

Making Everything Easier!™

3rd Edition

HTML5 and CSS3

ALL-IN-ONE

FOR
DUMMIES®
A Wiley Brand

**8 BOOKS
IN 1**

- Creating the HTML Foundation
- Styling with CSS
- Building Layouts with CSS
- Client-Side Programming with JavaScript®
- Server-Side Programming with PHP
- Managing Data with MySQL®
- Integrating the Client and Server with AJAX
- Moving from Pages to Sites

Andy Harris



Chapter 1: Sound HTML Foundations

In This Chapter

- ✓ Creating a basic web page
- ✓ Understanding the most critical HTML tags
- ✓ Setting up your system to work with HTML
- ✓ Viewing your pages

This chapter is your introduction to building web pages. Before this slim chapter is finished, you'll have your first page up and running. It's a humble beginning, but the basic web technology you learn here is the foundation of everything happening on the web today.

In this minibook, you discover the modern form of web design using HTML5. Your web pages will be designed from the ground up, which makes them easy to modify and customize. Although you figure out more advanced techniques throughout this book, you'll take the humble pages you discover in this chapter and make them do all kinds of exciting things.

Creating a Basic Page

Here's the great news: The most important web technology you need is also the easiest. You don't need any expensive or complicated software, and you don't need a powerful computer. You probably have everything you need to get started already.

No more talking! Fire up a computer and build a web page!

1. Open a text editor.

You can use any text editor you want, as long as it lets you save files as plain text. If you're using Windows, Notepad is fine for now. If you're using Mac, you'll really need to download a text editor. I like Komodo Edit (www.activestate.com/komodo-edit) or TextWrangler (www.barebones.com/products/textwrangler/). It's possible to make TextEdit work correctly, but it's probably easier to just download something made for the job. I explain text editors more completely in Chapter 3 of this mini-book.



Don't use a word processor like Microsoft Word or Mac TextEdit. These are powerful tools, but they don't save things in the right format. The way these tools do things like centering text and changing fonts won't work on the web. I promise that you'll figure out how to do all that stuff soon, but a word processing program won't do it correctly. Even the Save as HTML feature doesn't work right. You really need a very simple text editor, and that's it. In Chapter 3 of this minibook, I show you a few more editors that make your life easier. You should not use Word or TextEdit.

2. Type the following code.

Really. Type it in your text editor so you get some experience writing the actual code. I explain very soon what all this means, but type it now to get a feel for it:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<!-- myFirst.html -->

<title>My very first web page!</title>
</head>

<body>

<h1>This is my first web page!</h1>

<p>
This is the first web page I've ever made,
and I'm extremely proud of it.
It is so cool!
</p>

