LESSON 3:
Introducing HTML and XHTML

After finishing up the discussions about the World Wide Web and getting organized, with a large amount of text to read and concepts to digest, you're probably wondering when you're actually going to get to write a web page. That is, after all, why you bought the book. Wait no longer!

In this Lesson

Today, you get to create your very first (albeit brief) web page, learn about HTML (the language for writing web pages), and learn about the following:

- What HTML is and why you have to use it
- What you can and cannot do when you design HTML pages
- What HTML tags are and how to use them
- How to write pages that conform to the XHTML standard
- How you can use Cascading Style Sheets to control the look and feel of your pages

What HTML Is—And What It Isn't

Take note of just one more thing before you dive into actually writing web pages. You should know what HTML is, what it can do, and most importantly what it can't do.

HTML stands for *Hypertext Markup Language*. HTML is based on the *Standard Generalized Markup Language (SGML)*, a much larger document-processing system. To write HTML pages, you won't need to know a whole lot about SGML. However, knowing that one of the main features of SGML is that it describes the general structure of the content inside documents—rather than its actual appearance on the page or onscreen—does help. This concept might be a bit foreign to you if you're used to working with WYSIWYG (What You See Is What You Get) editors like Adobe's Dreamweaver or Microsoft FrontPage, so let's go over the information carefully.

HTML Describes the Structure of a Page

HTML, by virtue of its SGML heritage, is a language for describing the structure of a document, not its actual presentation. The idea here is that most documents have common elements—for example, titles, paragraphs, and lists. Before you start writing, therefore, you can identify and define the set of elements in that document and give them appropriate names (see Figure 3.1).

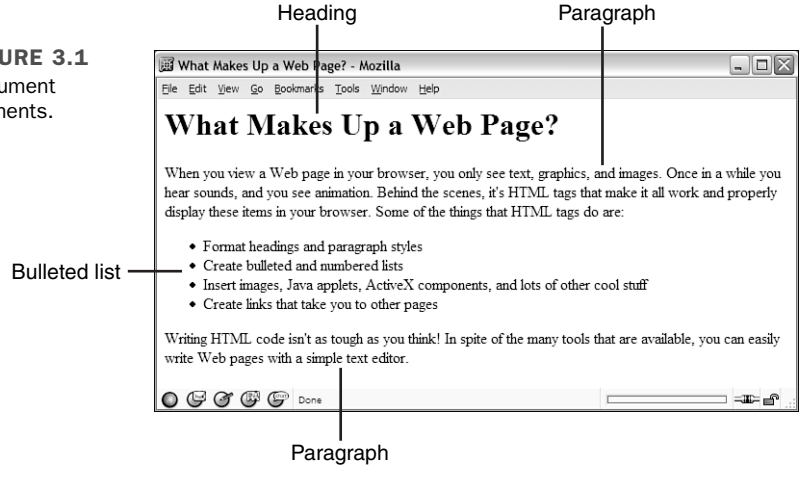


FIGURE 3.1
Document elements.

If you've worked with word processing programs that use style sheets (such as Microsoft Word) or paragraph catalogs (such as FrameMaker), you've done something similar; each section of text conforms to one of a set of styles that are predefined before you start working.

HTML defines a set of common styles for web pages: headings, paragraphs, lists, and tables. It also defines character styles such as boldface and code examples. These styles are indicated inside HTML documents using *tags*. Each tag has a specific name and is set off from the content of the document using a notation that I'll get into a bit later.

HTML Does Not Describe Page Layout

When you're working with a word processor or page layout program, styles are not just named elements of a page—they also include formatting information such as the font size and style, indentation, underlining, and so on. So, when you write some text that's supposed to be a heading, you can apply the Heading style to it, and the program automatically formats that paragraph for you in the correct style.

HTML doesn't go this far. For the most part, HTML doesn't say anything about how a page looks when it's viewed. HTML tags just indicate that an element is a heading or a list; they say nothing about how that heading or list is to be formatted. So, as with the magazine example and the layout person who formats your article, the layout person's job is to decide how big the heading should be and what font it should be in. The only thing you have to worry about is marking which section is supposed to be a heading.

