraph
  </p>

  <p>
    This is a regular paragraph
  </p>

  <p class = «alternate»>
    This is an alternate paragraph
  </p>

</body>
</html>
```

First, take a look at the HTML. It's almost the same, but I've added some special indicators to some of the paragraphs:

- **Ordinary paragraphs.** These paragraphs don't require any special features. They will be styled according to the regular `p` style rule.
- **Named paragraphs.** The first paragraph has an `id` property. This property allows you to specify a name for any HTML object. The `id` must be unique — that is, only one object on-screen can have any particular `id`. The `id` can be anything you want, but it should be one word without spaces or punctuation. If an object has an `id` property, you can apply a style to that particular `id`.
- **Paragraphs in a class.** In addition to the `id` property, you can assign a `class` to any HTML element. The `class` attribute allows you to indicate that an element is a member of a particular class. Unlike the `id`, you can have as many elements in the same class as you want. The alternate

paragraphs all have the `class` attribute set to `alternate`. You can use any term you want as a `class` name, but it should not have spaces or punctuation. You can apply a style to all elements with a certain class. Different kinds of elements can all have the same class, so you can apply the same class to paragraphs and headings if you want.

After you've applied `id` properties and classes in the HTML, you can modify the CSS code to apply styles to the various elements.

Use the number sign (`#`) in front of the `id` in your CSS to indicate you want to style an element with that `id`. For example, this code styles anything with the `id` `fancy`:

```
#fancy {
  background-color: black;
  color: white;
}
```

Use the period in front of a class name to define a style for a particular class. For example, you can style all elements of the `alternate` class with this code:

```
.alternate {
  background-color: blue;
  color: lightblue;
}
```

Use the `id` approach when you want to apply a style to an individual element in the page. Use the tag name when you want to attach a style to all the elements of a certain type. Use the `class` mechanism when you want to attach to a number of elements that might or might not be the same type.

Managing levels of CSS

CSS code can be added to a page in three different ways:

- **Locally inside the HTML body.** You can apply a style directly to most HTML tags using the `style` attribute. This technique is illustrated in the section, "Working with local styles," earlier in this chapter.
- **At the page level in the header.** This technique uses a `<style></style>` pair inside the head of the HTML page. This is a good way to specify styles for a specific page.
- **In an external document.** A style can be specified in a separate document and then referenced from a Web page. This approach allows you to share a set of style rules among several pages. It also cleans up the main page, as the styles are moved out of the way.

Figure BC2-4 demonstrates a simple page that uses an external style sheet:

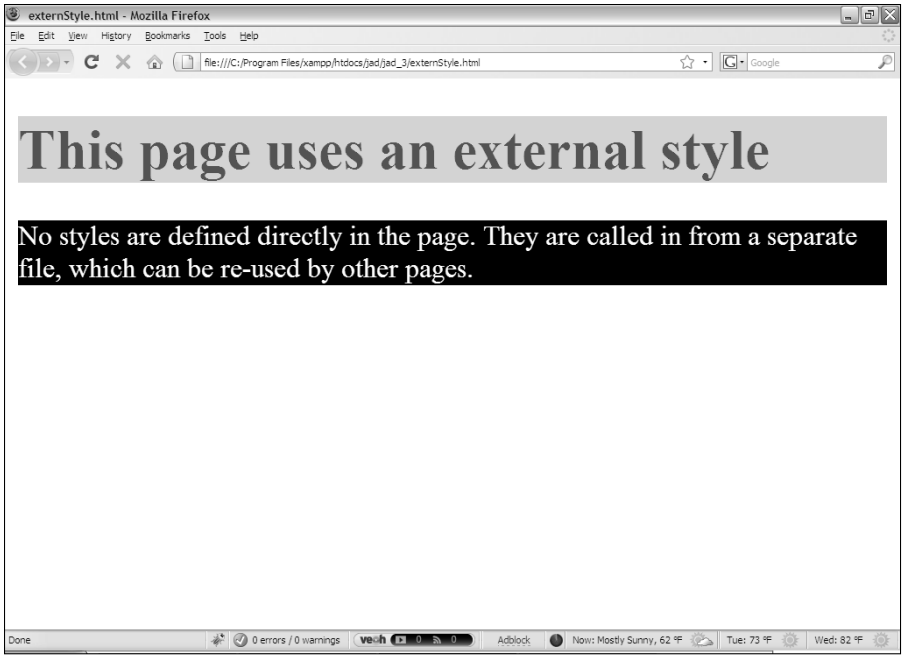

Figure BC2-4: This page pulls its style from another document.

When you look at the source code, you'll see no CSS at all:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
<head>
  <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />

  <link rel = «stylesheet»
        type = «text/css»
        href = «externStyle.css» />

  <title>externStyle.html</title>
</head>
<body>
  <h1>This page uses an external style</h1>

  <p>
    No styles are defined directly in the page.
    They are called in from a separate file, which
    can be re-used by other pages.
  </p>
</body>
</html>
```

The interesting thing about this listing is how the page clearly has a style, but the style information is not directly present on the page. The secret is the `link` tag in the header. This tag allows you to bring in a style from a separate page. To build an external link, follow these steps:

1. **Add a `<link>` tag in the header.**

   The `<link>` tag allows you to associate another file with the current page. It is very useful for attaching a style sheet to a Web page.

2. **Set the `rel` attribute to `stylesheet`.**

   The `rel` attribute specifies the nature of the external file. Use the `stylesheet` value to indicate you are attaching a style to the page.

3. **Set the `type` to `text/css`.**

   Linked styles are indicated as `text/css` in the same way as embedded styles (the ones created with the `<style>` tag.)

4. **Specify the location of the style sheet with the `href` attribute.**

   Use the `href` attribute to indicate where the style sheet is on the file system. It's normally best to use a relative reference for style sheets. This allows you to move the page and the style sheet to the server together.

5. **You can use the same style sheet over and over.**

   Multiple pages can use the same style sheet. This is perfect if you have one site that will have many pages using the same style. You can then change the style on one page, and the new style will be reflected through the entire site.