</body>
</html>
```

3. Save the file as `myFirst.html`.

It's important that your filename has no spaces and ends with the `.html` extension. Spaces cause problems on the Internet (which is, of course, where all good pages go to live), and the `.html` extension is how most computers know that this file is an HTML file (which is another name for a web page). It doesn't matter where you save the file, as long as you can find it in the next step.

4. Open your web browser.

The *web browser* is the program used to look at pages. After you post your page on a web server somewhere, your Great Aunt Gertrude can use her web browser to view your page. You also need one (a browser, not a Great Aunt Gertrude) to test your page. For now, use whatever browser you ordinarily use. Most Windows users already have Internet Explorer installed. If you're a Mac user, you probably have Safari. Linux folks generally have Chrome or Firefox. Any of these are fine. In Chapter 3 of this minibook, I explain why you probably need more than one browser and how to configure them for maximum usefulness.

5. Load your page into the browser.

You can do this a number of ways. You can use the browser's File menu to open a local file, or you can simply drag the file from your Desktop (or wherever) to the open browser window.

6. Bask in your newfound genius.

Your simple text file is transformed! If all went well, it looks like Figure 1-1.

Understanding the HTML in the Basic Page

The page you created in the previous section uses an extremely simple notation — HTML (HyperText Markup Language), which has been around since the beginning of the web. HTML is a terrific technology for several reasons:

- ◆ **It uses plain text.** Most document systems (like word processors) use special *binary encoding schemes* that incorporate formatting directly into the computer's internal language, which locks a document into a particular computer or software. That is, a document stored in Word format can't be read without a program that understands Word formatting. HTML gets past this problem by storing everything in plain text.

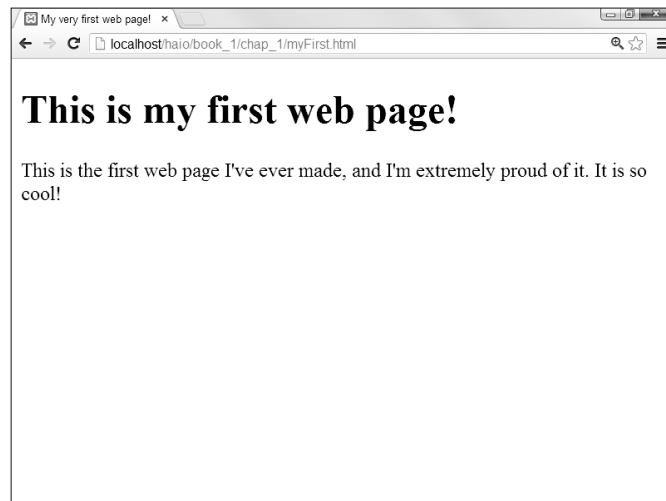


Figure 1-1:
Congratu-
lations!
You're
now a web
developer!

- ◆ **It works on all computers.** The main point of HTML is to have a universal format. Any computer should be able to read and write it. The plain-text formatting aids in this.
- ◆ **It describes what documents *mean*.** HTML isn't really designed to indicate how a page or its elements look. HTML is about describing the meaning of various elements (more on that very soon). This has some distinct advantages when you figure out how to use HTML properly.
- ◆ **It *doesn't* describe how documents *look*.** This one seems strange. Of course, when you look at Figure 1-1, you can see that the appearance of the text on the web page has changed from the way the text looked in your text editor. Formatting a document in HTML does cause the document's appearance to change. That's not the point of HTML, though. You discover in Book II and Book III how to use another powerful technology — CSS — to change the appearance of a page after you define its meaning. This separation of meaning from layout is one of the best features of HTML.
- ◆ **It's easy to write.** Sure, HTML gets a little more complicated than this first example, but you can easily figure out how to write HTML without any specialized editors. You only have to know a handful of elements, and they're pretty straightforward.
- ◆ **It's free.** HTML doesn't cost anything to use, primarily because it isn't owned by anyone. No corporation has control of it (although a couple have tried), and nobody has a patent on it. The fact that this technology is freely available to anyone is a huge advantage.

Meeting Your New Friends, the Tags

The key to writing HTML code is the special text inside angle braces (<>). These special elements are *tags*. They aren't meant to be displayed on the web page, but offer instructions to the web browser about the meaning of the text. The tags are meant to be embedded into each other to indicate the organization of the page. This basic page introduces you to all the major tags you'll encounter. (There are more, but they can wait for a chapter or two.) Each tag has a beginning and an end tag. The end tag is just like the beginning tag, except the end tag has a slash (/):

- ◆ **<!DOCTYPE HTML>:** This special tag is used to inform the browser that the document type is HTML. This is how the browser knows you'll be writing an HTML5 document. You will sometimes see other values for the doctype, but HTML5 is the way to go these days.
- ◆ **<html lang = "en"></html>:** The <html> tag is the foundation of the entire web page. The tag begins the page. Likewise, </html> ends the page. For example, the page begins with <html> and ends with </html>. The <html></html> combination indicates that everything in the page is defined as HTML code. In HTML5, you're expected to tell



the browser which language the page will be written in. Because I write in English, I'm specifying with the code "en."

Some books teach you to write your HTML tags in uppercase letters. This was once a standard, but it is no longer recommended.

- ◆ **<head></head>**: These tags define a special part of the web page called the *head* (or sometimes *header*). This part of the web page reminds me of the engine compartment of a car. This is where you put some great stuff later, but it's not where the main document lives. For now, the only thing you'll put in the header is the document's title. Later, you'll add styling information and programming code to make your pages sing and dance.
- ◆ **<meta charset="UTF-8">**: The meta tag is used to provide a little more information to the browser. This command gives a little more information to the browser, telling it which character set to use. English normally uses a character set called (for obscure reasons) UTF-8. You don't need to worry much about this, but every HTML5 page written in English uses this code.
- ◆ **<!--/-->**: This tag indicates a *comment*, which is ignored by the browser. However, a comment is used to describe what's going on in a particular part of the code.
- ◆ **<title></title>**: This tag is used to determine the page's title. The title usually contains ordinary text. Whatever you define as the title will appear in some special ways. Many browsers put the title text in the browser's title bar. Search engines often use the title to describe the page.

Throughout this book, I use the filename of the HTML code as the title. That way, you can match any figure or code listing to the corresponding file on the web site that accompanies this book. Typically, you'll use something more descriptive, but this is a useful technique for a book like this.



It's not quite accurate to say that the title text always shows up in the title bar because a web page is designed to work on lots of different browsers. Sure, the title does show up on most major browsers that way, but what about cellphones and tablets? HTML never legislates what will happen; it only suggests. This may be hard to get used to, but it's a reality. You trade absolute control for widespread capability, which is a good deal.

- ◆ **<body></body>**: The page's main content is contained within these tags. Most of the HTML code and the stuff the user sees are in the body area. If the header area is the engine compartment, the body is where the passengers go.
- ◆ **<h1></h1>**: H1 stands for *heading level one*. Any text contained within this markup is treated as a prominent headline. By default, most browsers add special formatting to anything defined as H1, but there's no guarantee. An H1 heading doesn't really specify any particular font or formatting, just the *meaning* of the text as a level one heading. When you find out how to use CSS in Book II, you'll discover that you can make your headline look however you want. In this first minibook, keep all the default layouts for now and make sure you understand that HTML is about semantic meaning, not about layout or design. There are other levels of headings, of



course, through `<h6>` where `<h2>` indicates a heading slightly less important than `<h1>`, `<h3>` is less important than `<h2>`, and so on.

Beginners are sometimes tempted to make their first headline an `<h1>` tag and then use an `<h2>` for the second headline and an `<h3>` for the third. That's not how it works. Web pages, like newspapers and books, use different headlines to point out the relative importance of various elements on the page, often varying the point size of the text. You can read more about that in Book II.

- ◆ **`<p></p>`:** In HTML, `p` stands for the paragraph tag. In your web pages, you should enclose each standard paragraph in a `<p></p>` pair. You might notice that HTML doesn't preserve the carriage returns or white space in your HTML document. That is, if you press Enter in your code to move text to a new line, that new line isn't necessarily preserved in the final web page.

The `<p></p>` structure is one easy way to manage spacing before and after each paragraph in your document.



Some older books recommend using `<p>` without a `</p>` to add space to your documents, similar to pressing the Enter key. This way of thinking could cause you problems later because it doesn't accurately reflect the way web browsers work. Don't think of `<p>` as the carriage return. Instead, think of `<p>` and `</p>` as defining a paragraph. The paragraph model is more powerful because soon enough, you'll figure out how to take any properly defined paragraph and give it yellow letters on a green background with daisies (or whatever else you want). If things are marked properly, they'll be much easier to manipulate later.

A few notes about the basic page

Be proud of this first page. It may be simple, but it's the foundation of greater things to come. Before moving on, take a moment to ponder some important HTML principles shown in this humble page you've created:

- ✓ **All tags are lowercase.** Although HTML does allow uppercase tags, modern developers have agreed on lowercase tags in most cases. (`<!DOCTYPE>` is one notable exception to this rule.)
- ✓ **Tag pairs are containers, with a beginning and an end.** Tags contain other tags or text.
- ✓ **Some elements can be repeated.** There's only one `<html>`, `<title>`, and `<body>` tag per page, but a lot of the other elements (`<h1>` and `<p>`) can be repeated as many times as you like.
- ✓ **Carriage returns are ignored.** In the Notepad document, there are a number of carriage returns. The formatting of the original document has no effect on the HTML output. The markup tags indicate how the output looks.

Setting Up Your System

You don't need much to make web pages. Your plain text editor and a web browser are about all you need. Still, some things can make your life easier as a web developer.

Displaying file extensions

The method discussed in this section is mainly for Windows users, but it's a big one. Windows uses the *extension* (the part of the filename after the period) to determine what type of file you're dealing with. This is very important in web development. The files you create are simple text files, but if you store them with the ordinary `.txt` extension, your browser can't read them properly. What's worse, the default Windows setting hides these extensions from you, so you have only the icons to tell you what type of file you're dealing with, which causes all kinds of problems. I recommend you have Windows explicitly describe your file extensions. Here's how to set that up in Windows 7:

1. **Click the Start button.**

This opens the standard Start menu.

2. **Open the Control Panel.**

The Control Panel application allows you to modify many parts of your operating system.

3. **Find Appearance and Personalization.**

This section allows you to modify the visual look and feel of your operating system.

4. **Choose Folder Options.**

This dialog box lets you modify the way folders look throughout the visual interface.

5. **Find Advanced Settings.**

Click the View tab and then look under Advanced Settings.

6. **Display file extensions.**

By default, the Hide Extensions for Known File Types check box is selected. Deselect this check box to display file extensions.

The process for displaying file types is similar in Windows 8:

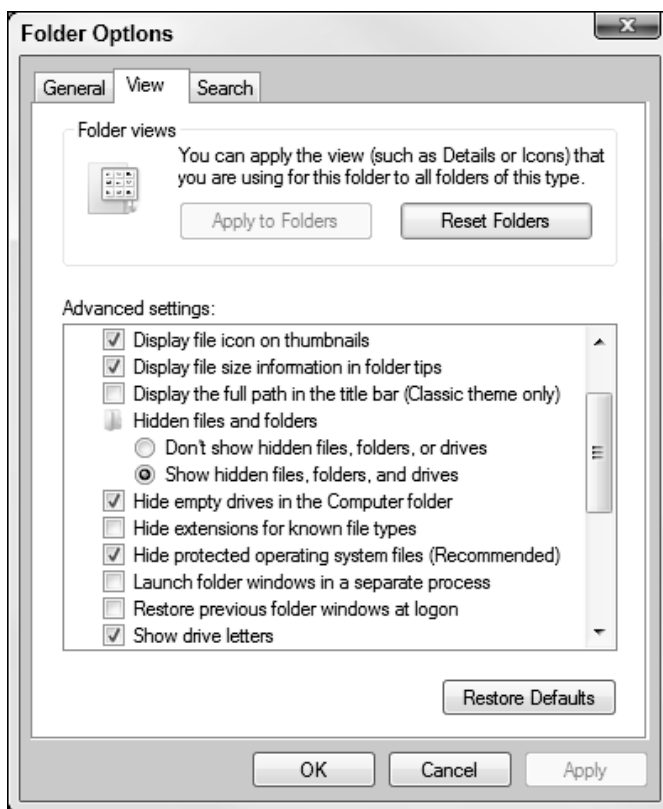
1. **Go to Windows Explorer.**

Use the Windows Explorer tile to view Windows Explorer — the standard file manager for Windows.

2. **Click the View tab.**

This tab allows you to modify how directories look.

Figure 1-2:
Don't
hide file
extensions
(deselect
that Hide
Extensions
check box).



3. De-select filename extensions.

If this button is checked, file extensions are shown (which is what you want.) (See Figure 1-2.) Note this is the opposite of Windows 7's behavior.



Although my demonstration uses Windows 7 and 8, the technique is similar in older versions of Windows. Just do a quick search for "displaying file extensions."

Setting up your software