NOTE

Although HTML doesn't say much about how a page looks when it's viewed, Cascading Style Sheets (CSS) enable you to apply advanced formatting to HTML tags. Many changes in HTML 4.0 favor the use of CSS tags. And XHTML, which is the current version of HTML, eliminates almost all tags that are associated with formatting in favor of Cascading Style Sheets. I'll talk about both XHTML and CSS later today.

Web browsers, in addition to providing the networking functions to retrieve pages from the Web, double as HTML formatters. When you read an HTML page into a browser such as Netscape or Internet Explorer, the browser interprets, or *parses*, the HTML tags and formats the text and images on the screen. The browser has mappings between the names of page elements and actual styles on the screen; for example, headings might be in a larger font than the text on the rest of the page. The browser also wraps all the text so that it fits into the current width of the window.

Different browsers running on diverse platforms might have various style mappings for each page element. Some browsers might use different font styles than others. For example, a browser on a desktop computer might display italics as italics, whereas a handheld device or mobile phone might use reverse text or underlining on systems that don't have italic fonts. Or it might put a heading in all capital letters instead of a larger font.

What this means to you as a web page designer is that the pages you create with HTML might look radically different from system to system and from browser to browser. The actual information and links inside those pages are still there, but the onscreen appearance changes. You can design a web page so that it looks perfect on your computer system, but when someone else reads it on a different system, it might look entirely different (and it might very well be entirely unreadable).

How the Visual Styles for Tags Evolved

In practice, most HTML tags are rendered in a fairly standard manner, on desktop computers at least. When the earliest browsers were written, somebody decided that links would be underlined and blue, visited links would be purple, and emphasized text would appear in italics. They also made similar decisions about every other tag. Since then, pretty much every browser maker has followed that convention to a greater or lesser degree. These conventions blurred the line separating structure from presentation, but in truth it still exists, even if it's not obvious.

Why It Works This Way

If you're used to writing and designing documents that will wind up printed on paper, this concept might seem almost perverse. No control over the layout of a page? The whole design can vary depending on where the page is viewed? This is awful! Why on earth would a system work like this?

Remember in Lesson 1, "Navigating the World Wide Web," when I mentioned that one of the cool things about the Web is that it's cross-platform and that web pages can be viewed on any computer system, on any size screen, with any graphics display? If the final goal of web publishing is for your pages to be readable by anyone in the world, you can't count on your readers having the same computer systems, the same size screens, the same number of colors, or the same fonts that you have. The Web takes into account all these differences and enables all browsers and all computer systems to be on equal ground.

The Web, as a design medium, is not a new form of paper. The Web is an entirely different medium, with its own constraints and goals that are very different from working with paper. The most important rules of web page design, as I'll keep harping on throughout this book, are the following:

DO	DON'T
DO design your pages so that they work in most browsers. **DO** focus on clear, well-structured content that's easy to read and understand.	**DON'T** design your pages based on what they look like on your computer system and on your browser.

Throughout this book, I'll show you examples of HTML code and what they look like when displayed. In examples in which browsers display code very differently, I'll give you a comparison of how a snippet of code looks in two very different browsers. Through these examples, you'll get an idea for how different the same page can look from browser to browser.

TIP

Although this rule of designing by structure and not by appearance is the way to produce good HTML, when you surf the Web, you might be surprised that the vast majority of websites seem to have been designed with appearance in mind—usually appearance in a particular browser such as Microsoft Internet Explorer. Don't be swayed by these designs. If you stick to the rules I suggest, in the end, your web pages and websites will be even more successful simply because more people can easily read and use them.

How Markup Works

HTML is a *markup language*. Writing in a markup language means that you start with the text of your page and add special tags around words and paragraphs. The tags indicate the different parts of the page and produce different effects in the browser. You'll learn more about tags and how they're used in the next section.

