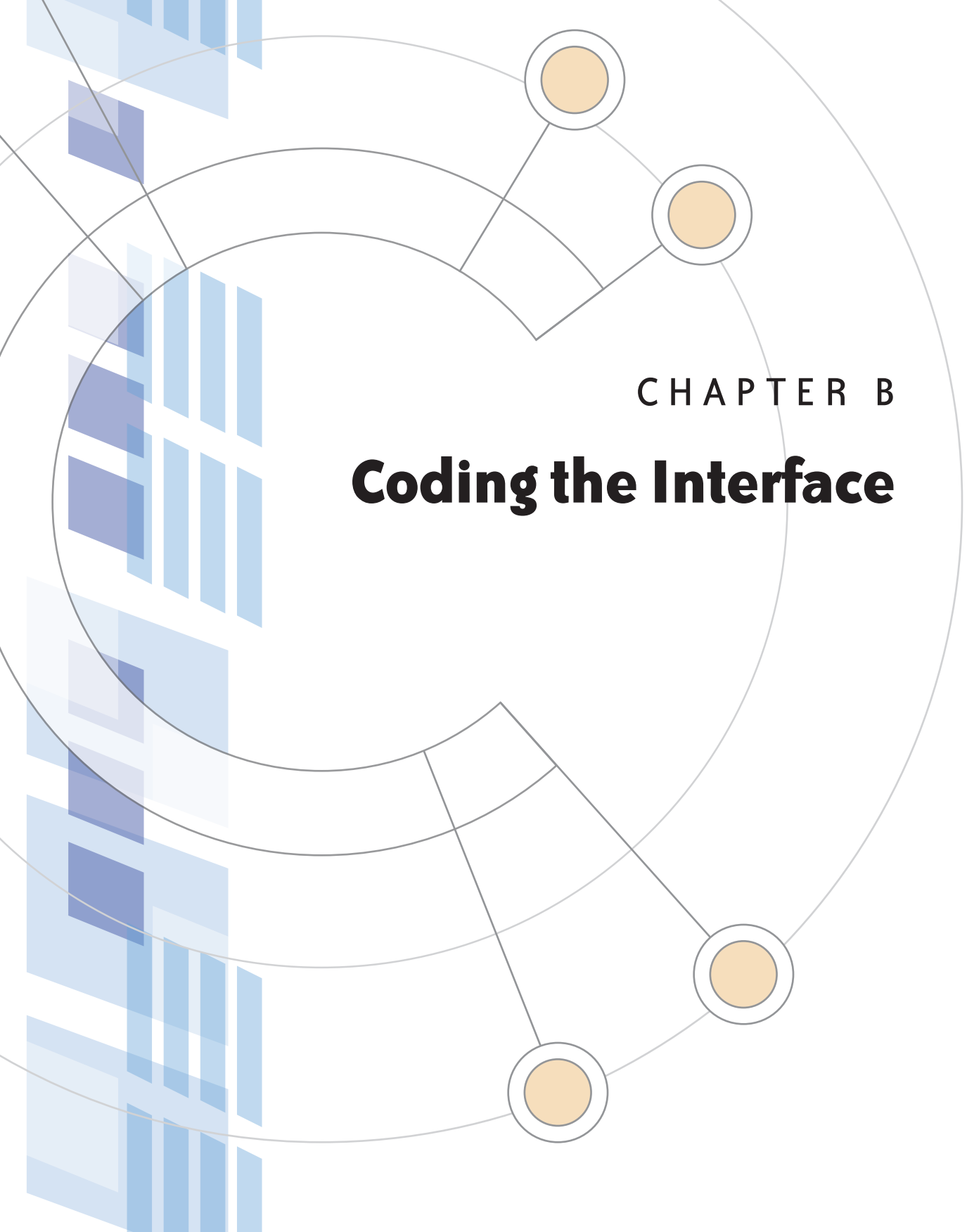


CHAPTER B

Coding the Interface

If you are working in the world of Microsoft Windows, note that .ASP and .NET, the two most widely used Windows-based middleware products, and SQL Server, the Microsoft database, use the same coding principles, and you need only read a book on these products to see the conceptual similarity between coding for the LAMP and Windows platforms.

To make *Codin' for the Web* usable for as many people as possible, Ithe examples in the book and at its companion Web site use open source software products. Open source software is generally developed collaboratively by loose affiliations of programmers and is distributed for free. I will discuss creating a Web site using open source technologies collectively known as LAMP:

- **L**inux system software
- **A**pache Web Server
- **m**ySQL database
- **P**HP middleware

All of these products can be downloaded for free from the Web and together form a robust, well-tested, and thoroughly documented platform for Web development.

Because we are discussing how to write code, not install and configure software, we are not going to discuss the installation and configuration of a LAMP platform here. If you are part of a large organization, you probably have IT people who handle those details for you, and if you are an individual just getting into Web development, you may simply want to use a hosting service that offers PHP and mySQL as part of their hosting packages.

Many Web hosting companies, including GoDaddy and 1to1, offer low-cost hosting plans that include PHP and mySQL. If you sign up for one of these plans, you can download, install, and run project examples that are available here on the *Codin' for the Web* Web site (www.bbd.com/codin).

If you have a computer that is connected to the Internet and has a static IP address and you want set up your own server, you can go to http://www.apachefriends.org/en/xampp.html and download the XAMPP package. It is a one-click installer for all four LAMP components, and it includes an administration tool for mySQL so that you can build and edit database tables.

XAMPP is available for Mac, Windows, and Linux, although the Mac implementation is still a beta version at the time of writing. If you are familiar with working on the command line in Windows or in Terminal on Unix-based machines such as the Mac, you will probably have no problems getting the installation configured to your needs; otherwise, you may want to get access to a system that is already set up.

Writing Code

You can code an entire Web site with a basic text editor such as WordPad in Windows or TextEdit on the Macintosh—code is just plain text, after all—but there some excellent tools that provide all kinds of capabilities to help you. Adobe Dreamweaver is my Web programming tool of choice (**Figure B.1**).