You'll write a lot of web pages, so it makes sense to set up your system to make that process as easy as possible. I talk a lot more about some software you should use in Chapter 3 of this minibook, but for now, here are a couple of easy suggestions:

- ◆ **Put a Notepad icon on your Desktop.** You'll edit a lot of text files, so it's helpful to have an icon for Notepad (or whatever other text editor you

use) available directly on the Desktop. That way, you can quickly edit any web page by dragging it to the Desktop. When you use more sophisticated editors than Notepad, you'll want links to them, too.

- ◆ **Get another web browser.** You may just *love* your web browser, and that's fine, but you can't assume that everybody likes the same browser you do. You need to know how other browsers interpret your code. Chrome is an incredibly powerful browser, and it's completely free, as well as having a lot of great programmer's features. If you don't already, I suggest having links to at least two browsers directly on your Desktop.

Understanding the magic

Most of the problems people have with the web are from misunderstandings about how this medium really works. Most people are comfortable with word processors, and we know how to make a document look how we want. Modern applications use WYSIWYG technology, promising that *what you see is what you get*. That's a reasonable promise when it comes to print documents, but it doesn't work that way on the web.

How a web page looks depends on a lot of things that you don't control. The user may read your pages on a smaller or larger screen than you. She may use a different operating system than you. She may have a slower connection or may turn off the graphics for speed. She may be blind and use screen-reader technology to navigate web pages. She may be reading your page on a tablet, smart phone,

or even an older (not so smart) cellphone. You can't make a document that looks the same in all these situations.

A good compromise is to make a document that clearly indicates how the information fits together and makes suggestions about the visual design. The user and her browser can determine how much of those suggestions to use.

You get some control of the visual design but never complete control, which is okay because you're trading total control for accessibility. People with devices you've never heard of can visit your page.

Practice a few times until you can easily build a page without looking anything up. Soon enough, you're ready for the next step — building pages like the pros.

Chapter 2: It's All About Validation

In This Chapter

- ✓ Introducing the concept of valid pages
- ✓ Using a doctype
- ✓ Setting the character set
- ✓ Meeting the W3C validator
- ✓ Fixing things when they go wrong
- ✓ Using HTML Tidy to clean your pages

Web development is undergoing a revolution. As the web matures and becomes a greater part of everyday life, it's important to ensure that web pages perform properly — thus, a call for web developers to follow voluntary standards of web development.

Somebody Stop the HTML Madness!

In the bad old days, the web was an informal affair. People wrote HTML pages any way they wanted. Although this was easy, it led to a lot of problems:

- ◆ **Browser manufacturers added features that didn't work on all browsers.** People wanted prettier web pages with colors, fonts, and doodads, but there wasn't a standard way to do these things. Every browser had a different set of tags that supported enhanced features. As a developer, you had no real idea if your web page would work on all the browsers out there. If you wanted to use some neat feature, you had to ensure your users had the right browser.
- ◆ **The distinction between meaning and layout was blurred.** People expected to have some kind of design control of their web pages, so all kinds of new tags popped up that blurred the distinction between describing and decorating a page.
- ◆ **Table-based layout was used as a hack.** HTML didn't have a good way to handle layout, so clever web developers started using tables as a layout mechanism. This worked, after a fashion, but it wasn't easy or elegant.
- ◆ **People started using tools to write pages.** Web development soon became so cumbersome that people began to believe that they couldn't do HTML by hand anymore and that some kind of editor was necessary

to handle all that complexity for them. Although these editing programs introduced new features that made things easier upfront, these tools also made code almost impossible to change without the original editor. Web developers began thinking they couldn't design web pages without a tool from a major corporation.

- ♦ **The nature of the web was changing.** At the same time, these factors were making ordinary web development more challenging. Innovators were recognizing that the web wasn't really about documents but was about applications that could dynamically create documents. Many of the most interesting web pages you visit aren't web pages at all, but programs that produce web pages dynamically every time you visit. This innovation meant that developers had to make web pages readable by programs, as well as humans.
- ♦ **XHTML tried to fix things.** The standards body of the web (there really is such a thing) is called the World Wide Web Consortium (W3C), and it tried to resolve things with a new standard called XHTML. This was a form of HTML that also followed the much stricter rules of XML. If everyone simply agreed to follow the XHTML standard, much of the ugliness would go away.
- ♦ **XHTML didn't work either.** Although XHTML was a great idea, it turned out to be complicated. Parts of it were difficult to write by hand, and very few developers followed the standards completely. Even the browser manufacturers didn't agree exactly on how to read and display XHTML. It doesn't matter how good an idea is if nobody follows it.

In short, the world of HTML was a real mess.

XHTML had some great ideas

In 2000, the World Wide Web Consortium (usually abbreviated as W3C) got together and proposed some fixes for HTML. The basic plan was to create a new form of HTML that complied with a stricter form of markup, or *eXtensible Markup Language (XML)*. The details are long and boring, but essentially, they came up with some agreements about how web pages are standardized. Here are some of those standards:

- ♦ **All tags have endings.** Every tag comes with a beginning and an end tag. (Well, a few exceptions come with their own ending built in. I'll explain when you encounter the first such tag in Chapter 6 of this minibook.) This was a new development because end tags were considered optional in old-school HTML, and many tags didn't even have end tags.
- ♦ **Tags can't be overlapped.** In HTML, sometimes people had the tendency to be sloppy and overlap tags, like this: `<a>my stuff`. That's not allowed in XHTML, which is a good thing because it confuses the browser. If a tag is opened inside some container tag, the tag must be closed before that container is closed.

- ◆ **Everything's lowercase.** Some people wrote HTML in uppercase, some in lowercase, and some just did what they felt like. It was inconsistent and made it harder to write browsers that could read all the variations.
- ◆ **Attributes must be in quotes.** If you've already done some HTML, you know that quotes used to be optional — not anymore. (Turn to Chapter 3 for more about attributes.)
- ◆ **Layout must be separate from markup.** Old-school HTML had a bunch of tags (like `` and `<center>`) that were more about formatting than markup. These were useful, but they didn't go far enough. XHTML (at least the strict version) eliminates all these tags. Don't worry, though; CSS gives you all the features of these tags and a lot more.

This sounds like strict librarian rules, but really they aren't restricting at all. Most of the good HTML coders were already following these guidelines or something similar.

Even though you're moving past XHTML into HTML5, these aspects of XHTML remain, and they are guidelines all good HTML5 developers still use.



HTML5 actually allows a looser interpretation of the rules than XHTML strict did, but throughout this book I write HTML5 code in a way that also passes most of the XHTML strict tests. This practice ensures nice clean code with no surprises.

You validate me

In old-style HTML, you never really knew how your pages would look on various browsers. In fact, you never really knew if your page was even written properly. Some mistakes would look fine on one browser but cause another browser to blow up.

The idea of *validation* is to take away some of the uncertainty of HTML. It's like a spell checker for your code. My regular spell checker makes me feel a little stupid sometimes because I make mistakes. I like it, though, because I'm the only one who sees the errors. I can fix the spelling errors before I pass the document on to you, so I look smart. (Well, maybe.)

It'd be cool if you could have a special kind of checker that does the same things for your web pages. Instead of checking your spelling, it'd test your page for errors and let you know if you made any mistakes. It'd be even cooler if you could have some sort of certification that your page follows a standard of excellence.

That's how page validation works. You can designate that your page will follow a particular standard and use a software tool to ensure that your page meets that standard's specifications. The software tool is a *validator*. I show you two different validators in the upcoming "Validating Your Page" section.

The browsers also promise to follow a particular standard. If your page validates to a given standard, any browser that validates to that same standard can reproduce your document correctly, which is a big deal.

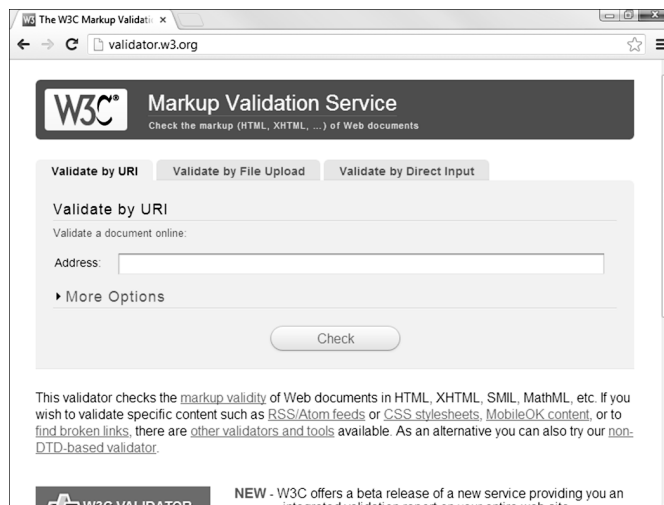
The most important validator is the W3C validator at <http://validator.w3.org>, as shown in Figure 2-1.

A validator is actually the front end of a piece of software that checks pages for validity. It looks at your web page's doctype and sees whether the page conforms to the rules of that doctype. If not, it tells you what might have gone wrong.

You can submit code to a validator in three ways:

- ◆ **Validate by URI.** This option is used when a page is hosted on a web server. Files stored on local computers can't be checked with this technique. Book VIII describes all you need to know about working with web servers, including how to create your own and move your files to it. (A *URI*, or uniform resource identifier, is a more formal term for a web address, which is more frequently seen as URL.)
- ◆ **Validate by file upload.** This technique works fine with files you haven't posted to a web server. It works great for pages you write on your computer but that you haven't made visible to the world. This is the most common type of validation for beginners.
- ◆ **Validate by direct input.** The validator page has a text box you can simply paste your code into. It works, but I usually prefer to use the other methods because they're easier.

Figure 2-1:
The W3C
validator
page isn't
exciting,
but it sure is
useful.



Validation might sound like a big hassle, but it's really a wonderful tool because sloppy HTML code can cause lots of problems. Worse, you might think everything's okay until somebody else looks at your page, and suddenly, the page doesn't display correctly.



As of this writing, the W3C validator can read and test HTML5 code, but the HTML5 validation is still considered experimental. Until HTML5 becomes a bit more mainstream, your HTML5 pages may get a warning about the experimental nature of HTML5. You can safely ignore this warning.

Validating Your Page

To explain all this, I created a web page the way Aesop might have done in ancient Greece. Okay, maybe Aesop didn't write his famous fables as web pages, but if he had, they might have looked like the following code listing:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">

<!-- oxWheels1.html -->

<!-- note this page has deliberate errors! Please see the text
      and oxWheelsCorrect.html for a corrected version.
-->

</head>
<body>
<title>The Oxen and the Wheels</title>
<h1>The Oxen and the Wheels
<h2></h1>From Aesop's Fables</h2>

<p>
  A pair of Oxen were drawing a heavily loaded wagon along a
  miry country road. They had to use all their strength to pull
  the wagon, but they did not complain.
</p>

<p>
  The Wheels of the wagon were of a different sort. Though the
  task they had to do was very light compared with that of the
  Oxen, they creaked and groaned at every turn. The poor Oxen,
  pulling with all their might to draw the wagon through the
  deep mud, had their ears filled with the loud complaining of
  the Wheels. And this, you may well know, made their work so
  much the harder to endure.
</p>

<p>
  "Silence!" the Oxen cried at last, out of patience. "What have
  you Wheels to complain about so loudly? We are drawing all the
  weight, not you, and we are keeping still about it besides."
</p>

<h2>
  They complain most who suffer least.
```



```
</h2>

</body>
</html>
```