HTML has a defined set of tags you can use. You can't make up your own tags to create new styles or features. And just to make sure that things are really confusing, various browsers support different sets of tags. To further explain this, take a brief look at the history of HTML.

A Brief History of HTML Tags

The base set of HTML tags, the lowest common denominator, is referred to as HTML 2.0. HTML 2.0 is the old standard for HTML (a written specification for it is developed and maintained by the W3C) and the set of tags that all browsers must support. In the next few lessons, you'll primarily learn to use tags that were first introduced in HTML 2.0.

The HTML 3.2 specification was developed in early 1996. Several software vendors, including IBM, Microsoft, Netscape Communications Corporation, Novell, SoftQuad, Spyglass, and Sun Microsystems, joined with the W3C to develop this specification. Some of the primary additions to HTML 3.2 included features such as tables, applets, and text flow around images. HTML 3.2 also provided full backward-compatibility with the existing HTML 2.0 standard.

NOTE

The enhancements introduced in HTML 3.2 are covered later in this book. You'll learn more about tables in Lesson 8, "Building Tables." Lesson 11, "Integrating Multimedia: Sound, Video, and More," tells you how to use Java applets.

HTML 4.0, first introduced in 1997, incorporated many new features that gave designers greater control over page layout than HTML 2.0 and 3.2. Like HTML 2.0 and 3.2, the W3C maintains the HTML 4.0 standard.

Framesets (originally introduced in Netscape 2.0) and floating frames (originally introduced in Internet Explorer 3.0) became an official part of the HTML 4.0 specification. Framesets are discussed in more detail in Lesson 14, "Working with Frames and Linked Windows." We also see additional improvements to table formatting and rendering. By far, however, the most important change in HTML 4.0 was its increased integration with style sheets.

NOTE

If you're interested in how HTML development is working and just exactly what's going on at the W3C, check out the pages for HTML at the Consortium's site at http://www.w3.org/pub/WWW/MarkUp/.

At one time, Microsoft and Netscape were releasing new versions of their browsers frequently, competing to see who could add the most compelling new features to HTML without waiting for the standards process to catch up. These days, browser releases are less frequent, and HTML is more "finished" than it was in the late nineties. Now developers must mostly concern themselves with slight differences between how the browsers handle the HTML they support rather than deciding against competing sets of features. Confused yet? You're not alone. The extra work involved in dealing with variations between browsers has been a headache for Web developers for a very long time. Keeping track of all this information can be really confusing. Throughout this book, as I introduce each tag, I'll explain any browser specific issues you'll run into.

The Current Standard: XHTML 1.0

The Internet is no longer limited to computer hardware and software. MSN TV enables you to access the Internet, giving you more reason to become a couch potato. Personal information managers and palmtop computers enable you to access the Internet while you're on the road. More and more people are accessing the Internet with mobile phones and other wireless devices. Special interfaces and hardware enable physically challenged individuals to access the Internet. As it has matured, the Internet has become an effective means of communication and education for the masses. Many of the newer portable technologies, however, pose problems for the old HTML specification. They simply don't have the processing power of a desktop computer, and aren't as forgiving of poorly written HTML as web browsers. The developers of the HTML specification have struggled to accommodate these ongoing changes, and the limitations of HTML have become evident. We're stretching and distorting the HTML specification far beyond its capabilities. The future of the Internet demands a markup language that's more extensible and portable than HTML. The direction is heading toward the use of XML (short for *Extensible Markup Language*), a subset of SGML that allows for custom tags to be processed. And that's where XHTML 1.0 comes into play.

XHTML 1.0 is written in XML, and is the current standard that will help web designers prepare for the future. Documents written in XHTML can be viewed on current browsers, but at the same time they're valid XML documents. The purpose of this book is not only to teach you HTML 4.01, but also to teach you how to format your HTML so that it's compliant with the XHTML 1.0 specification.

Technically, XHTML 1.0 and HTML 4.01 are *very* similar. The tags and attributes are virtually the same, but a few simple rules have to be followed in order to make sure that a document is compliant with the XHTML 1.0 specification. Throughout this book, I'll explain how to deal with the different HTML tags to make sure that your pages are readable and still look good in all kinds of browsers.