The external style is another standard text file that can also be edited with a plain-text editor. It simply contains the style rules. It does not require the `<style></style>` pair. Here's the `externStyle.css` page called by `externStyle.html`:

```
h1 {
  color: red;
  background-color: yellow;
}

p {
  color: white;
  background-color: black;
}
```

Managing the Appearance of Your Page

Of course, Web pages do much more than change color. The main ways you can modify a page are by changing the appearance of text, adding borders and background images, and changing the overall layout.

Understanding hex colors

For basic colors (like red and yellow) the color names are perfectly fine, but sometimes you need something with a little more sophistication. Color names are a bit confusing, and there are only 16 color names guaranteed to be understood by CSS. It's also a bit difficult to adjust colors. For example, ask yourself, *What color has just a little more green than aqua?*

CSS has a more specific way of indicating colors. It's a little geeky, but very powerful. Each dot on a computer monitor is actually three different tiny color emitters: red, green, and blue. The computer can adjust the amount of color that comes out of each of these emitters. If you want to see a red dot, the red emitter is turned to full strength, and the green and blue emitters are turned completely off. You can combine the emitters to get various colors, so red and green makes blue.

You might be confused by the notion that red and green makes blue, because in elementary school art class, they taught you a totally different way of mixing colors. Both are actually correct. In elementary school, you start with white paper and use pigments to subtract color values. Paper art normally works in a *subtractive* color model (as does your computer printer). The monitor starts with blackness and adds various amounts of colored light, so it evokes an *additive* color model. While the approach is different, the result is the same.

If you want to specify a particular color in the computer world, you can specify how much red, green, and blue are used to make the color. Then it would make sense for red to be the color 100, 0, 0. This would mean "turn on all the red, and turn off green and blue."

However, computers don't work as naturally with percentages as we do. Computer organization works in a different way, so color values actually range from 0 to 255. (Ack.) To make it even worse, technical people often convert these numbers to base 16 (hexadecimal notation), which brings in all kinds of crazy numbers and even letters. Each value takes a two-digit value that ranges from 00 (completely off) to FF (full brightness.) This funky system is called *hexadecimal* notation (often abbreviated *hex*).

Don't panic. It's not that hard. Look at the `colorTester` program shown in Figure BC2-5 to see an illustration.

The `colorTester` program uses some JavaScript skills you will learn soon! Feel free to look at the source code to see how it works. For now, use the program to see how these hex values can be used to specify colors.

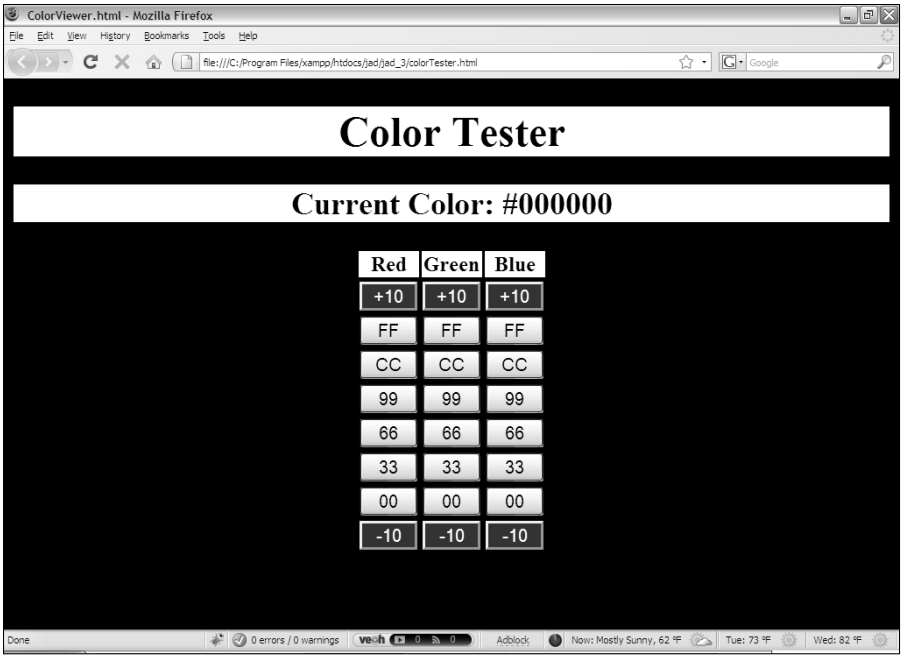

Figure BC2-5: Play with this page to see how hex colors work.

First, note how the page is arranged:

- **The page background is black.** The page background will change colors to reflect the current color settings.
- **There are three columns of buttons.** There are columns for red, green, and blue.
- **Gray buttons directly set hex values.** The gray buttons can be used to set a color value to a specific setting. Brighter values (FF) are on the top of the stack, with lower values (00) on the bottom.
- **The current color is modified on the fly.** As you click on the various buttons, the background color changes to reflect the current color, and the heading changes to indicate the hex value of the current color.
- **Gray buttons show preset values.** Web developers often begin with preset values in the ranges shown on the buttons (00, 33, 66, 99, CC, FF).

These values provide a reasonable range of colors while still being easy to modify.

- **Black buttons allow finer tuning.** Of course, you can use values besides the presets. If you want to add a little more red, for example, you can use the +10 button in the red column to do this.

You can use hex color values anywhere you use color names. For example, if you want to specify that a level-1 heading is red text on a yellow background, you can use these hex codes:

```
h1 {
  color: #FF0000;
  background-color: #FFFF00;
}
```

Use the pound sign (#) to indicate you are using hex values rather than color names. Hex values have a number of advantages over named colors:

- **There are more of them.** Only 16 named colors are officially recognized by CSS (although most browsers can read many more). With the hex system, you can actually represent more than 16 million different colors. Even if you stick with the 00336699CCFF system, you have 216 colors to play with.
- **Hex colors are easier to adjust.** You can directly tweak the hex values to get variations of the basic colors. If you want "a little more blue," this is much easier accomplished with hex colors than named colors.
- **Hex colors are more universal.** Most computer graphic programs use the hex notation, so you can sample a color in your graphics editor and match it in your Web page.
- **Color scheme generators can help you match colors.** If (like me) you have a design disability, you can use a tool such as the color-scheme generator at `http://colorschemedesigner.com`. This marvelous tool lets you play around with various color schemes in real time, and then generates hex codes you can use in your own page.