The code looks okay, but actually has a number of problems. Aesop may have been a great storyteller, but from this example, it appears he was a sloppy coder. The mistakes can be hard to see, but trust me, they're there. The question is, how do you find the problems before your users do?

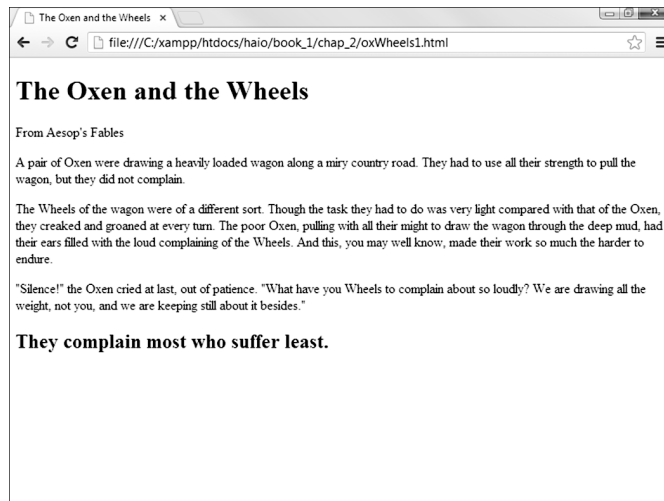
You might think that the problems would be evident if you viewed the page in a web browser. The various web browsers seem to handle the page decently, even if they don't display it in an identical way. Figure 2-2 shows oxWheels1.html in a browser.

Chrome appears to handle the page pretty well, but *From Aesop's Fables* is supposed to be a headline level two, or *H2*, and it appears as plain text. Other than that, there's very little indication that something is wrong.

If it looks fine, who cares if it's exactly right? You might wonder why we care if there are mistakes in the underlying code, as long as everything works okay. After all, who's going to look at the code if the page displays properly?

The problem is, you don't know if it'll display properly, and mistakes in your code will eventually come back to haunt you. If possible, you want to know immediately what parts of your code are problematic so you can fix them and not worry.

Figure 2-2:
The page
looks okay,
but the
headings
are strange.



Aesop visits W3C

To find out what's going on with this page, pay a visit to the W3C validator at <http://validator.w3.org>. Figure 2-3 shows me visiting this site and uploading a copy of `oxWheels1.html` to it.

Hold your breath and click the Check button. You might be surprised at the results shown in Figure 2-4.