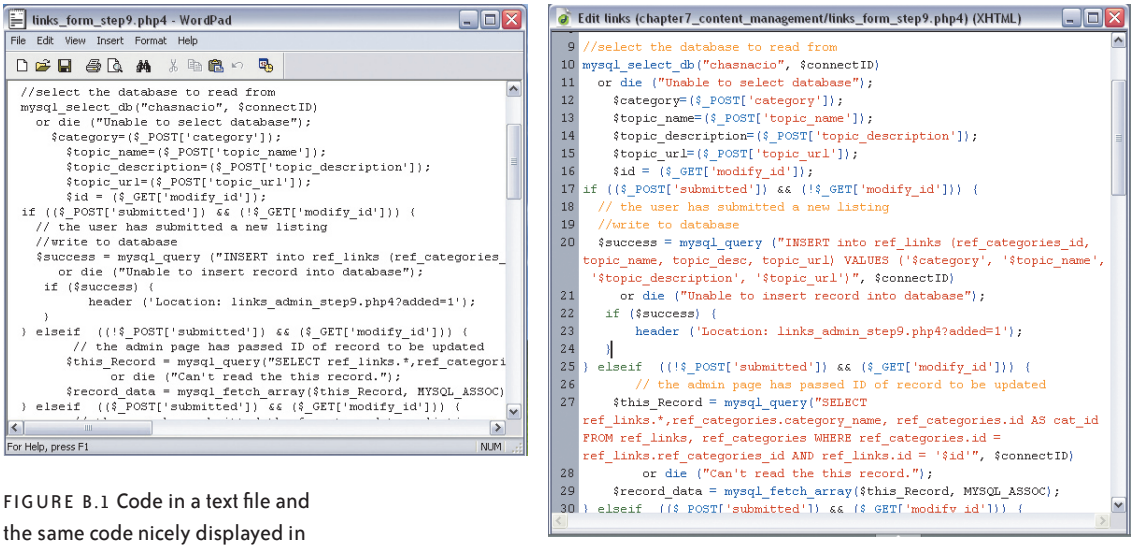


FIGURE B.1 Code in a text file and the same code nicely displayed in Dreamweaver.

Dreamweaver colors the different elements of the code and wraps the code nicely in the window so you never have to scroll sideways.

Some purists disdain Dreamweaver because of its WYSIWYG Layout view, but I like Dreamweaver because of its code management features, including the capabilities to do the following:

- Automatically color the different types of elements of the code as you write, for easy identification and readability.
- Highlight many kinds of errors as you type, so you can debug as you go.
- Insert common snippets of code that you can select from menus as you work, saving lots of repetitious keystrokes.
- Pop up a related list of attributes you can pick for each tag as you write it, saving you from guessing about which attributes you can use.
- Format your code so its underlying structure is visually presented. For example, if you write a loop (a piece of code that

processes several pieces of similar data), Dreamweaver automatically indents the lines within the loop, so you can clearly see where the loop starts and ends. If you don't appreciate this feature now, you certainly will once you start to code.

- Use a built-in FTP client so you can move you files easily onto your Web server.

I highly recommend investing in Adobe Studio, which includes Dreamweaver, Flash for animation, and Fireworks for graphics, providing a comprehensive and integrated suite of tools for your site development work.

Once you have your site set up in Dreamweaver and have verified that you can successfully upload files to the Web server and display them in the browser, it's time to start building your site.

First, we'll take a look at the basic principles of XHTML and CSS.

Using XHTML and CSS

This discussion of XHTML and CSS is not a comprehensive tutorial. Many readers will have some familiarity with XHTML and CSS, and the focus of *Codin' for the Web* is on PHP, whose decision-making capabilities and intelligent objects are the hallmarks of "real" programming languages. Therefore, we are going to look at XHTML and CSS only to a level that ensures that all readers are up to speed. We will cover the basics, focusing on programming XHTML and styling that XHTML with lean CSS to achieve extensive control over the presentation of our content without filling our XHTML markup with extraneous presentation code.

If you want a more in-depth look at XHTML and CSS, I refer you to my book *Stylin' with CSS*, also published by New Riders.

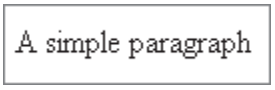
A Web page is made up of XHTML elements. Elements either contain or reference the content we want to display on the page. We define each element—for example, as a paragraph, an image, a check box, or a link—with an XHTML tag.

Here's a paragraph:

```
<p>A simple paragraph</p>
```

Because a browser applies only very basic styling to markup, this tag results in a dull-looking paragraph (**Figure B.2**).

FIGURE B.2 Default browser styling of a paragraph tag.



XHMTL is the mechanism by which you define the *structure* of your site's content, so that browsers and other user agents can display it in an appropriate manner. XHTML enables you to indicate to the user agent what each element is: a heading, a paragraph, a link, a list, and so on. Every browser has a basic built-in style sheet that enables it to display each of these in an appropriate manner.

User agent *is the generic term for browsers and a multitude of other devices on which the Web is now viewed, such as cell phones and PDAs.*

Using CSS, you can determine the *presentation* of that XHTML structure. Instead of using the browser's default text styles (such as headlines in a large Times font) and page layout (straight down the page in one column), you can select other typefaces and type sizes for text and create multicolumn layouts. For instance, CSS lets you associate a list of properties that define elements such as these:

```
p {font-style:italic; font-weight:bold; color:red; border-
bottom:2px dashed blue;}
```

This results in the more eye-catching paragraph (**Figure B.3**).

FIGURE B.3 Styling applied to a paragraph element.

Writing Good XHTML

What happened to HTML, you ask? The answer is that Extensible Markup Language, or XML, came along. XML is a format for moving data between different applications. XHTML is a reformulation of HTML that complies with the XML standard. It follows stricter rules than HTML, however: rules consistent with XML.

Writing good XHTML enables user agents to display your content correctly. HTML was very forgiving of sloppy coding: you could indicate the end of a content element such as a paragraph by simply writing a tag for the next element. XHTML is not so permissive; its rules require that you indicate the end of every element before starting the next. This may seem cumbersome to those of us who got away with the old methods for years, but today a Web site's content might be displayed on multiple user agent types, and perhaps syndicated via XML and RSS feeds, so we need to be more explicit in marking up our content to ensure that it is always displayed correctly.

Most important, following the rules of XHTML provides a structure to a document that complies with the *Document Object Model* (DOM), which defines how underlying page structure should be written.

According to the World Wide Web Consortium, the guiding body of the Web's development, the DOM is "a platform- and language-neutral interface that allows programs and scripts to update the content, style and structure of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page." In other words, the DOM provides a structure that allows the code in the Web browser to be changed, or appear to be changed, without having to refresh the page.

CSS and JavaScript are both designed to work within the framework of the DOM. With a correctly formed page structure that meets the requirements of the DOM, CSS and JavaScript can accurately target any XHTML element, enabling you to set and change its properties. This lets you use CSS to position and style onscreen elements, and use JavaScript to enable onscreen elements to respond to user actions and even modify the markup itself without reloading the page.

The key to a well-coded Web site, therefore, is the structure of the XHTML—it's the ironwork of the building on which everything else hangs.

When writing XHTML, you need to make sure that the code you write is valid and well formed. *Valid* code complies with the rules of XHTML; you can (and should) check that a page is valid by uploading it to your server and then typing its URLs in the validator at http://validator.w3.org. Using the W3C validator is also a great way to test and debug your markup.

Well-formed XHTML code is structured correctly according to the markup rules, described in the following pages.

Writing XHTML Tags

As noted, the purpose of XHTML is to indicate the nature of each piece of content and its position in the structure of the document. To do this, you enclose each element of the content (heading, paragraph, list item, and so on) in a pair of tags, known as the opening and closing tags. A tag consists of an abbreviation of the element name within angle brackets (also known as the less-than < and

greater-than > symbols). If we wanted to mark up the text "All about XHTML" as a heading, we would write it like this:

```
<h1>All about XHTML</h1>
```

Note that the closing tag is identical to the opening tag except that it has a forward slash as its first character. These tags are called enclosing tags as the content is sandwiched between the opening and closing tags.