What HTML Files Look Like

Pages written in HTML are plain text files (ASCII), which means that they contain no platform- or program-specific information. Any editor that supports text (which should be just about any editor—more about this subject in "Programs to Help You Write HTML" later today) can read them. HTML files contain the following:

- The text of the page itself
- HTML tags that indicate page elements, structure, formatting, and hypertext links to other pages or to included media

Most HTML tags look something like the following:

```
<thetagname>affected text</thetagname>
```

The tag name itself (here, *thetagname*) is enclosed in brackets (< >). HTML tags generally have a beginning and an ending tag surrounding the text they affect. The beginning tag "turns on" a feature (such as headings, bold, and so on), and the ending tag turns it off. Closing tags have the tag name preceded by a slash (/). The opening tag (for example, for paragraphs) and closing tag (for example, for paragraphs) compose what is officially called an *HTML element*.

CAUTION

Be aware of the difference between the forward slash (/) mentioned with relation to tags, and backslashes (\), which are used by DOS and Windows in directory references on hard drives (as in C:\window or other directory paths). If you accidentally use the backslash in place of a forward slash in HTML, the browser won't recognize the ending tags.

Not all HTML tags have both an opening and closing tag. Some tags are only one-sided, and still other tags are containers that hold extra information and text inside the brackets. XHTML 1.0, however, requires that *all* tags be closed. You'll learn the proper way to open and close the tags as the book progresses.

Another difference between HTML 4.0 and XHTML 1.0 relates to usage of lowercase tags and attributes. HTML tags are not case sensitive; that is, you can specify them in uppercase, lowercase, or in any mixture. So, <HTML> is the same as <html>, which is the same as <HtMl>. This isn't the case for XHTML 1.0, where all tag and attribute names must be written in lowercase. To get you thinking in this mindset, the examples in this book display tag and attribute names in bold lowercase text.

▼ Task: Exercise 3.1: Creating Your First HTML Page

Now that you've seen what HTML looks like, it's your turn to create your own web page. Start with a simple example so that you can get a basic feel for HTML.

To get started writing HTML, you don't need a web server, a web provider, or even a connection to the Web itself. All you really need is an application in which you can create your HTML files and at least one browser to view them. You can write, link, and test whole suites of web pages without even touching a network. In fact, that's what you're going to do for the majority of this book. I'll talk later about publishing everything on the Web so that other people can see your work.

▼

To get started, you'll need a text editor. A *text editor* is a program that saves files in ASCII format. ASCII format is just plain text, with no font formatting or special characters. For Windows, Notepad and Microsoft WordPad are good basic text editors (and free with your system). Shareware text editors are also available for various operating systems, including DOS, Windows, Mac OS, and Linux. If you point your web browser to www.download.com and enter **`Text Editors`** as a search term, you'll find many resources available to download. If you're a Windows user, you might want to check out HTML-Kit in particular. It's a free text editor specifically built for editing HTML files. You can download it at http://www.chami.com/html-kit/. By the same token, Mac users might want to look at TextWrangler, available from http://www.barebones.com. If you prefer to work in a word processor such as Microsoft Word, don't panic. You can still write pages in word processors just as you would in text editors, although doing so is more complicated. When you use the Save or Save As command, you'll see a menu of formats you can use to save the file. One of them should be Text Only, Text Only with Line Breaks, or DOS Text. All these options will save your file as plain ASCII text, just as if you were using a text editor. For HTML files, if you have a choice between DOS Text and just Text, use DOS Text, and use the Line Breaks option if you have it.

CAUTION

If you do use a word processor for your HTML development, be very careful. Many recent word processors are including HTML modes or mechanisms for creating HTML or XML code. This feature can produce unusual results or files that simply don't behave as you expect. If you run into trouble with a word processor, try using a text editor and see whether it helps.