The validator is a picky beast, and it doesn't seem to like this page at all. The validator does return some useful information and gives enough hints that you can decode things soon enough.

Figure 2-3:
I'm
checking
the
oxWheels
page to
look for any
problems.

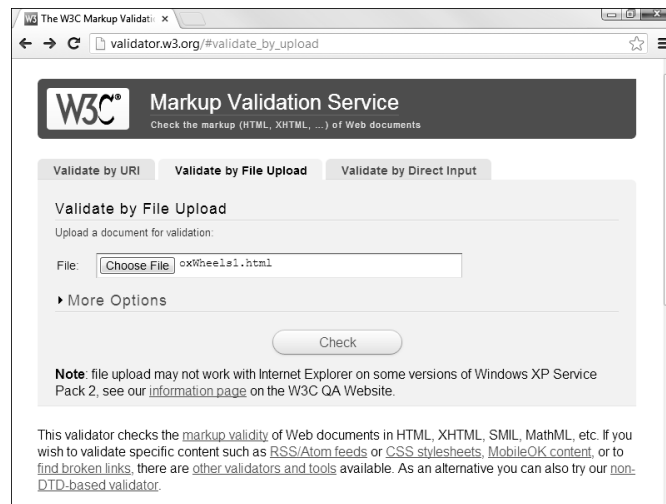
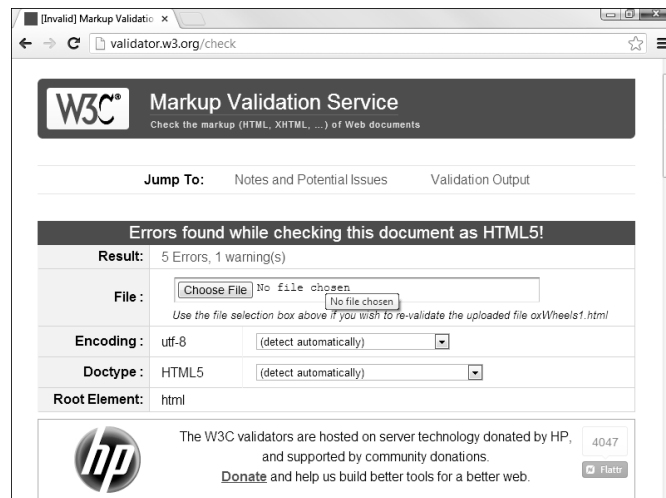


Figure 2-4:
Five errors?
That can't
be right!



Examining the overview

Before you look at the specific complaints, take a quick look at the web page the validator sends you. The web page is chock-full of handy information. The top of the page tells you a lot of useful things:

- ◆ **Result:** This is really the important thing. You'll know the number of errors remaining by looking at this line. Don't panic, though. The errors in the document are probably fewer than the number you see here.
- ◆ **File:** The name of the file you're working on.
- ◆ **Encoding:** The text encoding you've set. If you didn't explicitly set text encoding, you may see a warning here.
- ◆ **Doctype:** This is the doctype extracted from your document. It indicates the rules that the validator is using to check your page. This should usually say HTML5.
- ◆ **The dreaded red banner:** Experienced web developers don't even have to read the results page to know if there is a problem. If everything goes well, there's a green congratulatory banner. If there are problems, the banner is red. It doesn't look good, Aesop.



Don't panic because you have errors. The mistakes often overlap, so one problem in your code often causes more than one error to pop up. Most of the time, you have far fewer errors than the page says, and a lot of the errors are repeated, so after you find the error once, you'll know how to fix it throughout the page.

Validating the page

The validator doesn't always tell you everything you need to know, but it does give you some pretty good clues. Page validation is tedious but not as difficult as it might seem at first. Here are some strategies for working through page validation:

- ◆ **Focus only on the first error.** Sure, 100 errors might be on the page, but solve them one at a time. The only error that matters is the first one on the list. Don't worry at all about other errors until you've solved the first one.
- ◆ **Note where the first error is.** The most helpful information you get is the line and column information about where the validator recognized the error. This isn't always where the error is, but it does give you some clues.
- ◆ **Look at the error message.** It's usually good for a laugh. The error messages are sometimes helpful and sometimes downright mysterious.
- ◆ **Look at the verbose text.** Unlike most programming error messages, the W3C validator tries to explain what went wrong in something like English. It still doesn't always make sense, but sometimes the text gives you a hint.

- ◆ **Scan the next couple of errors.** Sometimes, one mistake shows up as more than one error. Look over the next couple of errors, as well, to see if they provide any more insight; sometimes, they do.
- ◆ **Try a change and revalidate.** If you've got an idea, test it out (but only solve one problem at a time.) Check the page again after you save it. If the first error is now at a later line number than the previous one, you've succeeded.
- ◆ **Don't worry if the number of errors goes up.** The number of perceived errors will sometimes go up rather than down after you successfully fix a problem. This is okay. Sometimes, fixing one error uncovers errors that were previously hidden. More often, fixing one error clears up many more. Just concentrate on clearing errors from the beginning to the end of the document.
- ◆ **Lather, rinse, and repeat.** Look at the new top error and get it straightened out. Keep going until you get the coveted Green Banner of Validation. (If I ever write an HTML adventure game, the Green Banner of Validation will be one of the most powerful talismans.)

Examining the first error

Look again at the results for the oxWheels1.html page. The first error message looks like Figure 2-5.

Figure 2-5:
Well, that
clears
every-
thing up.

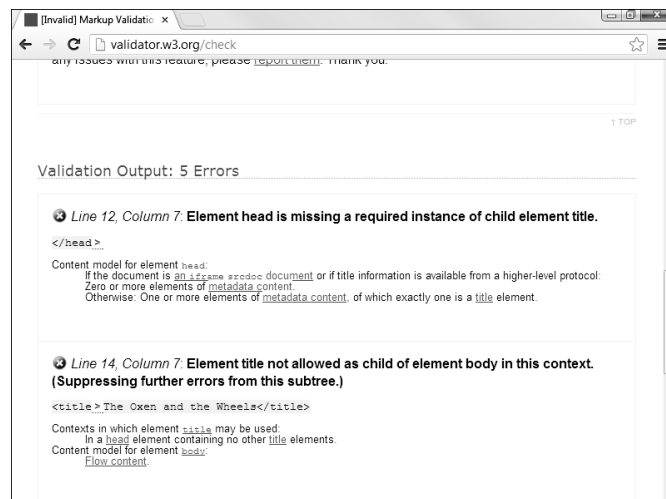


Figure 2-5 shows the first two error messages. The first complains that the head is missing a title. The second error message is whining about the title being in the body. The relevant code is repeated here:

```
<!DOCTYPE HTML>
<html lang="en-US">
```

```
<head>
  <meta charset="UTF-8">

<!-- oxWheels1.html -->

<!-- note this page has deliberate errors! Please see the text
      and oxWheelsCorrect.html for a corrected version.
-->

</head>
<body>
<title>The Oxen and the Wheels</title>
```

Look carefully at the `head` and `title` tag pairs and review the notes in the error messages, and you'll probably see the problem. The `<title>` element is supposed to be in the heading, but I accidentally put it in the body! (Okay, it wasn't accidental; I made this mistake deliberately here to show you what happens. However, I have made this mistake for real in the past.)