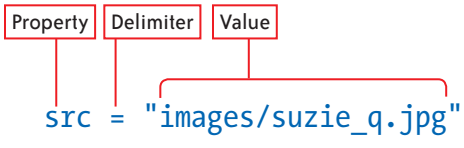
Some tags are nonenclosing: they don't enclose content, but simply reference it. Here's an example:

```
<img src="images/suzie_q.jpg" alt="my friend Suzie" />
```

Note that while there is not a separate closing tag, there is a forward slash before the closing angle bracket, indicating that the tag is closed. If you don't include the forward slash, your code won't validate.

Specifying Tag Attributes

In the preceding example, the image tag has two attributes: `src` and `alt`. Attributes provide additional information relating to the tag and are structured in this example in a typical programming format: property, delimiter, value.



The property is the name of the attribute, the value is what the property is set to, and the delimiter is a symbol (in this case, an equals sign) that separates the property from its associated value. The two attributes associated with this image are the source of the image (its location on the server, provided so that the browser can retrieve and display it) and the alternative text (which the browser will display if the image fails to load, or which will be read aloud if the site is being browsed by a user using a screen reader).

Until recently, tags also include presentation attributes, such as `FONT` and `COLOR`, to create the desired onscreen appearance of the element. Today, we avoid adding such attributes to the markup and instead link a style sheet containing the presentation information to the purely structural XHTML document.

Rules for XHTML Markup

There are nine simple rules for writing XHTML.

1. Declare a DOCTYPE.

   Example:

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

   A DOCTYPE is usually the first line of XHTML in a Web page, and it lets the browser know what it is dealing with—HTML, XHTML, or a bit of both. In the reality of today's Web design, there are really only two DOCTYPES that matter: `Strict`, meaning that the page is pure XHTML, and `Transitional`, meaning that older, deprecated (phased out but still valid) tags such as `FONT` might be present in the page.

2. Declare an XML namespace.

   Example:

   ```
   <html xmlns="http://www.w3.org/1999/xhtml">
   ```

   Usually, the namespace is a pointer to the World Wide Web Consortium (W3C) Web site where a document type definition (DTD) file can be found that helps the browser interpret the XHTML.

The code for these first three rules is generated automatically by Dreamweaver if you select the XHTML document choice in the New > File dialog box and is part of the templates available at the Codin' *Web site. If you have lots of time and are a very accurate typist, you can write this code yourself.*

3. Declare the content type.

   Example:

   ```
   <meta http-equiv="Content-Type" content="text/html;
   charset=iso-8859-1" />
   ```

   This line tells the browser what character set to use when displaying the page. ISO-8859 is the Latin character set used by English and other European origin language. You will need a different character set declaration if your site is in a language such as Chinese or Farsi.

4. Close every tag, both enclosing and nonenclosing.

   Examples:

   ```
   <p>This paragraph has a closing tag.</p>
   ```

   ```
   <img src="images/spaceship" alt="Apollo 10 rocket" />
   ```

5. Correctly nest all tags.

   Correct example:

   ```
   <p>Correct nesting is <strong>very</strong> important</p>
   ```

   Incorrect example:

   ```
   <p>Be sure <strong>not to do this.</p></strong>
   ```

   Because the `strong` tag opened after the `p` tag opened, it must close before the `p` tag closes. This structure ensures that the tags have a properly formed hierarchy. CSS and JavaScript rely on this hierarchy.

6. Don't put block tags inside inline tags.

   For example, placing a paragraph inside a link, as shown here, breaks this rule:

   ```
   <a href="big_no_no.htm"><p>An invalid piece of code</p>
   </a>
   ```

7. Write tags in lowercase only.

   Correct XHTML:

   ```
   <img src="my dog" alt="Picture of Rover" />
   ```

   Incorrect XHTML:

   ```
   <IMG SRC="My Dog" ALT="Picture of Rover" />
   ```

   Quoted attribute values (such as "Picture of Rover") can contain uppercase letters.

8. Attributes must have values and must be quoted.

   If you want to write valid XHTML (and you do), attributes must be placed within quotation marks.

   Example:

   ```
   <img src="my dog" alt="Picture of Rover" />
   ```

9. Use encoded equivalents (also known as entities) for & and <.

   For &, use `&`

   For <, use `<`

   Because these two characters have special uses (starting entities and opening tags respectively), you must use the encoded equivalents of them in your XHTML.

Entities are strings of characters that create a single character onscreen. They are useful for writing accented characters and symbols such as ® and £. Entities start with an ampersand and end with a semicolon (for example, the copyright symbol entity is `&copy;`). You can find a list of these and other entities at the Web Design Group Web site: www.htmlhelp.com/reference/html40/entities.

Coding the Interface

You can find this page structure template in the Chapter B files at the Codin' *Web site, or you can let Dreamweaver generate it for you when you create a new page.*

CODE B.1: xhtml_template.html

So we can get started building a dynamic site, let's build a page template using XHTML and CSS.

If you want to follow along, create a new XHTML document and save it as `xhtml_template.html` or some similar name.

Let's start by coding the basic XHTML page structure.

```
<?xml version="1.0" encoding="iso-8859-1"?>

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

  <title>Codin' - Sample XHTML Page</title>

  <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />

</head>

<body>

  <!-- comment - content goes here -->

</body>

</html>
```

Adding Comments to Your Code

You can add comments to your code to help you remember what the code does when you review it later. I strongly recommend that you add lots of comments to your code. XHTML comments are not visible to the user unless the user views your source code by selecting View Source from the browser's View menu. Comments within HTML are written like this:

```
<!--  this is an XHTML comment -->
```

Within CSS, they are written like this:

```
/* this is a CSS comment */
```

Because all content the user sees in a Web page goes between the body tags, and there is presently only a comment between those tags, you only see a blank page if you display this page in a Web browser, so let's now add some content between the body tags (**Figure B.4**).

CODE B.2: basic_xmhtml_markup1.html

```
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph of text</p>
  <a href="#">read more</a>
</body>
```

FIGURE B.4 Three XHTML elements displayed in browser.

These three elements are displayed in the Times typeface; the heading is large and bold, and the paragraph is displayed in smaller type. The link is displayed blue and underlined. These styles are set by the browser's internal CSS style sheet. The internal style sheet defines some basic styles for each type of XHTML element. These styles are applied unless you modify them in a style sheet that you author; an author's style sheet loads after the browser's style sheet, as we will see shortly. For now, just remember that the browser has default (preset) styles for each XHTML element that are intended to provide a basic visual and hierarchical layout—for example, each heading, h1 through h6, is styled sequentially smaller than the last, and headings are larger than other text.

The two elements, the heading and the paragraph, sit one above the other. This is not because they are on separate lines in the markup; if you put them on the same line, they would still sit above one another when displayed in the browser. They appear one above the other because headings and paragraphs are both block elements.

About Block Elements

Block elements are by default set to the CSS width property of `auto`, which sets blocks as wide as possible, so they stretch across the width of the browser and change width as the browser width is changed by the user. They also always appear stacked one above another in the browser window, unless you use CSS to change them to inline elements (this is done with the CSS `display` property).