What about the plethora of free and commercial HTML editors that claim to help you write HTML more easily? Some are text editors that simplify common tasks associated with HTML coding. If you've got one of these editors, go ahead and use it. If you've got a fancier editor that claims to hide all the HTML for you, put it aside for the next couple of days and try using a plain text editor just for a little while. Appendix A, "Sources for Further Information," lists many URLs where you can download free and commercial HTML editors that are available for different platforms. They appear in the section titled "HTML Editors and Converters" (in Appendix A).

Open your text editor and type the following code. You don't have to understand what any of it means at this point. You'll learn more about much of this today and tomorrow. This simple example is just to get you started.

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

```
<html>
<head>
<title>My Sample HTML Page</title>
</head>
<body>
<h1>This is an HTML Page</h1>
</body>
</html>
```

NOTE

Note that the <!DOCTYPE> tag in the previous example doesn't appear in lowercase like the rest of the tags. This tag is an exception to the XHTML rule and should appear in uppercase. This is explained in detail in Lesson 17, "Designing for the Real World." In fact, you don't have to specify a DOCTYPE at all to get your pages to work. The purpose of the DOCTYPE is to tell validators and browsers which specification your page was written to. I'll include them in all examples in the book, but you can leave them out if you like.

After you create your HTML file, save it to your hard disk. Remember that if you're using a word processor like Microsoft Word, choose Save As and make sure that you're saving it as "Text Only". When you choose a name for the file, follow these two rules:

- The filename should have an extension of `.html` (`.htm` on DOS or Windows systems that support only three-character extensions)—for example, `myfile.html`, `text.html`, or `index.htm`. Most web software requires your files to have these extensions, so get into the habit of doing it now. (If you are using Windows, make sure that your computer is configured to show file extensions. If it isn't, you'll find yourself creating files named things like `myfile.html.txt`, which your browser will not think are HTML files.)
- Use small, simple names. Don't include spaces or special characters (bullets, accented characters)—just letters and numbers are fine.

Task: Exercise 3.2: Viewing the Result

Now that you have an HTML file, start your web browser. You don't have to be connected to the Internet because you're not going to be opening pages at any other site. Your browser or network connection software might complain about the lack of a network connection, but you should be able to work offline.

After your browser is running, look for a menu item or button labeled Open, Open File, or maybe Open Page. Choosing it enables you to browse your local disk. The Open command (or its equivalent) opens a document from your local disk, parses it, and displays it.

By using your browser and the Open command, you can write and test your HTML files on your computer in the privacy of your own home. (On most operating systems, you can just drag the icon from your HTML file into an open browser window if you prefer.)

If you don't see something similar to what's shown in Figure 3.2 (for example, if parts are missing or if everything looks like a heading), go back into your text editor and compare your file to the example. Make sure that all your tags have closing tags and that all your < characters are matched by > characters. You don't have to quit your browser to do so; just fix the file and save it again under the same name.

FIGURE 3.2
The sample HTML file.

Next, go back to your browser. Locate and choose a menu item or button called Reload (for Netscape users) or Refresh (for Internet Explorer users). The browser will read the new version of your file, and voilà! You can edit and preview and edit and preview until you get the file right.

If you're getting the actual HTML text repeated in your browser rather than what's shown in Figure 3.2, make sure that your HTML file has an `.html` or `.htm` extension. This file extension tells your browser that it's an HTML file. The extension is important.

If things are going really wrong—if you're getting a blank screen or you're getting some really strange characters—something is wrong with your original file. If you've been using a word processor to edit your files, try opening your saved HTML file in a plain text editor (again, Notepad will work just fine). If the text editor can't read the file or if the result is garbled, you haven't saved the original file in the right format. Go back into your original editor, and try saving the file as text only again. Then try viewing the file again in your browser until you get it right.

Text Formatting and HTML

When an HTML page is parsed by a browser, any formatting you might have done by hand—that is, any extra spaces, tabs, returns, and so on—is ignored. The only thing that specifies formatting in an HTML page is an HTML tag. If you spend hours carefully

editing a plain text file to have nicely formatted paragraphs and columns of numbers but don't include any tags, when a web browser loads the page, all the text will flow into one paragraph. All your work will have been in vain.