Fixing the title

If the `title` tag is the problem, a quick change in the HTML should fix it. `oxWheels2.html` shows another form of the page with my proposed fix:

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

<!-- oxWheels2.html -->

<!-- Moved the title tag inside the header -->
<title>The Oxen and the Wheels</title>

</head>
<body>
```

Note: I'm only showing the parts of the page that I changed. The entire page is available on this book's website. See this book's Introduction for more on the website.

The fix for this problem is pretty easy:

1. **Move the title inside the head.** I think the problem here is having the `<title>` element inside the body, rather than in the head where it belongs. If I move the title to the body, the error should be eliminated.
2. **Change the comments to reflect the page's status.** It's important that the comments reflect what changes I make.
3. **Save the changes.** Normally, you simply make a change to the same document, but I've elected to change the filename so you can see an archive of my changes as the page improves. This can actually be a good idea because you then have a complete history of your document's changes, and you can always revert to an older version if you accidentally make something worse.

4. **Note the current first error position.** Before you submit the modified page to the validator, make a mental note of the position of the current first error. Right now, the validator's first complaint is on line 12, column 7. I want the first mistake to be somewhere later in the document.
5. **Revalidate by running the validator again on the modified page.**
6. **Review the results and do a happy dance.** It's likely you still have errors, but that's not a failure! Figure 2-6 shows the result of my revalidation. The new first error is on line 17, and it appears to be very different from the last error. I solved it!

Solving the next error

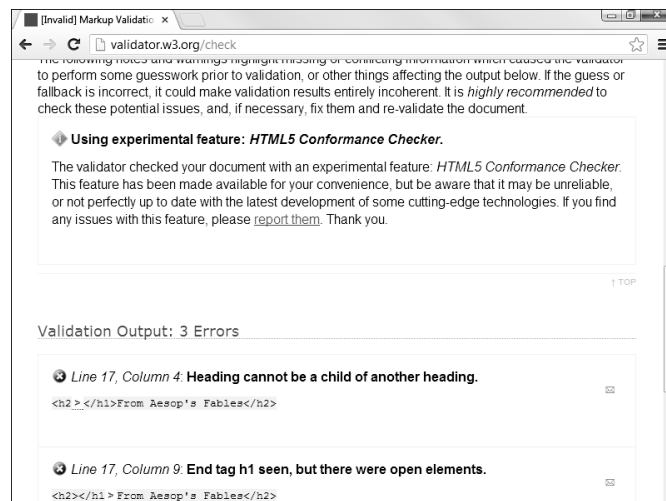
One down, but more to go. The next error (refer to Figure 2-6) looks strange, but it makes sense when you look over the code.

This type of error is very common. What it usually means is you forgot to close something or you put something in the wrong place. The error message indicates a problem in line 17. The next error is line 17, too. See if you can find the problem here in the relevant code:

```
<body>
<h1>The Oxen and the Wheels
<h2></h1>From Aesop's Fables</h2>
```

After you know where to look, the problem becomes a bit easier to spot. I got sloppy and started the `<h2>` tag before I finished the `<h1>`. In many cases, one tag can be completely embedded inside another, but you can't have tag definitions overlap as I've done here. The `<h1>` has to close before I can start the `<h2>` tag.

Figure 2-6:
Heading
cannot be
a child of
another
heading.
Huh?



This explains why browsers might be confused about how to display the headings. It isn't clear whether this code should be displayed in H1 or H2 format, or perhaps with no special formatting at all. It's much better to know the problem and fix it than to remain ignorant until something goes wrong.

The third version — `oxWheels3.html` — fixes this part of the program:

```
<!-- oxWheels3.html -->
<!-- sort out the h1 and h2 tags at the top -->
<title>The Oxen and the Wheels</title>
</head>
<body>
<h1>The Oxen and the Wheels</h1>
<h2>From Aesop's Fables</h2>
```

The validator has fixed a number of errors, but there's one really sneaky problem still in the page. See if you can find it, and then read ahead.

Using Tidy to repair pages

The W3C validator isn't the only game in town. Another great resource — HTML Tidy — can be used to fix your pages. You can download Tidy or just use the online version at <http://infohound.net/tidy>. Figure 2-7 illustrates the online version.

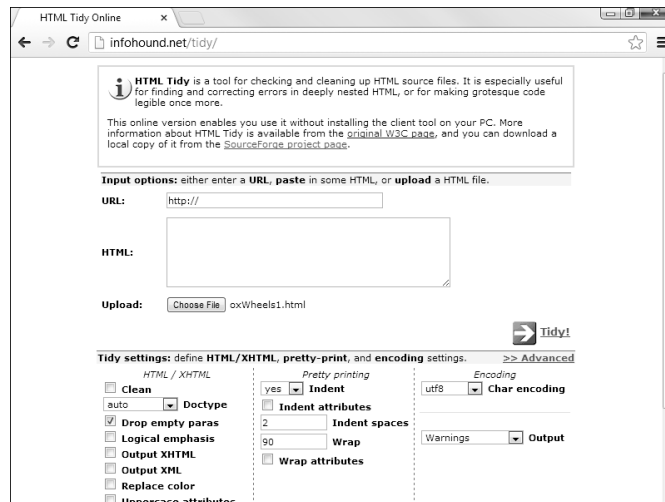
Is validation really that big a deal?

I can hear the angry e-mails coming in. "Andy, I've been writing web pages since 1998, and I never used a validator." Okay, it's true. A lot of people, even some professional web developers, work without validating their code. Some of my older web pages don't validate at all. (You can run the W3C validator on any page you want, not just one you wrote. This can be a source of great joy if you like feeling superior to sloppy coders.) When I became more proficient and more prolific in my web development, I found that those little errors often caused a whole lot of grief down the road. I really believe you should validate every single page you write. Get into the habit now, and it'll pay huge dividends. When you're figuring out this stuff for the first time, do it right.

If you already know some HTML, you're gonna hate the validator for a while because it rejects coding habits that you might think are perfectly fine. Unlearning a habit is a lot harder than learning a new practice, so I feel your pain. It's still worth it.

After you discipline yourself to validate your pages, you'll find you've picked up good habits, and validation becomes a lot less painful. Experienced programmers actually like the validation process because it becomes much easier and prevents problems that could cause lots of grief later. You may even want to re-validate a page you've been using for a while. Sometimes a content update can cause mistakes.

Figure 2-7:
HTML
Tidy is an
alternative
to the W3C
validator.

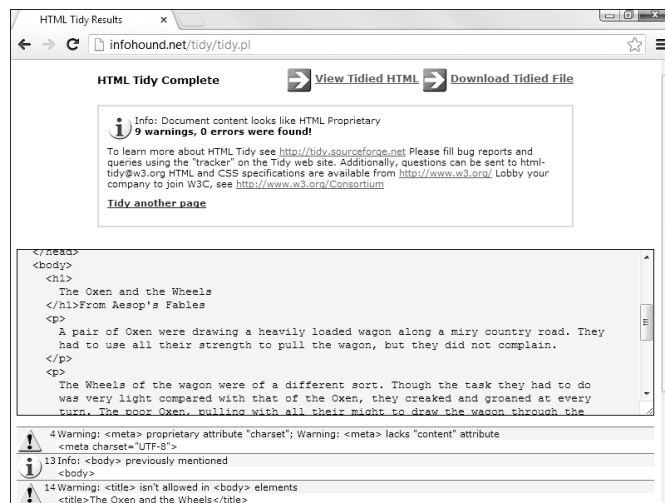


Unlike W3C's validator, Tidy actually attempts to fix your page. Figure 2-8 displays how Tidy suggests the oxWheels1.html page be fixed.

Tidy examines the page for a number of common errors and does its best to fix the errors. However, the result is not quite perfect:

- ◆ **It outputs XHTML by default.** XHTML is fine, but because we're doing HTML here, deselect the Output XHTML box. The only checkbox you need selected is Drop Empty Paras.

Figure 2-8:
Tidy fixes
the page,
but the fix
is a little
awkward.



- ◆ **Tidy got confused by the headings.** Tidy correctly fixed the level one heading, but it had trouble with the level two heading. It removed all the tags, so it's valid, but the text intended to be a level two heading is just sort of hanging there.
- ◆ **Sometimes, the indentation is off.** I set Tidy to indent every element, so it is easy to see how tag pairs are matched up. If I don't set up the indentation explicitly, I find Tidy code very difficult to read.
- ◆ **The changes aren't permanent.** Anything Tidy does is just a suggestion. If you want to keep the changes, you need to save the results in your editor. Click the Download Tidied File button to do this easily.

I sometimes use Tidy when I'm stumped because I find the error messages are easier to understand than the W3C validator. However, I never trust it completely. Until it's updated to truly understand HTML5, it sometimes deletes perfectly valid HTML5 tags. There's really no substitute for good old detective skills and the official W3C validator.

Did you figure out that last error? I tried to close a paragraph with `<p>` rather than `</p>`. That sort of thing freaks out an XHTML validator, but HTML takes it in stride, so you might not even know there is a problem. Tidy does notice the problem and repairs it. Remember this when you're working with a complex page and something doesn't seem right. It's possible there's a mistake you can't even see, and it's messing you up. In that case, consider using a validator and Tidy to figure out what's going wrong and fix it.

Contents at a Glance



<i>Introduction</i>	<i>1</i>
<i>Part I: Creating the HTML Foundation.....</i>	<i>7</i>
Chapter 1: Sound HTML Foundations	9
Chapter 2: It's All About Validation	19
Chapter 3: Choosing Your Tools.....	33
Chapter 4: Managing Information with Lists and Tables	51
Chapter 5: Making Connections with Links	67
Chapter 6: Adding Images, Sound, and Video	77
Chapter 7: Creating Forms.....	105
<i>Part II: Styling with CSS.....</i>	<i>129</i>
Chapter 1: Coloring Your World	131
Chapter 2: Styling Text	149
Chapter 3: Selectors: Coding with Class and Style	175
Chapter 4: Borders and Backgrounds	197
Chapter 5: Levels of CSS.....	225
Chapter 6: CSS Special Effects.....	245
<i>Part III: Building Layouts with CSS</i>	<i>263</i>
Chapter 1: Fun with the Fabulous Float	265
Chapter 2: Building Floating Page Layouts	285
Chapter 3: Styling Lists and Menus	309
Chapter 4: Using Alternative Positioning.....	327
<i>Part IV: Client-Side Programming with JavaScript.....</i>	<i>353</i>
Chapter 1: Getting Started with JavaScript	355
Chapter 2: Talking to the Page	375
Chapter 3: Decisions and Debugging.....	399
Chapter 4: Functions, Arrays, and Objects.....	429
Chapter 5: Getting Valid Input.....	459
Chapter 6: Drawing on the Canvas	483
Chapter 7: Animation with the Canvas	511

Part V: Server-Side Programming with PHP 527

Chapter 1: Getting Started on the Server.....	529
Chapter 2: PHP and HTML Forms	549
Chapter 3: Using Control Structures	569
Chapter 4: Working with Arrays	587
Chapter 5: Using Functions and Session Variables	605
Chapter 6: Working with Files and Directories	617
Chapter 7: Exceptions and Objects	639

Part VI: Managing Data with MySQL 653

Chapter 1: Getting Started with Data	655
Chapter 2: Managing Data with MySQL.....	679
Chapter 3: Normalizing Your Data.....	705
Chapter 4: Putting Data Together with Joins	719
Chapter 5: Connecting PHP to a MySQL Database	741

Part VII: Integrating the Client and Server with AJAX 759

Chapter 1: AJAX Essentials.....	761
Chapter 2: Improving JavaScript and AJAX with jQuery	775
Chapter 3: Animating jQuery	795
Chapter 4: Using the jQuery User Interface Toolkit	819
Chapter 5: Improving Usability with jQuery.....	841
Chapter 6: Working with AJAX Data.....	859
Chapter 7: Going Mobile	883

Part VIII: Moving from Pages to Sites..... 909

Chapter 1: Managing Your Servers	911
Chapter 2: Planning Your Sites	933
Chapter 3: Introducing Content Management Systems	953
Chapter 4: Editing Graphics	977
Chapter 5: Taking Control of Content	995

Index 1015

Table of Contents



***Introduction*..... 1**

About This Book.....	1
Foolish Assumptions.....	2
Use Any Computer.....	3
Don't Buy Any Software.....	3
How This Book Is Organized.....	3
New for the Third Edition.....	4
Icons Used in This Book.....	5
Beyond the Book.....	6
Where to Go from.....	6

***Part I: Creating the HTML Foundation*..... 7**

Chapter 1: Sound HTML Foundations9

Creating a Basic Page.....	9
Understanding the HTML in the Basic Page.....	11
Meeting Your New Friends, the Tags.....	12
Setting Up Your System.....	15
Displaying file extensions.....	15
Setting up your software.....	16

Chapter 2: It's All About Validation.....19

Somebody Stop the HTML Madness!.....	19
XHTML had some great ideas.....	20
Validating Your Page.....	23
Aesop visits W3C.....	25
Using Tidy to repair pages.....	30

Chapter 3: Choosing Your Tools.....33

What's Wrong with the Big Boys: Expression Web and Adobe Dreamweaver.....	33
How About Online Site Builders?.....	34
Alternative Web Development Tools.....	35
Picking a Text Editor.....	35
Tools to avoid unless you have nothing else.....	36
Suggested programmer's editors.....	36
My Personal Choice: Komodo Edit.....	41
Other text editors.....	43
The bottom line on editors.....	44

Finding a Good Web Developer's Browser	44
A little ancient history	44
Overview of the prominent browsers	46
Other notable browsers	48
The bottom line in browsers	49
Chapter 4: Managing Information with Lists and Tables	51
Making a List and Checking It Twice	51
Creating an unordered list	51
Creating ordered lists	53
Making nested lists	54
Building the definition list	57
Building Tables	59
Defining the table	60
Spanning rows and columns	63