About Inline Elements

Inline elements, by contrast, appear next to each other on the same line and wrap to a new line only if there is no room on the current line. Links and images are examples of inline elements. You can see this inline behavior by adding some links before the two block elements.

CODE B.3: basic_xhtml_markup2.html

```
<body>
  <a href="#">Link 1</a>
  <a href="#">Link 2</a>
  <a href="#">Link 3</a>
  <a href="#">Link 4</a>

  <h1>This is a heading</h1>
  <p>This is a paragraph of text</p>
  <a href="read more"></a>
</body>
```

Figure B.5 shows the results.

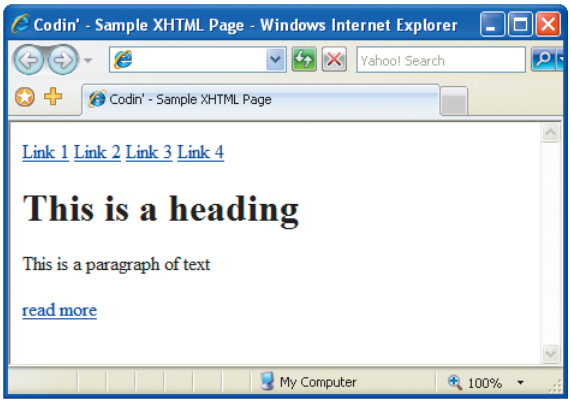


FIGURE B.5 Links are inline elements and appear next to each other on the same line.

Setting Link Placeholders

In these links, I have used the # character in place of a URL for now. This is valid coding and creates a handy placeholder; in dynamic sites, the exact URL—the place to which the link links—is almost always determined later in the programming process and may even be generated by the code itself as the user navigates the content. If you leave the `href` attribute as an empty string—quotation marks with nothing inside—as you might be tempted to do if you are using the link to trigger a JavaScript function, the link will not respond to CSS styles, such as a rollover highlight, when the user interacts with it.

Code Element Boxes

Each element, whether block or inline, is actually a box with its content inside it. Block element boxes default to the width of the browser, and inline element boxes always shrink to fit their contents and so are only as wide as whatever they contain.

We can accurately size and position these element boxes to create page layouts with CSS, and we can style these boxes with borders, margins, and padding to position the content within them.

Both block and inline elements are only as tall as they need to be to enclose their contents.

To see element boxes in action, let's add some CSS styles to our page (of embedded styles) to see where these boxes actually are.

Modify the code between the head tags to look like this:

For now, we are going to add CSS directly to the head of the page we are working on, as it makes simultaneously developing the XHTML and the CSS easier. Later, we will move these CSS styles to a separate style sheet, so the styles can be shared by many pages.

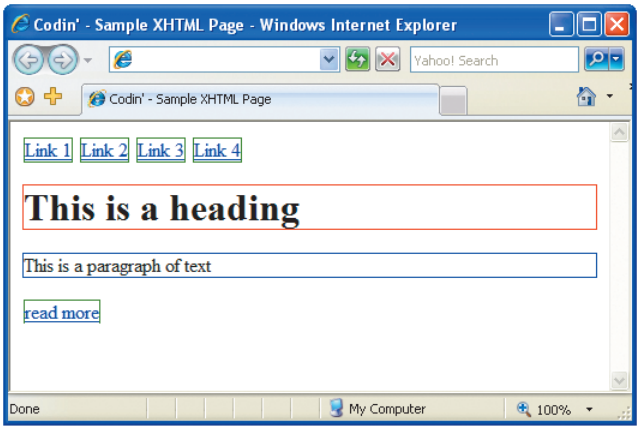
CODE B.4: basic_xhtml_markup3.html

```
<head>
  <title>Codin' - Sample XHTML Page</title>
  <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<style type="text/css">

a {border:1px solid green;}
h1 {border:1px solid red;}
p {border:1px solid blue;}

</style>
</head>
```

The `style` tag with the `text/css` attribute tells the browser to stop treating the code as XHTML and to treat it as CSS. When the style tag is closed, the browser reverts to interpreting the code as XHTML. This is a first taste of how we can mix programming languages in our work to achieve the desired onscreen result.

Inside the `style` tag, you can see three styles that set one-pixel-thick borders for each of the three kinds of tags in our markup (**Figure B.6**). Each has a different color.

FIGURE B.6 Borders around the elements.



Note that these boxes are vertically separated. What stops them from touching are the margin styles that the browser's style sheet is applying to them. Margins create a buffer of space around an element's box. One of the first things I do when writing CSS is get rid of these default margins that are applied to almost every element; I plan to style this XHTML myself, and I'll decide what has margins, thank you very much.

Margins create space between element boxes, and padding creates space between an element box and its content.

The way to remove these default margins is to use the CSS selector * (Shift-8), which means select "any element." While removing the default margins, I also remove the default padding, with one very short line of CSS.

CODE B.5: basic_xhtml_markup4.html

```
<head>

  <title>Codin' - Sample XHTML Page</title>

  <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />

<style type="text/css">

* {margin:0; padding 0}
```

```
a {border:1px solid green;}
h1 {border:1px solid red;}
p {border:1px solid blue;}

</style>
</head>
```

The elements now not only touch vertically, but also go right to the edge of the browser, because we have removed the margin from the `body` element. The `body` element is always the same size as the browser window and, as you can see in **Figure B.7**, has a small margin applied to it so that an unstyled element does not touch the edge of the browser window.

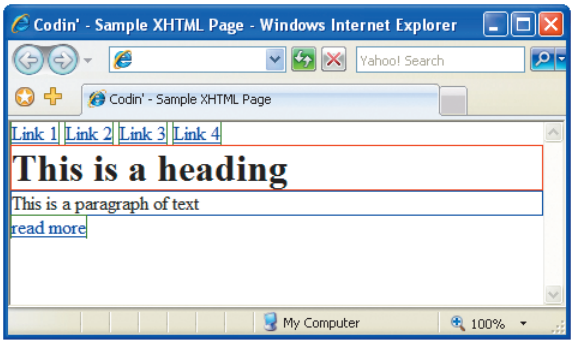


FIGURE B.7 All default margins have been removed from the elements.

Using div Tags to Give Content Structure

You may have wondered, as you browsed a link-rich news site such as CNN or The Onion, how a single page can have so many different link styles. Giving the same element different styles on the same page is achieved by using contextual CSS selectors. These allow us to write styles that state that links within one context have one kind of appearance, and links within another context look different. This context is usually supplied by XHTML `div` tags with `id` attributes.

A `div` (short for division) element can be thought of as a neutral element—that is, it has no default styles and therefore has no visible effect unless we explicitly style it. It's typically used to enclose a number of related XHTML elements.

```
<div id="div_name">
  <!-- a number of XHTML elements go here
</div>
```

Using the id attribute, we can give each div tag, and thereby each section of our markup, a unique name. By referencing the id value, we can target our CSS (and JavaScript) to the elements inside it, without affecting other elements of the same kind elsewhere in the markup.