Modifying text

Web pages are primarily about text, and CSS has many great features for manipulating text. Figure BC2-6 shows a page with a number of text effects:

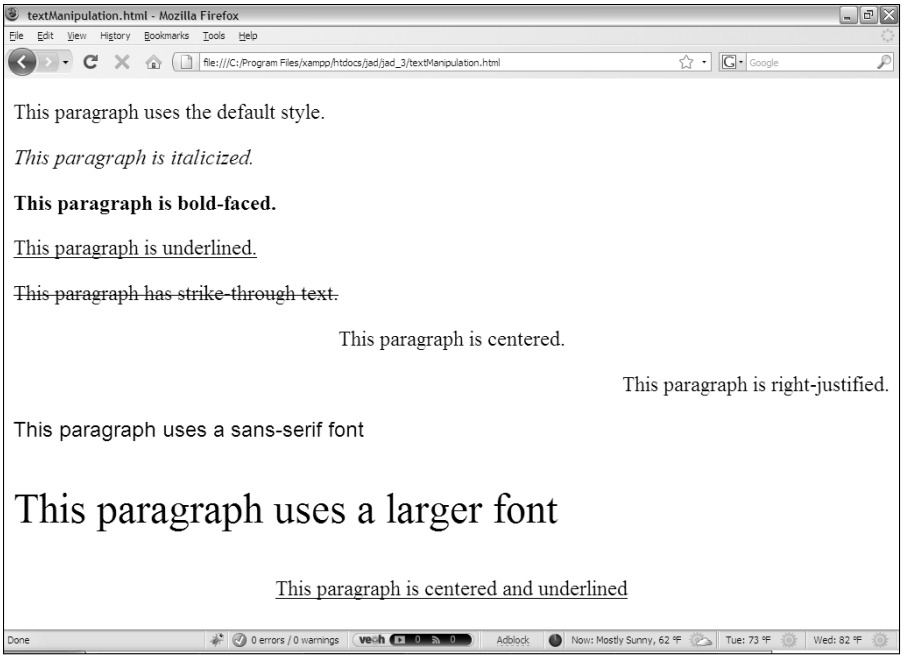

Figure BC2-6: Using various style effects to modify text.

Text can be manipulated in a number of interesting ways. Look over the HTML source of the `textManipulation.html` page to see the general overview:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
  <head>
    <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />
    <title>textManipulation.html</title>
    <link rel = «stylesheet»
          type = «text/css»
          href = «textManipulation.css» />

  </head>

  <body>

    <p>
      This paragraph uses the default style.
    </p>

    <p class = «italic»>
      This paragraph is italicized.
```

```
    </p>

    <p class = «bold»>
      This paragraph is bold-faced.
    </p>

    <p class = «underline»>
      This paragraph is underlined.
    </p>

    <p class = «strikeThrough»>
      This paragraph has strike-through text.
    </p>

    <p class = «center»>
      This paragraph is centered.
    </p>

    <p class = «right»>
      This paragraph is right-justified.
    </p>

    <p class = «sans»>
      This paragraph uses a sans-serif font
    </p>

    <p class = «big»>
      This paragraph uses a larger font
    </p>

    <p class = «center underline»>
      This paragraph is centered and underlined
    </p>
  </body>
</html>
```

There's not too much going on here, because most of the work happens in the CSS. Here are the things to notice:

- **The page is mainly a series of paragraphs.** There's no real styling in the HTML itself.
- **Styles are indicated by class identifiers.** Each paragraph has a class identifier to specify how it should be styled. You'll find a corresponding class definition in the style sheet.
- **The page calls an external style sheet.** The styles are handled by `textManipulation.css`.
- **One paragraph uses more than one style.** The last paragraph actually combines two classes. It calls both the center and underline classes.

The style sheet is where all the fun stuff happens. It uses a number of CSS rules to clarify how the various paragraphs should be styled. Look at the overall code; then I break it down to show the details.

```
.italic {
  font-style: italic;
}

.bold {
  font-weight: bold;
}

.underline {
  text-decoration: underline;
}

.strikeThrough {
  text-decoration: line-through;
}

.center {
  text-align: center;
}

.right {
  text-align: right;
}

.sans {
  font-family: sans-serif;
}

.big {
  font-size: 200%;
}
```

You can see that I've defined a number of classes here. The class names indicate the various effects, and each class contains a single rule to generate that effect.

Here's how all the various rules work:

- **Setting the font style.** You can set the overall font style with the `font-style` attribute. Valid options for this rule are `italic`, `normal`, and `oblique` (tipped backward).
- **Changing text weight.** You can specify how much weight (boldness) to apply to text with the `font-weight` attribute. The most common values are `bold` and `normal`.

- **Managing text decoration.** The `text-decoration` attribute can modify a number of effects, but it is normally used to add a line to text. The most commonly used values are `underline`, `overline`, `line-through`, and `none`.
- **Handling text alignment.** Text alignment is normally controlled through the (aptly named) `text-align` attribute. Most common values are `center`, `left`, `right`, and `justify`. Note that this attribute is only used to align text *inside* an element. If you want to center an entire element (say a paragraph or table), look ahead to the `margin` attributes described later in this chapter.
- **Managing fonts.** You can specify a font to display with the `font-family` attribute. This can be used to specify any font on your system, but users will not be able to see these fonts if they aren't installed. See the sidebar "Font frustration" later in this chapter for an overview of font names and how to work with them.
- **Changing font size.** The size of your text can be specified with the `font-size` attribute. This attribute can be measured in many ways, but the safest approach for Web development is to specify percentage of the base font. To make text twice as large as normal, set its `font-size` to `200%`. (You can use traditional measures such as points, but they have less meaning and reliability in the Web setting than they do in standard print application.)

Font frustration

Fonts cause a lot of headaches for Web developers. Computer fonts were designed with word processing in mind. When you install a font, it is registered to your local operating system, and you can use it in the programs installed on your computer. When you print a word processing document, the font information is transferred to the image on the paper, but the font itself stays on the computer. Web development is a bit different, because the document is transmitted across the Web to a distant computer. That computer might or might not have the same fonts installed on it as yours. (Probably it doesn't.) While there are some good solutions proposed in the next version of CSS (CSS3) they aren't workable in current browsers. A few font names, however, are guaranteed to work (at some level) in any browser: serif, sans-serif, cursive, fantasy, and monospace.

When you specify a font family, you can provide a list of fonts. The browser will try each font in the list until it finds something it can use. For example, if you really like the Tahoma font in Windows and want the browser to use something like it, you can use the following code:

```
font-family: "Tahoma",
   "Geneva", "sans-serif";
```

This code tries to find Tahoma (which is present on most Windows installations); if it can't, it looks for Geneva (similar to Tahoma but found on Macs). If neither of these searches works, it applies whatever sans-serif font it knows about.

Joining the Border Patrol

It's possible to draw a border around an element. This is a potentially useful design element, but it can also be very helpful when debugging a page layout. There are three main border properties:

- `border-width`: Specifies the width of the border. This can use the standard CSS measurement schemes, but borders are usually measured in pixels (px).
- `border-color`: Determines the color of the border. Border color is specified with a color name or hex value.
- `border-style`: Specifies a pattern for the border.

Figure BC2-7 shows the possible border styles.

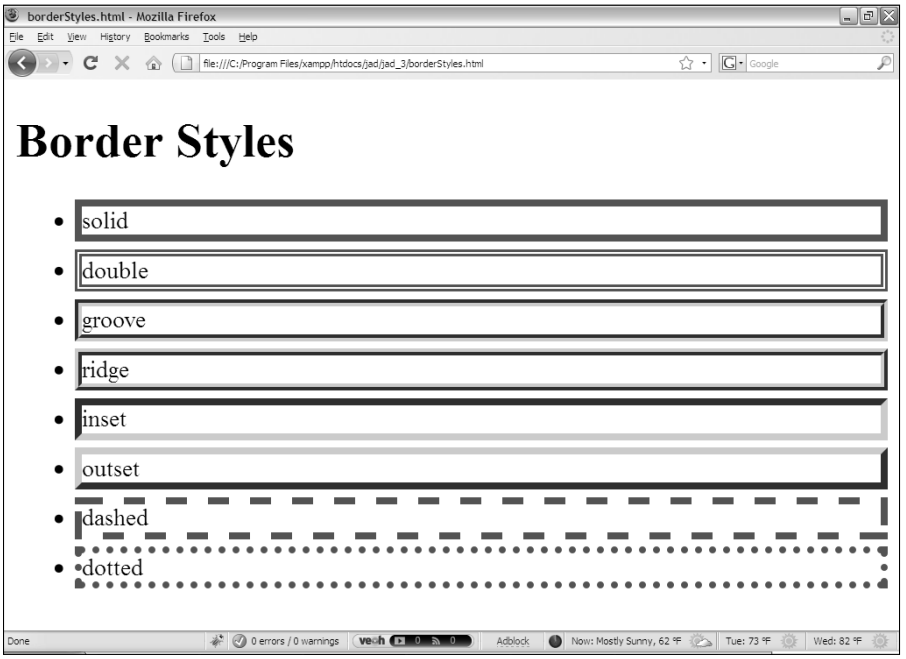

Figure BC2-7: Here are all the border styles.

Generally, the various border attributes are combined into the single `border` property, which allows you to specify width, style, and color all in one. For

example, to specify a 5-pixel blue double border on your paragraphs, you could use the following code:

```
p {
  border: 5px double blue;
}
```

One more handy trick is to isolate the various parts of the border to get lines. For example, you can specify `border-top` to draw a line above an element and `border-right` to draw to the right of the element. Each of these mini-borders can be given the same list of value as the standard border.

Adding background images

You can add a background image to any element. The `background` attribute has a slightly different format than some of the other elements you've seen so far. Figure BC2-8 illustrates a page with a background image.

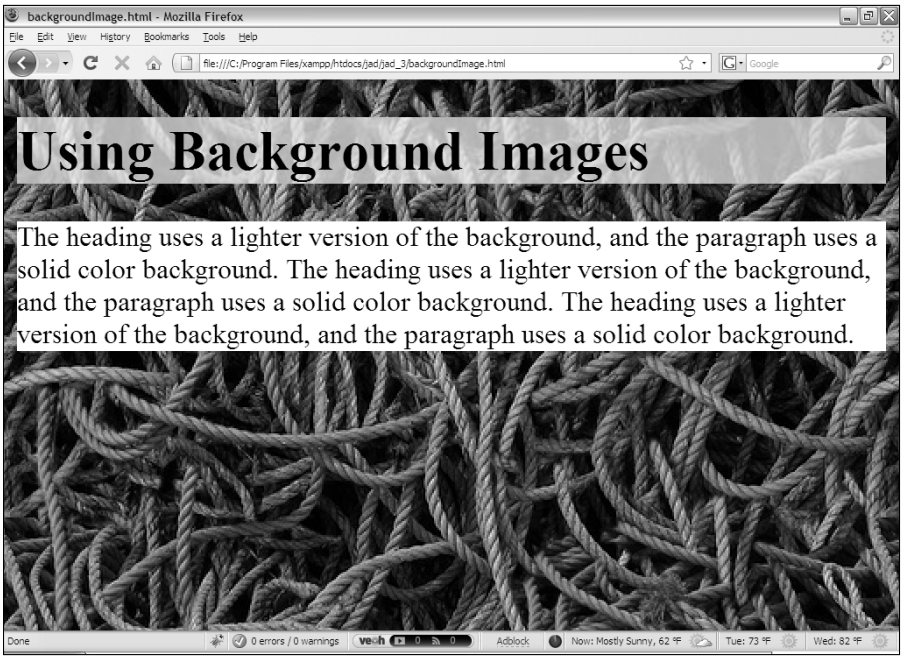

Figure BC2-8: The page has a background image, and so does the headline.

Background images are added through CSS:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang=»EN» dir=»ltr» xmlns=»http://www.w3.org/1999/
          xhtml»>
  <head>
    <meta http-equiv="content-type" content="text/xml;
          charset=utf-8" />
    <title>backgroundImage.html</title>
    <style type = "text/css">
      body {
        background-image: url("ropeBG.jpg");
      }
      h1 {
        background-image: url("ropeBGLight.jpg");
      }
      p {
        background-color: white;
      }
    </style>
  </head>

  <body>
    <h1>Using Background Images</h1>

    <p>
      The heading uses a lighter version of the
           background,
      and the paragraph uses a solid color background.
      The heading uses a lighter version of the
           background,
      and the paragraph uses a solid color background.
      The heading uses a lighter version of the
           background,
      and the paragraph uses a solid color background.
    </p>
  </body>
</html>
```