NOTE

There are two exceptions to this rule, a tag called `<pre>` and a CSS property. You'll learn about both of them in Lesson 6, "Formatting Text with HTML and CSS."

The advantage of having all white space (spaces, tabs, returns) ignored is that you can put your tags wherever you want. The following examples all produce the same output. Try them!

```
<h1>If music be the food of love, play on.</h1>
```

```
<h1>
If music be the food of love, play on.
</h1>
```

```
<h1>
If music be the food of love, play on.       </h1>
```

```
<h1>   If  music   be   the   food  of  love,
play   on. </h1 >
```

Using Cascading Style Sheets

Earlier, I mentioned Cascading Style Sheets as a way you could control the look and feel of your pages. Styles are a way to control how the browser renders HTML tags (or elements, as they're called in standards documents). For example, in today's lesson, I've used the `<h1>` tag a number of times. Most browsers print text enclosed inside an `<h1>` tag in a large, boldface font and leave some white space after the heading before printing something else. Using Cascading Style Sheets, you can tell the browser to render the `<h1>` tag differently than it normally would. CSS provides a lot of flexibility in how you can alter the appearance of any type of element, and the styles can be applied in a number of different ways.

The advantage of CSS is that it can be used at varying levels of specificity. For example, you can put all your styles into a separate file, and link to that file from your web page. That way, if you want to change the appearance of your site, you can simply edit your CSS file and make changes that span every page that links to your style sheet. Or, if you prefer, you can include styles at the top of your page so that they apply only to that page. You can also include styles inside the tags themselves using the `style` attribute (which I'll discuss in Lesson 9, "Creating Layouts with CSS").

You can also control the specificity of the styles you create based on how you define them. For example, you can write rules that apply to all tags of a specific type, such as all `<h1>` elements. Or you can specify classes for your elements and then write rules that apply only to members of that class. For example, you could create a class called `headline` and then make all `<h1>` elements in the `headline` class red. You can also write rules that apply to specific elements by assigning them a particular identifier and writing rules that apply to that identifier.

One thing you'll find as you progress through the book is that CSS can serve as a replacement for many common tags. As I describe various tags, I'll explain how the same effects can be achieved using CSS instead. Generally, the flexibility of CSS means you should use HTML to describe the structure of pages and CSS to define their appearance. The coverage of CSS in this book culminates with Lesson 9, which explains how to use CSS to manage the entire layout of the page, or even the entire layout of a site.

Including Styles in Tags

You've already seen how HTML pages are created using tags. I want to stop briefly and discuss attributes as well. An attribute is an additional bit of information that somehow affects the behavior of a tag. Attributes are included inside the opening tag in a pair. Here's an example:

```
<tag attribute="value">
```

Some attributes can be used with nearly any tag; others are highly specific. One attribute that can be used with nearly any tag is `style`. By including the `style` attribute in a tag, you can include one or more style rules within a tag itself. Here's an example using the `<h1>` tag, which I introduced earlier:

```
<h1 style="font-family: Verdana, sans-serif;">Heading</h1>
```

The `style` attribute of the `<h1>` tag contains a style declaration. All style declarations follow this same basic pattern, with the property on the left and the value associated with that property on the right. The rule ends with a semicolon, and you can include more than one in a `style` attribute by placing commas between them. If you're only including one rule in the `style` attribute, the semicolon is optional, but it's a good idea to include it. In the preceding example, the property is `font-family`, and the value is `Verdana, sans-serif`. This attribute modifies the standard `<h1>` tag by changing the font to Verdana, and if the user doesn't have that font installed on his system, whichever sans-serif font the browser selects. (Sans-serif fonts are those that do not include *serifs*, the small lines at the ends of characters.)

There are many, many properties that can be used in style declarations. As I've already said, putting a declaration into a `style` attribute is just one of several ways that you can apply styles to your document.