A div tag creates a box just like other elements and can be styled to, say, provide a border around a group of navigation links that it encloses. Let's see this in action.

CODE B.6: basic_xmhtml_markup4a.html

We added a link after the paragraph so we can see how links in different div contexts can be styled differently.

```
<body>
<div id="navigation">
 <a href="#">Link 1</a>
 <a href="#">Link 2</a>
 <a href="#">Link 3</a>
 <a href="#">Link 4</a>
</div>
<div id="content">
  <h1>This is a heading</h1>
  <p>This is a paragraph of text</p>
  <a href="#">read more</a>
</div>
</body>
```

Now the fun begins.

First, we'll delete the styles that added the borders to the div structures, and then we'll temporarily add a style that puts 20 pixels of margin all around the body so our content isn't right against the edge of the browser window.

```
<style type="text/css">
* {margin:0; padding:0;}
body {margin:20px;}
</body>
```

We'll add font-family and font-style styles for each div structure.

CODE B.7: basic_xhtml4a.html

```
<style type="text/css">

* {margin:0; padding:0;}

body {margin:20px;}

div#navigation {font-family: Verdana, Arial, Helvetica, sans-serif; font-size.8em}

div#content {font-family:Georgia, "Times New Roman", Times, serif; font-size:.75em}

</body>
```

Figure B.8 shows the results.

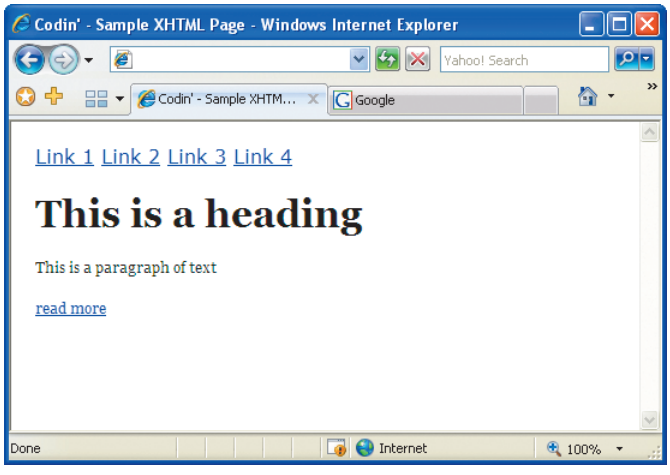


FIGURE B.8 Div tags allows us to organize our XHTML elements into logical groups and style elements of the same kind—links in this case—in different ways.

Font styles are inherited, so all the elements inside the div structure take on these styles once we apply them to the parent div element—we don't have to apply them to every child element. The default browser font size is 1 em, and all our sizes are inherited from it; if we set a font size of 0.9 em, it will be displayed at nine-tenths the default size. The em is a proportional font size—if we change the font size of the body, then the size of all the child elements (everything in our markup) will change size proportionately. Using proportional font sizes, in contrast to fixed sizes such as pixels, also allows users to change the overall type size of the site from their browser menu in the same way that restyling the size in the body tag does, which is very helpful for low-vision users. Here we have reduced the size of the type in the navigation links to 0.8 of the default font size.

Now we'll style the navigation links and the content link with two completely different looks. To do this, each rule we write leads off with the context in which the tag is located. Here, the # character refers to the ID.

```
<style type="text/css">

* {margin:0; padding:0;}

body {margin:20px;}

div#navigation {font-family: Verdana, Arial, Helvetica, sans-serif;}

div#navigation a:link {color: #CCC; background-color:#666;}

div#navigation a:hover {color: #FFF; background-color:#333;}

div#content {font-family:Georgia, "Times New Roman", Times, serif;}

div#content a:link {color: #FF3300;}

div#content a:hover {text-decoration: none;}

</style>
```

In this way, we can write one set of styles for the navigation div links, and another set of styles for the content div links. Here we use the :link (not rolled over) and :hover (rolled over) CSS pseudo-classes to give a visual response when the user rolls the mouse pointer over the links (**Figure B.9**).

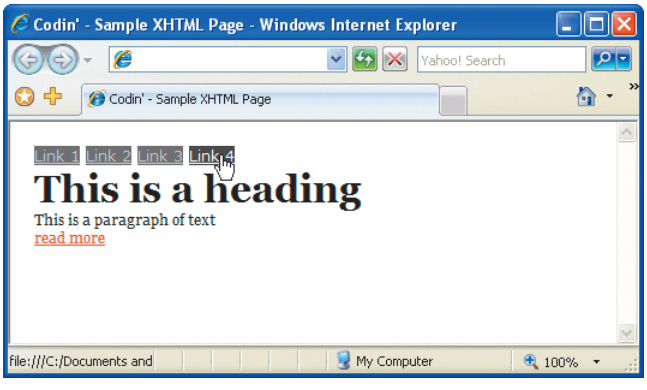


FIGURE B.9 Because the links have different div contexts, they can be styled to look and behave differently (the underlining would disappear from the red link when the user rolls the mouse over it).

Creating Navigation Elements

Navigation elements are a list of links from which the user can select. It's good practice to mark them as an XHTML list, like this:

```
<div id="navigation">
  <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    <li><a href="#">Link 3</a></li>
    <li><a href="#">Link 4</a></li>
  </ul>
</div>
```

We've seen the basic concepts of block and inline elements, so now we're going to remove the demonstration styles for these links. With the links correctly marked in an unordered list, we'll see how to use some real-world styles to create a nice navigation component from this markup, with every line commented so you can understand what it takes to style something that you might actually want on your Web site. If you have been following along, you can trash the styles you currently have and open a file with the following code:

CODE B.8: basic_xhtml_markup6a.html

```
* {margin:0; padding:0;}
body {margin:20px;}
div#navigation {
    width:150px;
    background-color:#CCCC99;
    border:1px solid #999966;
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size:.8em
    }
div#navigation ul {
    margin:12px 10px;
    border-top:1px dotted #999966;
    }
```

Set width of nav element

Set background color of div

Set border of nav element

Set font of nav element

Set font size of nav element

Space around menu items on background

Add a line over the first item in the menu

```
div#navigation li {
    list-style-type:none;
    border-bottom:1px dotted #999966;
}
div#navigation a:link {
    display:block;
    padding:3px 10px;
    color: #666600;
    text-decoration:none;
}
div#navigation a:hover {
    color: #000;
    text-decoration:underline;
    background-color:#CCCC66;
}

div#content {
    font-family:Georgia, "Times New Roman", Times, serif;
    font-size:.75em;
}
div#content a:link {
    color: #FF3300;
    text-decoration:none;}
div#content a:hover {
    text-decoration:underline;
}
```

Remove the bullets from the list

Add a line under each menu item

Change the link elements from inline (default) to block so they fill the 'li' elements

Create t/b & l/r space around the the link text—indents text from start of lines

Sets link color

Remove link underlining

Color of type when rolled over

Underlines type when rolled over

Changes background color when link is rolled over

Different font family for content

Set overall font size for content

Link color

No underlining on link

Adds underlining to link when rolled over

Figure B.10 shows the results.