The key to adding background images is the `background-image` attribute. Here's how you use it:

1. **Identify the image you wish to use.**

   Choose a background image carefully. Make sure the image supports the ideas you're trying to communicate and doesn't distract from your message. Any of the standard Web formats (PNG, GIF, or JPG) is fine. You might want to adjust your image in an editor (IrfanView and Gimp are excellent free options) to ensure it's the right size and resolution.

2. **Place the image in the same directory as your page.**

   Although this step isn't absolutely necessary, it's much easier to manage images if they are physically close to your page. That way when you move the page to a server, you can easily move the associated images as well.

3. **Build your page as you normally would.**

   Create the XHTML code as you normally do.

4. **Add a background image to the body CSS.**

   This will apply an image to the entire page. Of course, you can also apply images to any other element you want.

5. **Specify the URL of your image.**

   The value of the background image has a unique syntax. You must specify that you're invoking a URL, so if the image is `background.png`, the value of the `background-image` attribute will be `url("background.png")`.

6. **Consider modifying the background.**

   You can change the background image by using a number of other CSS attributes: `background-repeat` allows you to control how the background repeats, and `background-position` lets you manipulate the position of the background.

7. **Test your page.**

   Make sure your background image is not too distracting.

Background images can be problematic. Take a look at Figure BC2-9 for an example of this phenomenon.

Inappropriate background images are one of the most common beginner mistakes. Consider the number of Web pages you've seen that have unreadable text.

Most interesting photos have a lot of contrast. This is great for a picture that's meant to grab the user's attention, but it's a problem when the image is supposed to be in the background. High contrast grabs the user's attention, which is a problem when the user is trying to read text. There are a couple of standard solutions to this problem. You can either provide lower-contrast versions of your background images, or you can use plain colors as the background of elements that feature text.

If you look at Figure BC2-9 you'll see that I use both these tricks. The entire page has the rope background. (Thanks to Julian Burgess for the great image.) Note, however, that the title has a lighter version of the image, and the paragraphs use a solid-color background.

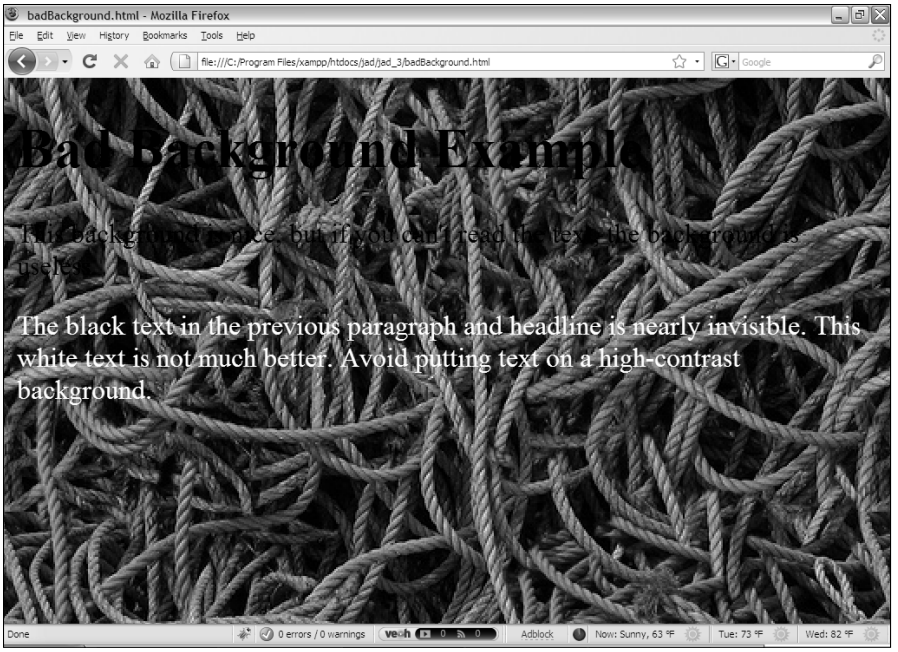

Figure BC2-9: Make sure you can still read the foreground!

You can create a lower-contrast version of an image using a tool such as IrfanView. (Use the adjust colors to make an extremely dark or extremely light version of your image.) Use a darker background with lighter text, or a lighter background with darker text.

Using Float Positioning

Page layout has long been one of the biggest weaknesses of HTML. Table-based hacks used to be the best way to get a page to act correctly, but now CSS provides a number of useful tools for managing the position of elements.

Floating position is extremely flexible but a bit challenging to understand. Once you get the idea, you can use floating positions to set up a page that works very well on a variety of browsers. As an example, think of a standard HTML or XHTML form. Figure BC2-10 shows a typical form with no CSS applied.

Figure BC2-10: This page has no CSS at all.