Programs to Help You Write HTML

You might be thinking that all this tag stuff is a real pain, especially if you didn't get that small example right the first time. (Don't fret about it; I didn't get that example right the first time, and I created it.) You have to remember all the tags, and you have to type them in right and close each one. What a hassle!

Many freeware and shareware programs are available for editing HTML files. Most of these programs are essentially text editors with extra menu items or buttons that insert the appropriate HTML tags into your text. HTML-based text editors are particularly nice for two reasons: You don't have to remember all the tags, and you don't have to take the time to type them all. I've already mentioned HTML-Kit, but there are plenty of others as well. Many general-purpose text editors also include special features to make it easier to deal with HTML files these days.

Many editors on the market purport to be WYSIWYG. As you learned earlier today, there's really no such thing as WYSIWYG when you're dealing with HTML. "What You Get" can vary wildly based on the browser.

With that said, as long as you're aware that the result of working in those editors can vary, using WYSIWYG editors can be a quick way to create simple HTML pages. For professional web development and for using many of the very advanced features, however, WYSIWYG editors can fall short, and you'll need to go "under the hood" to play with the HTML code anyhow. Even if you intend to use a WYSIWYG editor for the bulk of your HTML work, bear with me for the next couple of days and try these examples in text editors so that you get a feel for what HTML really is before you decide to move on to an editor that hides the tags.

CAUTION

WYSIWYG editors tend to work best with files they've created themselves. If you have some existing HTML files that you need to edit, opening them in a WYSIWYG editor can do more harm than good, particularly if the files were created in a different WYSIWYG editor.

In addition to HTML and WYSIWYG editors, you also can use converters, which take files from many popular word processing programs and convert them to HTML. With a simple set of templates, you can write your pages entirely in your favorite word processing program and then convert the result when you're done.

In many cases, converters can be extremely useful, particularly for putting existing documents on the Web as quickly as possible. However, converters suffer from many of the same problems as WYSIWYG editors. The results can vary from browser to browser, and many newer or advanced features aren't available in the converters. Also, most converter programs are fairly limited, not necessarily by their own features, but mostly by the limitations in HTML itself. No amount of fancy converting will make HTML do things that it can't do already. If a particular capability doesn't exist in HTML, the converter can't do anything to solve that problem. In fact, the converter might end up doing strange things to your HTML files, causing you more work than if you just did all the formatting yourself.

As previously mentioned, Appendix A lists many of the web page editors that are currently available. For now, if you have a simple HTML editor, feel free to use it for the examples in this book. If all you have is a text editor, no problem; you'll just have to do a little more typing.

Summary

Today, you learned some basic points about what HTML is and how you define a text document as a web page. You learned a bit about the history of HTML and the reasons why the HTML specification has changed several times since the beginning. You also learned how Cascading Style Sheets can be used to augment your HTML. You created your first web page with some basic tags. It wasn't so bad, was it? You also learned a bit about the current standard version of HTML—XHTML, and how to apply styles using Cascading Style Sheets. In tomorrow's lesson, you'll expand on this and will learn more about adding headings, text, and lists to your pages.

Workshop

Now that you've had an introduction to HTML and a taste of creating your first very simple web page, here's a workshop that will guide you toward more of what you'll learn. A couple of questions and answers that relate to HTML formatting are followed by a brief quiz and answers about HTML. The exercises prompt you to examine the code of a more advanced page in your browser.

Q&A

Q **Can I do *any* formatting of text in HTML?**

A You can do some formatting to strings of characters; for example, you can make a word or two bold. Pretty much all browsers support tags for formatting text (most were added in HTML 3.2), but most of these tags have given way to CSS formatting in HTML 4.01 and XHTML 1.0. You'll learn some formatting tricks in Lesson 6.

Q **I'm using Windows. My word processor won't let me save a text file with an extension that's anything except `.txt`. If I type in `index.html`, my word processor saves the file as `index.html.txt`. What can I do?**

A You can rename your files after you've saved them so that they have an `html` or `htm` extension, but having to do so can be annoying if you have a large number of files. Consider using a text editor or HTML editor for your web pages.