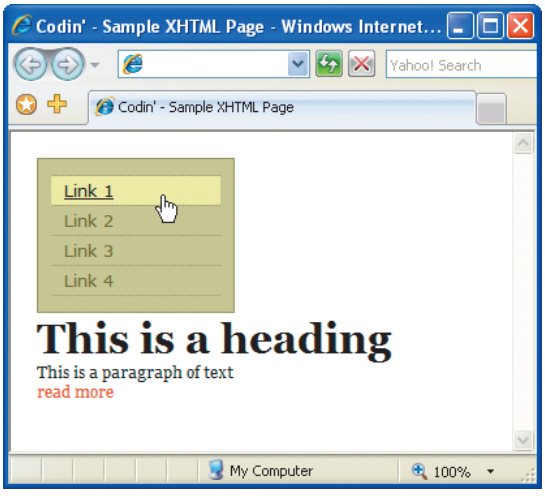


FIGURE B.10 The addition of background styles for the div and border styles for the ul and li elements unify these elements into a navigation component.

Creating a Multicolumn Page Layout

We've seen how the boxes of XHTML elements are organized onscreen by default and how to apply some basic styles to them—now let's style the elements as a group to create a more pleasing layout.

Almost always, you'll want more than one vertical column in your page design. The most common reason is so you can have navigation links down the left side of the screen, or somewhere else if you dare break this common convention, so let's create a simple two-column layout with CSS.

Setting the Width of Elements

We've already set the width of the `div` tag that contains the list of links to `150px` and switched the display property of the links themselves to `block`. Inline element boxes are only as big as the content inside them, but we want to make those links take on the `auto` width of block level elements, so that the whole area of the `li` element becomes filled with the link, and the entire area of the list item becomes "hot" (responds to rollovers), not just the text itself.

Setting the width of block elements is the first step in creating page layouts with more than one column. For this example, let's create a two-column layout that is 700 pixels wide, using a width of 150 pixels for the navigation area, which we have already set, and a width of 550 pixels for the content area, which we will set next (**Figure B.11**).

CODE B.9: basic_xhtml7.tif

```
div#content {
        width:550px;
        border: 1px solid;
        font-family:Georgia, "Times New Roman", Times, serif;
        font-size:.75em;
      }
```

Temporary style so we can see div width

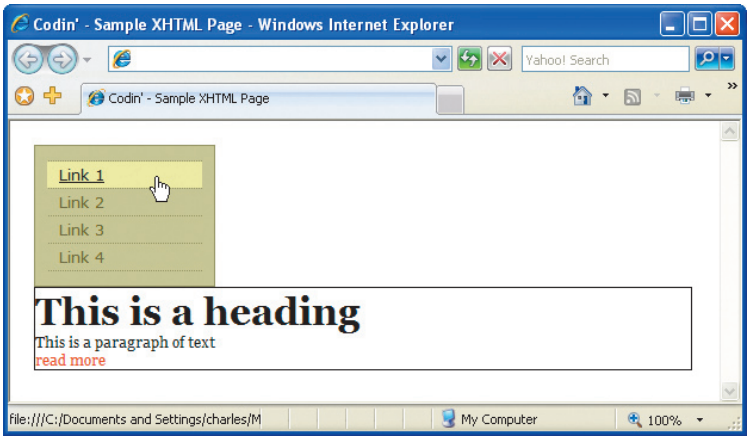


FIGURE B.11 The navigation and content area elements now have their widths defined.

Now we have the two elements set to the required widths. All we have to do is get them to sit next to each other instead of under one another. The easiest way to accomplish this is to float both elements. Float is a property that is commonly used to wrap text around images or other elements, which we can see by floating just the navigation `div` element (**Figure B.12**).

CODE B.10: basic_xhtml8.tif

```
div#navigation {
        width:150px;
        float:left;
        margin: 0 6px 6px 0;
```

Set width of nav element

Moves nav up and to left as far as possible within containing body element

Small margin on right and bottom of element stops content div text from touching it

Set background color of nav element

Set border of nav element

```
    background-color:#CCCC99;
    border:1px solid #999966;
}
```

FIGURE B.12 Floating the navigation element causes the text in the content element to wrap around it.

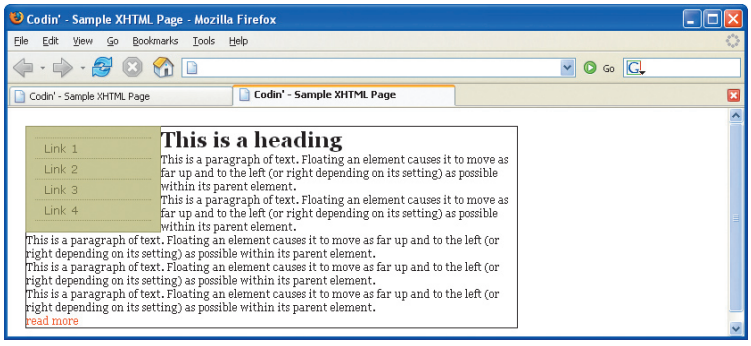


Figure B.12 shows more text added to the content `div` element so you can see that once the text gets below the floated navigation `div` element, the text wraps around it.

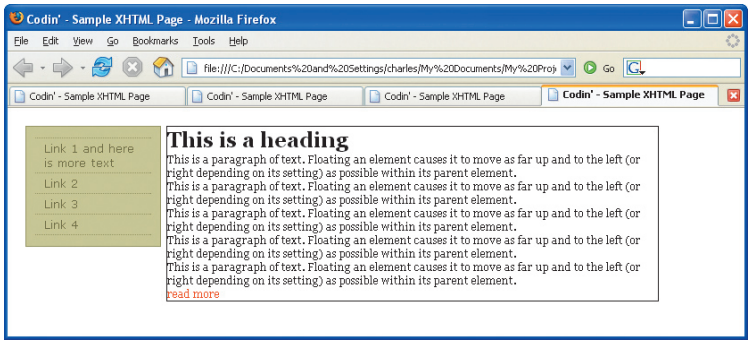
The simple way to make the content `div` element a column instead of having it wrap under the navigation `div` element is to float it, too (**Figure B.13**).