The form has all the necessary features, but it is ugly. Take a look at the HTML code to see how it's formatted:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang=»EN» dir=»ltr» xmlns=»http://www.w3.org/1999/
          xhtml»>
  <head>
    <meta http-equiv="content-type" content="text/xml;
          charset=utf-8" />
    <title>formNoStyle.html</title>
  </head>

  <body>
    <form action = "">
      <fieldset>
        <label>Name</label>
        <input type = "text"
               id = "txtName" />
        <label>Address</label>
        <input type = "text"
               id = "txtAddress" />
        <label>Phone</label>
        <input type = "text"
               id = "txtPhone" />
        <button type = "button">
          submit request
        </button>
      </fieldset>
    </form>
  </body>
</html>
```

There are a number of important features to note about this code:

- **It supports a form.** `Form` elements are critical in JavaScript coding, and you'll be building a lot of forms in your travels.
- **The form has a `fieldset` container.** Most of the form elements are `inline` elements (that is, they must exist inside a block element). The `fieldset` tag is a special block element designed to live in a form, so it's a perfect container for `form` elements.
- **The form has a number of labels and inputs.** Most forms have this general structure: labels to indicate what the user is to enter and input elements to accept the user input. Each row is typically a label-and-input pair.
- **The `<label>` tag describes labels.** This is a relatively new development in HTML. The `<label>` tag doesn't have any formatting associated with it, so it was not used traditionally. With CSS, you can provide whatever formatting you want.
- **The last element is a button.** Most forms include one or more buttons. The real action happens when the user clicks a button. Because the

button will have different formatting than the `input` elements, I use the `<button>` tag to describe it.

- **No formatting is described in the HTML.** The HTML code simply describes the *intention* of the various elements, not their formatting. CSS will handle that.
- **The HTML is self-explanatory.** You can tell what everything is just by looking at the code. There's no code here that isn't directly related to the purpose of the form.
- **It's kind of like a table.** The general structure of the form looks a bit like a table, but not quite. The goal of the CSS is to take this very clean data structure and make it look *visually* like a table without having to muddy the HTML code with actual table tags.

When you look at Figure BC2-10, it's clear that the browser is not displaying the form in a way that's acceptable. Typically we want forms to look more like a table. Of course, you can embed an HTML table into the code, but that's a lot more work (and complexity) than you need. CSS provides a simpler solution.

Understanding the display types

To understand how this works, you need to understand a little about how Web browsers manage page layout.

It takes very little CSS code to turn the form into a basic table-style format, but the code can be mysterious. The secret has to do with the way HTML lays out pages. Essentially, a Web browser can lay out Web elements in three different ways:

- `Inline`**:** Place the element exactly where you would place the next character of text.
- `Block`**:** The element is basically independent and gets its own line. `Block` elements (such as `H1` tags and paragraphs) typically have line breaks before and after themselves.
- `Alternative`**:** Some special CSS attributes remove elements from the normal layout scheme (at least to some extent) and apply different placement rules to them. The `float` attribute described in this section is one example.

All of the HTML tags have their own default display mechanism (inline or block). You can alter the way a tag is displayed by changing its `display` attribute. You can also add an alternative placement scheme by changing other CSS attributes. That's how you can make a form look and act like a table without needing `table` tags.

Having a block party

The first step is to define some of the elements as block-level using the `display` attribute.

Take a look at Figure BC2-11 to see how this is done.

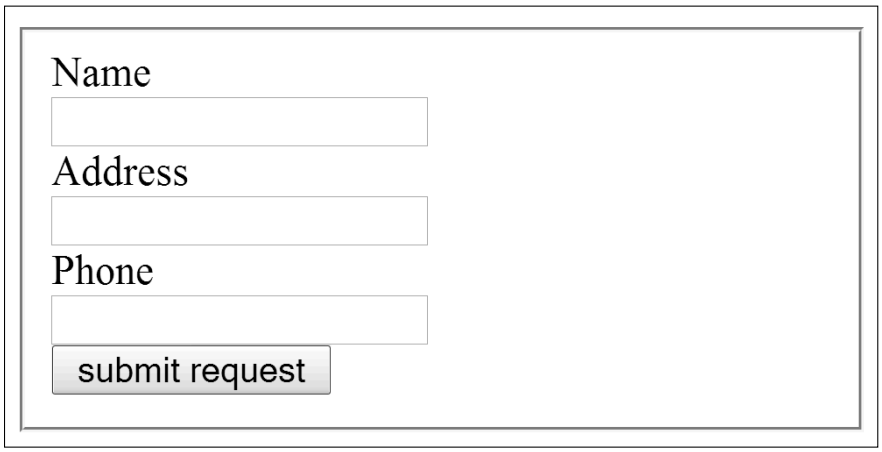

Figure BC2-11: The elements are all stacked up on top of each other.

The HTML in `formBlock.html` is no different than the HTML in `formNoStyle.html`. The only difference is the inclusion of an external style sheet: `formBlock.css`.

The code for `formBlock.css` is pretty simple:

```
input {
  display: block;
}

button {
  display: block;
}
```

All it does is specify that buttons and `input` elements (the text boxes) should be block-level elements. This forces the page to a stacked look, but more importantly, it sets the stage for a nicer layout.

Note that I changed the text manually (by pressing Control-+) on all the figures in this chapter to make them easier to read in the book. The CSS doesn't change the size of the text, but of course you can use it to do that if you want.

Floating to a two-column look

This is a starting place, but you really want the labels to be to the left of the corresponding block. The `float` attribute can be used to create exactly this effect, as you can see in Figure BC2-12.

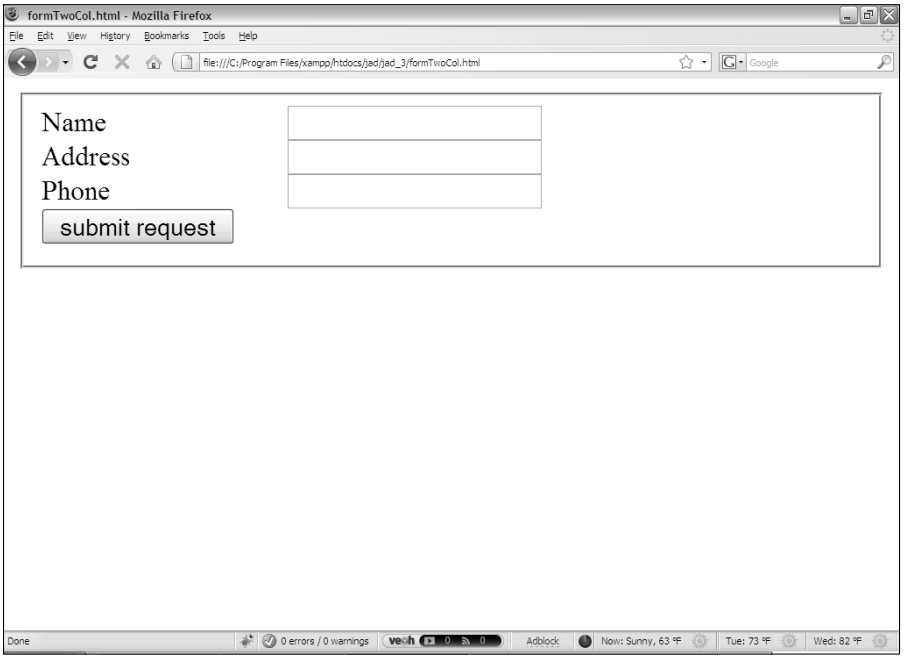

Figure BC2-12: The labels are floated, giving a two-column effect.

The `float` attribute allows you to remove an element from the normal layout rules and apply a special floating behavior. The `float` attribute describes the relationship between an element and its neighbors. In this case, I tell each label to float to the left. (You can also float to the right, but this is rarely done in practice.) This causes the label to be immediately to the left of the corresponding input element. Adding a width to the floated label makes the input elements line up nicely (looking and acting like a table with no additional HTML code).

Here's the code for `formTwoCol.css`:

```
label {
  float: left;
  width: 30%;
```

```
}

input {
  display: block;
}

button {
  display: block;
}
```

Cleaning up the form

Of course, there's a lot more you can do with the CSS to make things look better. `formFloat.html` in Figure BC2-13 shows a nicely formatted form.

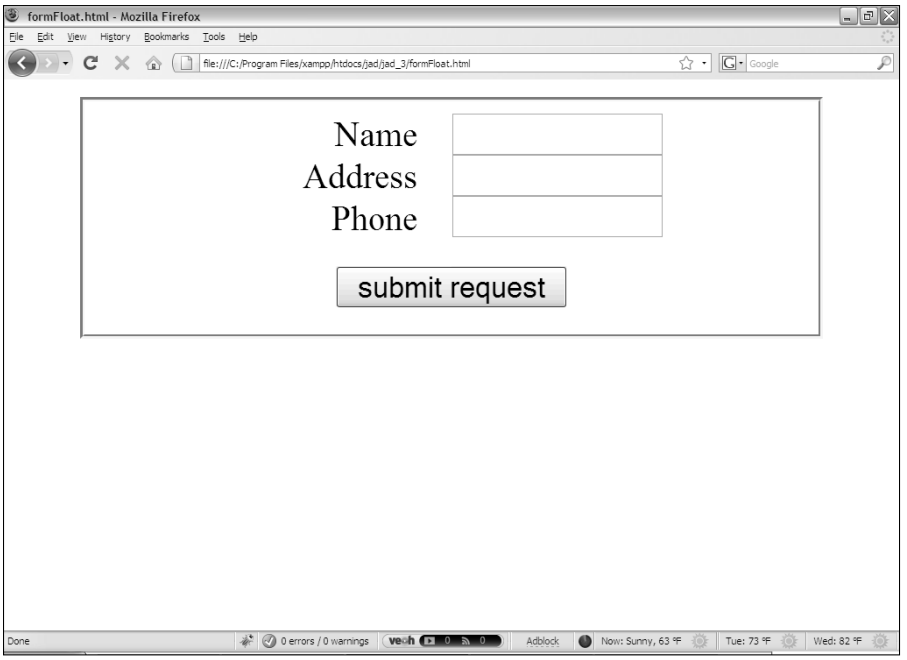

Figure BC2-13: Now the form has very nice formatting.

A few more CSS attributes are used to tweak the form's appearance:

- `margin`**:** The `margin` attribute describes what margin occurs outside the boundary of an element. You can define all margins with the plain

margin attribute, or you can specify individual margins (margin-left controls the left margin, for example). If you set the margin attribute to auto, you will center the element horizontally. (There is no easy way to do vertical centering in CSS.)

- padding: The padding attribute specifies the space between the content of an element and its boundary. Padding is used to fix text that is crowded too close to a border.
- text-alignment: The text-alignment attribute is used to manipulate the content of an element (use the margin attribute to center the element itself).

Take a look at the code for formFloat.css and then I explain how it works:

```
fieldset {
  width: 80%;
  margin: auto;
}

label {
  float: left;
  width: 30%;
  text-align: right;
  padding-right: 1em;
  margin-left: 15%;
}