Quiz

1. What does HTML stand for? How about XHTML?
2. What's the primary function of HTML?
3. Why doesn't HTML control the layout of a page?
4. Which version of HTML provides the lowest common denominator of HTML tags?
5. What's the basic structure of an HTML tag?

Quiz Answers

1. HTML stands for Hypertext Markup Language. XHTML stands for Extensible HyperText Markup Language.
2. HTML defines a set of common styles for web pages (headings, paragraphs, lists, tables, character styles, and more).
3. HTML doesn't control the layout of a page because it's designed to be cross-platform. It takes the differences of many platforms into account and allows all browsers and all computer systems to be on equal ground.
4. The lowest common denominator for HTML tags is HTML 2.0, the oldest standard for HTML. This is the set of tags that *all* browsers *must* support. HTML 2.0 tags can be used anywhere.
5. Most HTML elements consist of opening and closing tags, and they surround the text that they affect. The tags are enclosed in brackets (<>). The beginning tag turns on a feature, and the ending tag, which is preceded by a forward slash (/), turns it off.

Exercises

1. Before you actually start writing a meatier HTML page, getting a feel for what an HTML page looks like certainly helps. Luckily, you can find plenty of source material to look at. Every page that comes over the wire to your browser is in HTML (or perhaps XHTML) format. (You almost never see the codes in your browser; all you see is the final result.)

   Most web browsers have a way of letting you see the HTML source of a web page. If you're using Internet Explorer 6.0, for example, navigate to the web page that you want to look at. Choose View, Source to display the source code in a text window. In Netscape, choose View, Page Source.

TIP

> In some browsers, you can't directly view the source of a web page, but you can save the current page as a file to your local disk. In a dialog box for saving the file, you might find a menu of formats—for example, Text, PostScript, or HTML. You can save the current page as HTML and then open that file in a text editor or word processor to see the HTML source.

Try going to a typical home page and then viewing its source. For example, Figure 3.3 shows the home page for Craigslist, a free online classified ads service search page at http://www.craisglist.org/.

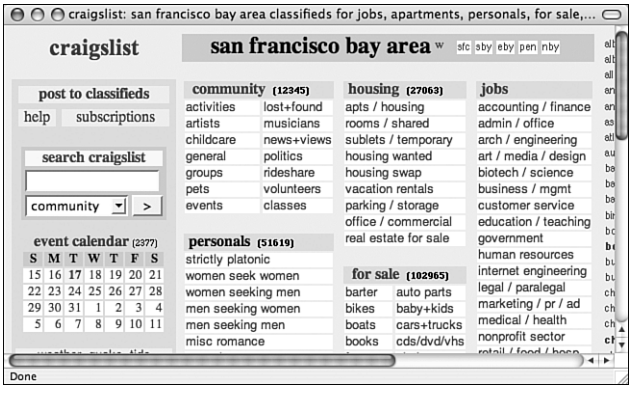
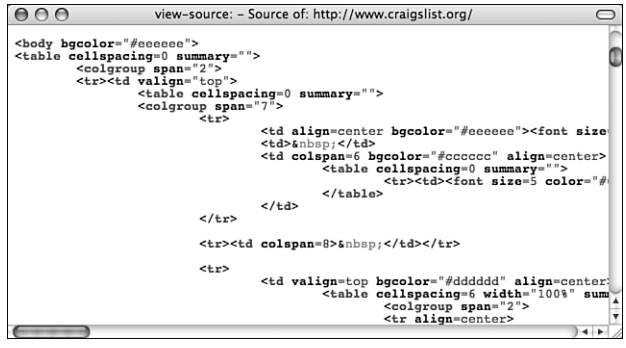


FIGURE 3.3
Craigslist home page.

The HTML source code looks something like Figure 3.4.

FIGURE 3.4
Some HTML source code.



2. Try viewing the source of your own favorite web pages. You should start seeing some similarities in the way pages are organized and get a feel for the kinds of tags that HTML uses. You can learn a lot about HTML by comparing the text onscreen with the source for that text.