CODE B.11: basic_xhtml9.tif

```
div#content {
    width:550px;
    float:left;
    border: 1px solid;
    font-family:Georgia, "Times New Roman", Times, serif;
    font-size:.75em;
}
```

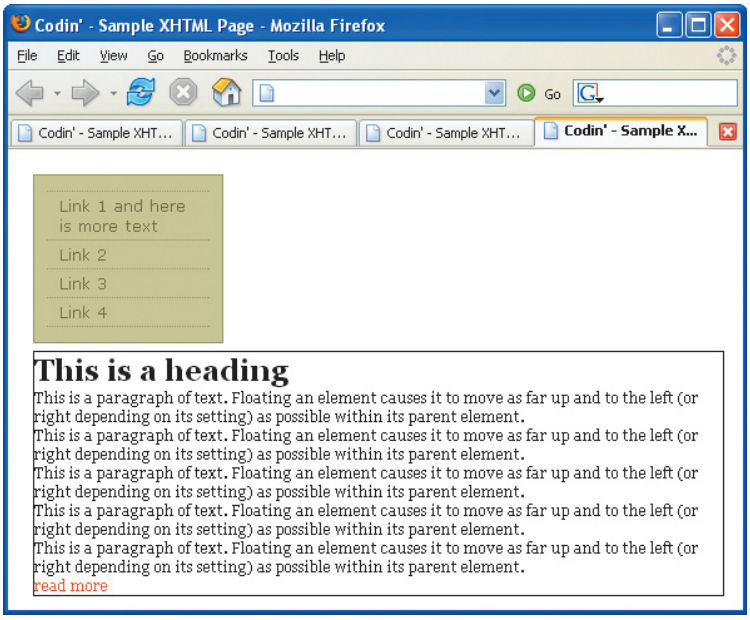
Floating this div as well as the nav div forms two columns

FIGURE B.13 Floating both the navigation and the content div elements is a simple way to create two columns.



The problem with a floated-columns layout is that if the browser window is very narrow, there is no longer room for the two columns to sit side by side, and the content column will move under the navigation column (**Figure B.14**).

FIGURE B.14 The floated elements will stack if there is not enough room for them to sit side by side.



Fortunately, the problem of stacked elements is easy to fix. We just wrap a new `div` element around both columns and fix its width. We add our wrapper `div` element to the markup right inside the `body` tags, like this:

CODE B.12: basic_xhtml10.tif

```
<body>
<div id = "wrapper">
<div id="navigation">
  <ul>
    <li><a href="#">Link 1 and…
```

Big chunk of markup removed here to save space

```
…element.</p>
  <a href="#">read more</a> </div>
  </div><!--end of wrapper div-->
</body>
```

Then we add styles for this new element.

```
div#wrapper {
        width:712px;
        float:left;
    }
```

Floated left to ensure it encloses the floated nav and content divs

Figure B.15 shows the results.

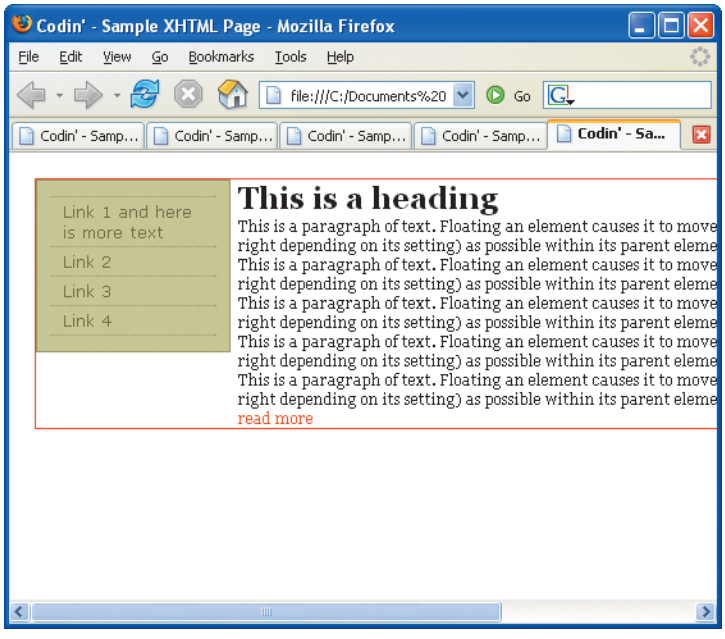


FIGURE B.15 The fixed width wrapper prevents one column from moving under the other, even when the browser window is narrower than the layout, as shown here.

I made the wrapper 712 pixels wide instead of 700 because there are 6 pixels of padding on each side of the navigation element, adding an extra 12 pixels of width to the layout. You have to allow for every pixel in the width when you set the wrapper width, because if the wrapper `div` element is too narrow, it will squeeze the columns under each other no matter how wide the browser is set. That said, the wrapper doesn't have to be a snug fit; you could set it to 750 pixels to allow for later alterations to the width of the contained elements, and it would still do its job of protecting the columns from getting squeezed under one another if the browser window is made narrow by the user.

Note that the wrapper is also floated; `div` elements don't normally enclose floated elements, but floating the wrapper itself makes it enclose the floated columns.

The "floated columns" technique is just one of several ways to create a multicolumn page layout. See my book Stylin' with CSS, *also from New Riders, for a number of other approaches. You can also learn more about page layouts with CSS at http://css-discuss.incutio.com/?page=CssLayouts.*

Adding a Header

Most sites have a banner across the top of the page to identify the site (provide orientation), so we will add one here. The graphic we will use was created in Adobe Fireworks and is 72 pixels high and 712 pixels wide.

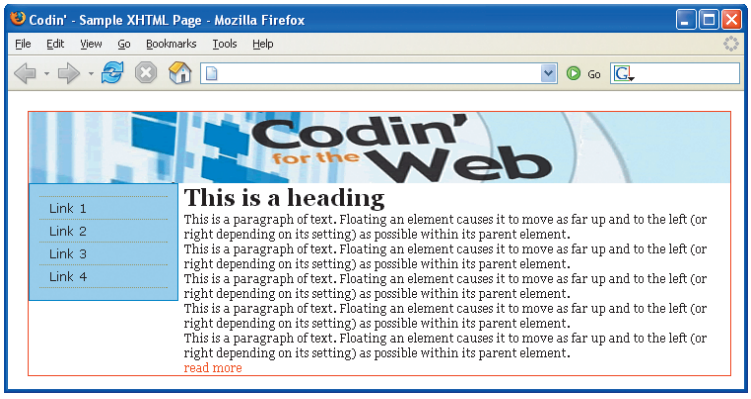
CODE B.13: basic_xhtml11.tif

```
<body>
<div id = "wrapper">
<div id="header"><!-- --></div>
<div id="navigation">
  <ul>
    <li><a href="#">…etc….

div#header {
      width:712px;
    height:72px;
    background-image:url(images/codin_header.gif);
    text-align:center;
    }
```

We created a header `div` tag in the markup but put only a comment in it. As you can see, the graphic is added via the CSS as a background element. What's neat about working this way, rather than adding the graphic to the markup with an `image` tag, is that we can add text (or images) in the markup of the header `div` tag later, and it will overlay the background graphic that we have just set in the CSS.

Figure B.16 shows our results.

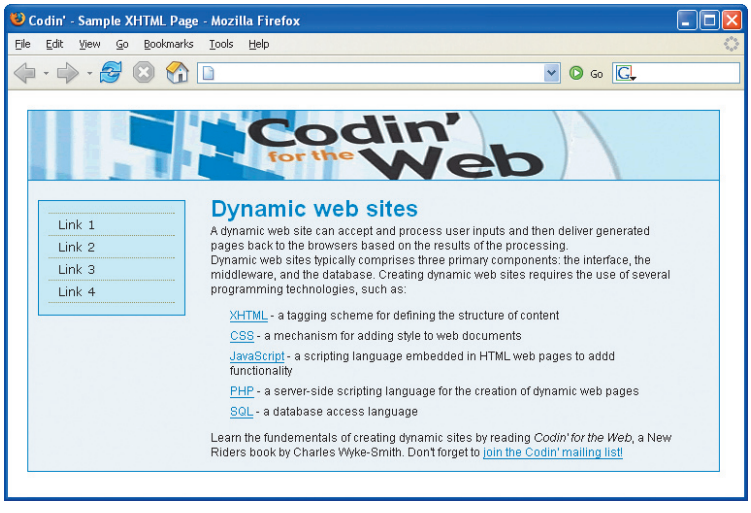
FIGURE B.16 The header added and some color changes in the navigation area to make it match the new colors.