input {
  display: block;
  width: 30%
}

button {
  display: block;
  margin: auto;
  margin-top: 1em;
}
```

This version of the CSS still works on exactly the same HTML as the previous examples. It adds a few formatting attributes to clean up the page and get a better-looking form. Here's how to build this type of form layout:

1. **Begin with a two-column layout.**

   Begin by building the simple two-column layout described in the previous section.

2. **Center the** fieldset**.**

   The fieldset is a block-level element by default, which is what you want. Block-level elements typically take up 100% of their container's width, so if you want to center a fieldset (or any other block-level element,) you need to make it narrower and set the margin to auto.

3. **Right-justify the labels.**

   I think it's easier to enter data in a form if the label is very close to the text box. For that reason, I usually right-justify the labels. Set the labels' `text-align` attribute to `right` to achieve this effect.

4. **Pad the labels a little bit.**

   When the `text-align` attribute is set to `right`, the labels seem to crowd the input elements a bit. Add a little bit of `padding-right` to the labels to give them a little breathing space. (`1em` is the width of the widest character in the current font.)

5. **"Center" the labels and input elements.**

   You can't exactly center the label and input combination, because they're two different elements on the same line. However, you can use percentages to get the same effect. If the label and input are both set at 30% width and the left margin of the label is 20%, your `label` and `input` elements will be centered within the `fieldset` element. However, if you right-justify the labels (as I tend to do), the form looks better if you drift it a little more to the left. I actually set the `margin-left` of the label to 15% instead, because I think it looks better.

6. **Center the button.**

   Although buttons can be created as HTML `input` elements, I tend to use the `<button>` tag instead. Buttons usually have different styles than input elements (because they don't require labels) so making them different HTML elements makes life easier. To center the button, just set its `display` attribute to `block` and the `margin` to `auto`. I find the button needs a little more vertical space, so I add a little `margin-top` to make it look a little better.

Of course, you can do much more to make your forms look better. You can add colors, background images, and custom fonts if you wish. The important idea here is to let CSS handle all the formatting so your pages can look good with the cleanest possible HTML code. Separating the CSS from the HTML will make your life a lot easier when you start writing JavaScript code to manipulate the page. (That will be very soon, I promise!)

Using absolute positioning

CSS allows some other useful mechanisms for positioning elements. The *absolute positioning* scheme is especially useful, as it allows you to have much more precise control of the position of CSS elements. When you specify that an element will use absolute positioning, you completely remove it from the normal inline and/or block calculations, and you are expected to specify the exact position of the object yourself.

This makes the absolute positioning scheme very powerful, but often too tedious for general layout. If you rely on absolute positioning to set up a page, you generally have to use the technique for every element on the screen. Absolute positioning techniques are best used for specialty objects that can ignore the rest of the page layout scheme. I use it mainly for creating moving objects that are animated with JavaScript (see Chapter 8 for complete instructions on how to achieve this effect).

Figure BC2-14 shows an example of absolute positioning.

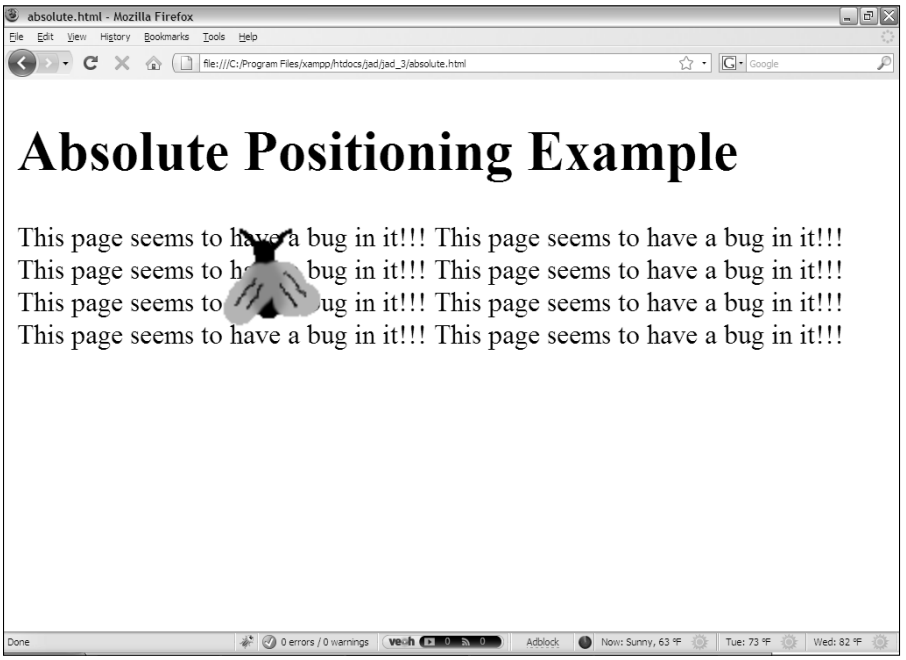

Figure BC2-14: The bug isn't following the normal layout rules!

The bug shown in Figure BC2-14 is sitting on top of the paragraph. This effect is possible with absolute positioning.

When you use absolute positioning, you manually specify the position of the element. Here's the HTML for the `absolute.html` demonstration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="EN" dir="ltr" xmlns="http://www.w3.org/1999/
          xhtml">
  <head>
    <meta http-equiv=»content-type» content=»text/xml;
          charset=utf-8» />
    <title>absolute.html</title>
    <link rel = «stylesheet»
          type = «text/css»
          href = «absolute.css» />
</head>

<body>
  <h1>Absolute Positioning Example</h1>
  <p>
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
    This page seems to have a bug in it!!!
  </p>

  <p id = «bug»>
    <img src = «bug.gif»
         alt = «bug picture» />
  </p>

</body>
</html>
```

This page contains an ordinary paragraph and a second paragraph named "bug." The bug paragraph contains only an image of a bug. Note that images are inline-level tags, and they must be embedded within a block-level tag to make the page validate. That's why the image is inside another element, and it's the paragraph element that will be manipulated.

If there were no CSS, the page would simply display the bug image as its own separate paragraph after the ordinary (text-laden) paragraph. However, the CSS file changes things:

```
#bug {
  position: absolute;
  left: 100px;
  top: 50px;
}
```

The CSS changes the behavior of the element named `bug` in a few important ways:

- **The `position` attribute is set to `absolute`.** This means ordinary layout mechanisms are overruled by specific position information.
- **The `left` attribute is set to 100 pixels.** After you've assigned absolute positioning to an element, you're committed to specifying its top and left positions. Normally you set the absolutely positioned elements by using pixels (`px`).
- **The `top` attribute is set to 50 pixels.** This will force the object's upper left corner to be (100, 50) pixels from the upper-left corner of the document.
- **The absolutely positioned element will obscure traditional elements.** Anything placed with absolute positioning will ignore previously positioned elements. This can be a problem in ordinary Web design, but in animation, it can be a nice feature. (For example, you can make the bug fly around the screen with JavaScript tricks.)

There is much more to CSS positioning than I can describe in this introductory chapter, but the basic tools you learn here can be used in most JavaScript and AJAX applications to vastly improve the look and behavior of your pages. If you want to investigate CSS positioning in more detail, please check out one of my other books: *HTML, XHTML, and CSS All-in-One Desktop Reference For Dummies*. I have hundreds of pages in that book dedicated to explaining multi-column layouts, drop-down menus, and other CSS goodness.