Completing the Layout

That completes the structure of this simple two-column layout. Now it's time to add some content and style it into a more complete page (**Figure B.17**).

FIGURE B.17 Some real-world content and several small refinements to the CSS get us close to a usable template.



I've made a number of changes here, most notably adding some real content, including an unordered list in the content `div` element. and reworking the styles accordingly. I also set a single font family for the entire page in a single declaration in the CSS `body` rule; because this is the topmost, parent element, all the other elements inherit this font, so I can remove the other `font-family` styles that I had elsewhere.

I also made some changes to the margins for some of the elements to get everything better aligned.

You can see all these changes in this final version of the code.

CODE B.14: basic_xhtml12.tif

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />

<title>Codin' - Sample XHTML Page</title>

<style type="text/css">

* {margin:0; padding:0;}

body {margin:20px; font-family:Geneva, Arial, Helvetica,
sans-serif;}

div#header {

    width:712px;

    height:72px;

    background-image:url(images/codin_header.gif);

    border-bottom:#067EC5 2px;

    }

div#wrapper {

    width:712px;

    float:left;

    border: 1px solid #067EC5;

    background-color:#F1F8FF;

    }

div#navigation {

    width:150px;

    float:left;
```

Floated left to ensure it encloses the floated nav and content divs

Set width of nav element

Moves nav up and to left as far as possible within containing body element

Small margin on right and bottom of element stops content div text from touching it

Set background color of nav element

Set border of nav element

Set font of nav element

Set font size of nav element

```
        margin: 20px 6px 6px 10px;
        background-color:#CEF0FC;
        border:1px solid #067EC5;
        font-family: Verdana, Arial, Helvetica, sans-serif;
        font-size:.8em
        }
    div#navigation ul {
```

Create t/b & l/r space around menu items on background

Add a line over the first item in the menu

```
        margin:12px 10px;
        border-top:1px dotted #999966;
        }
    div#navigation li {
```

Remove the bullets from the list

Add a line under each menu item

```
        list-style-type:none;
        border-bottom:1px dotted #999966;
        }
    div#navigation a {
```

Change the link elements from inline (default) to block so they fill the li elements

Create t/b & l/r space around the the link text—indents text from start of lines

Sets link color

Removes underlining

```
        display:block;
        padding:3px 10px;
        color: #000;
        text-decoration:none;
        }
    div#navigation a:hover {
```

Color of type when rolled over

Underlines type when rolled over

Changes background color when link is rolled over

```
        color: #000;
        text-decoration:underline;
        background-color:#CEF0FC;
        }
    div#content {
        width:490px;
        padding:1em 20px;
```

Floating this div as well as the nav div forms two columns

```
        float:left;
        font-size:.75em;
```

```
    }
div#content h1 {
    margin-top:2px;
    color:#067EC5;
    }
div#content ul {
    margin:1em 20px;
    }
div#content li {
    list-style-type:none;
    margin:0 0 .5em 0}
div#content a {
    color: #1A78BE;
    }
div#content a:hover {
    text-decoration:none;
    }
</style>
</head>
<body>
<div id = "wrapper">
  <div id="header">
    <!-- -->
  </div>
  <div id="navigation">
    <ul>
      <li><a href="#">Link 1</a></li>
      <li><a href="#">Link 2</a></li>
      <li><a href="#">Link 3</a></li>
      <li><a href="#">Link 4</a></li>
```

```
    </ul>
  </div>
  <div id="content">
    <h1>Dynamic web sites</h1>
    <p>A dynamic web site can accept and process user inputs
and then deliver generated pages back to the browsers based
on the results of the processing.</p>
  <p>Dynamic web sites typically comprises three primary
components: the interface, the middleware, and the database.
Creating dynamic web sites requires the use of several
programming technologies, such as:</p>
  <ul>
  <li><a href="#">XHTML</a> - a tagging scheme for defining
the structure of content</li>
  <li><a href="#">CSS</a> - a mechanism for adding style to
web documents</li>
  <li><a href="#">JavaScript</a> - a scripting language
embedded in HTML web pages to addd functionality</li>
  <li><a href="#">PHP</a> - a server-side scripting language
for the creation of dynamic  web pages</li>
  <li><a href="#">SQL</a> - a database access language</li>
  </ul>
  <p>Learn the fundementals of creating dynamic sites by
reading <em>Codin' for the Web</em>, a New Riders book by
Charles Wyke-Smith. Don't forget to <a href="#">join the
Codin' mailing list!</a>
</div>
<!--end of wrapper div-->
</body>
</html>
```

Moving the Styles to a Style Sheet

The embedded styles within the `style` tag affect only this page: the page in which they are included. The next step is to move these styles into a style sheet that can be shared by many pages. This is where CSS really comes into its own, because it allows you to change the look and feel of an entire site by changing a single file.

1. Create a new folder called **css** in the same folder as the XHTML document that you have been working on.
2. Create a new text document, copy and paste all the styles between the `<style>` and `</style>` tags into this new document, and save it into the css folder with the name **codin_styles.css**.
3. Delete the `style` tags and all the styles between them from the XHTML document.
4. Link the document to the style sheet, by inserting the following into the head of the XHTML document (for example, right after the `title` tag):

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

   When you do this, any graphics referenced from the styles (such as the header graphic in this example) will now have a different relative path. When the styles were in the head of the page, this was the background image URL:

   ```
   div#header {
         width:712px;
   height:72px;
   background-image:url(images/codin_header.gif);
   border-bottom:#067EC5 2px;
   }
   ```

   Now that we have moved the CSS into a separate .css document in a different folder (the new css folder we just created), the relative path to the images folder needs to be modified.

5. Change the path to the images folder to this:

   ```
   background-image:url(../images/codin_header.gif);
   ```

   The `../` means "up one folder."

6. Preview the page.

   It should look just as it did before. (If it doesn't, it's probably because you don't have the file name correct, the file is not in the right place, or the path is wrong.)

Now all you have to do is add this `link` tag to the head of any XHTML page (you may need to tweak the URL if it is in a different folder), and it too will take on the formatting defined in the style sheet.

Summary

This completes this brief overview of XHTML and CSS. Now you can move on to PHP, a language that, in contrast to XHTML and CSS, is capable of accepting and processing data and generating XHTML pages that are then displayed in the user's browser. TChapter 1 of *Codin' for the Web* introduces basic coding concepts that apply to PHP—and to almost any programming language you may encounter.