Avoiding the table-based layout trap	65
Chapter 5: Making Connections with Links	67
Making Your Text Hyper	67
Introducing the anchor tag	68
Comparing block-level and inline elements	69
Analyzing an anchor	69
Introducing URLs	70
Making Lists of Links	71
Working with Absolute and Relative References	73
Understanding absolute references	73
Introducing relative references	73
Chapter 6: Adding Images, Sound, and Video	77
Adding Images to Your Pages	77
Linking to an image	78
Adding inline images using the tag	80
src (source)	81
height and width	81
alt (alternate text)	81
Choosing an Image Manipulation Tool	82
An image is worth 3.4 million words	82
Introducing IrfanView	84
Choosing an Image Format	85
BMP	85
JPG/JPEG	86
GIF	86
PNG	88
SVG	89
Summary of web image formats	90
Manipulating Your Images	90
Changing formats in IrfanView	90
Resizing your images	91
Enhancing image colors	92

Using built-in effects	93
Other effects you can use	97
Batch processing	98
Working with Audio.....	99
Adding video.....	101

Chapter 7: Creating Forms. 105

You Have Great Form.....	105
Forms must have some form.....	107
Building Text-Style Inputs	109
Making a standard text field.....	109
Building a password field	111
Making multi-line text input.....	112
Creating Multiple Selection Elements	114
Making selections	114
Building check boxes.....	116
Creating radio buttons	117
Pressing Your Buttons	119
Making input-style buttons	120
Building a Submit button	121
It's a do-over: The Reset button.....	121
Introducing the <button> tag	121
New form input types.....	122
date	122
time	123
datetime	123
datetime-local.....	123
week.....	124
month	125
color.....	125
number	125
range.....	126
search	126
email	127
tel	127
url.....	127

Part II: Styling with CSS..... 129

Chapter 1: Coloring Your World. 131

Now You Have an Element of Style	131
Setting up a style sheet	133
Changing the colors.....	134
Specifying Colors in CSS	134
Using color names	135
Putting a hex on your colors	136
Coloring by number.....	136

Hex education.....	137
Using the web-safe color palette.....	139
Choosing Your Colors.....	140
Starting with web-safe colors	141
Modifying your colors	141
Doing it on your own pages	141
Changing CSS on the fly.....	142
Creating Your Own Color Scheme.....	143
Understanding hue, saturation, and lightness	143
Using HSL colors in your pages	145
Using the Color Scheme Designer.....	146
Selecting a base hue	147
Picking a color scheme	148
Chapter 2: Styling Text	149
Setting the Font Family	149
Applying the font-family style attribute.....	150
Using generic fonts	151
Making a list of fonts	153
The Curse of Web-Based Fonts	154
Understanding the problem	154
Using Embedded Fonts	155
Using images for headlines	158
Specifying the Font Size	160
Size is only a suggestion!.....	160
Using the font-size style attribute.....	161
Absolute measurement units	162
Relative measurement units.....	163
Determining Other Font Characteristics	164
Using font-style for italics	165
Using font-weight for bold	166
Using text-decoration	167
Using text-align for basic alignment	169
Other text attributes.....	170
Using the font shortcut	171
Working with subscripts and superscripts	172
Chapter 3: Selectors: Coding with Class and Style	175
Selecting Particular Segments.....	175
Defining more than one kind of paragraph.....	175
Styling identified paragraphs	176
Using Emphasis and Strong Emphasis.....	177
Modifying the Display of em and strong.....	179
Defining Classes	180
Adding classes to the page	181

Using classes	182
Combining classes	182
Introducing div and span.....	184
Organizing the page by meaning.....	185
Why not make a table?	186
Using Pseudo-Classes to Style Links	187
Styling a standard link.....	187
Styling the link states	187
Best link practices	189
Selecting in Context.....	190
Defining Styles for Multiple Elements	191
Using New CSS3 Selectors	193
attribute selection	193
not.....	194
nth-child	194
Other new pseudo-classes.....	195
Chapter 4: Borders and Backgrounds.	197
Joining the Border Patrol	197
Using the border attributes	197
Defining border styles	199
Using the border shortcut	200
Creating partial borders.....	201
Introducing the Box Model.....	202
Borders, margin, and padding.....	203
Positioning elements with margins and padding.....	205
New CSS3 Border Techniques.....	207
Image borders	207
Adding Rounded Corners	209
Adding a box shadow	210
Changing the Background Image.....	212
Getting a background check.....	214
Solutions to the background conundrum.....	215
Manipulating Background Images	218
Turning off the repeat	218
Using CSS3 Gradients	219
Using Images in Lists	223
Chapter 5: Levels of CSS	225
Managing Levels of Style	225
Using local styles	225
Using an external style sheet	228
Understanding the Cascading Part of Cascading Style Sheets	233
Inheriting styles	233
Hierarchy of styles.....	234

Overriding styles.....	235
Precedence of style definitions	236
Managing Browser Incompatibility	237
Coping with incompatibility	237
Making Internet Explorer-specific code	238
Using a conditional comment with CSS	240
Checking the Internet Explorer version.....	242
Using a CSS reset.....	243

Chapter 6: CSS Special Effects 245

Image Effects	245
Transparency	245
Reflections	247
Text Effects.....	249
Text stroke.....	249
Text-shadow	251
Transformations and Transitions.....	252
Transformations	253
Three-dimensional transformations.....	254
Transition animation	257
Animations.....	259

Part III: Building Layouts with CSS 263

Chapter 1: Fun with the Fabulous Float 265

Avoiding Old-School Layout Pitfalls.....	265
Problems with frames	265
Problems with tables.....	266
Problems with huge images.....	267
Problems with Flash	267
Introducing the Floating Layout Mechanism	268
Using float with images	269
Adding the float property	270
Using Float with Block-Level Elements	271
Floating a paragraph.....	271
Adjusting the width	273
Setting the next margin.....	275
Using Float to Style Forms.....	276
Using float to beautify the form	279
Adjusting the fieldset width.....	282
Using the clear attribute to control page layout	283

Chapter 2: Building Floating Page Layouts	285
Creating a Basic Two-Column Design	285
Designing the page.....	285
Building the HTML.....	287
Using temporary background colors	288
Setting up the floating columns	290
Tuning up the borders	291
Advantages of a fluid layout	292
Using semantic tags.....	292
Building a Three-Column Design	295
Styling the three-column page	296
Problems with the floating layout.....	298
Specifying a min-height	299
Using height and overflow	300
Building a Fixed-Width Layout.....	302
Setting up the HTML.....	303
Fixing the width with CSS	303
Building a Centered Fixed-Width Layout.....	305
Making a surrogate body with an all div.....	306
How the jello layout works	307
Limitations of the jello layout	308
 Chapter 3: Styling Lists and Menus.	 309
Revisiting List Styles	309
Defining navigation as a list of links	310
Turning links into buttons	310
Building horizontal lists	313
Creating Dynamic Lists	314
Building a nested list	315
Hiding the inner lists	317
Getting the inner lists to appear on cue	318
Building a Basic Menu System	321
Building a vertical menu with CSS.....	322
Building a horizontal menu	324
 Chapter 4: Using Alternative Positioning.	 327
Working with Absolute Positioning.....	327
Setting up the HTML.....	327
Adding position guidelines	328
Making absolute positioning work.....	330
Managing z-index.....	331
Handling depth.....	331
Working with z-index.....	332
Building a Page Layout with Absolute Positioning.....	332
Overview of absolute layout.....	333

Writing the HTML	334
Adding the CSS	335
Creating a More Flexible Layout.....	336
Designing with percentages.....	337
Building the layout.....	339
Exploring Other Types of Positioning.....	340
Creating a fixed menu system	340
Setting up the HTML.....	341
Setting the CSS values	342
Flexible Box Layout Model	344
Creating a flexible box layout.....	345
Viewing a flexible box layout.....	346
... And now for a little reality	348
Determining Your Layout Scheme.....	351

Part IV: Client-Side Programming with JavaScript 353

Chapter 1: Getting Started with JavaScript. 355

Working in JavaScript	355
Choosing a JavaScript editor.....	356
Picking your test browser.....	356
Writing Your First JavaScript Program.....	357
Embedding your JavaScript code	358
Creating comments.....	358
Using the alert() method for output.....	358
Adding the semicolon.....	359
Introducing Variables.....	359
Creating a variable for data storage	360
Asking the user for information	361
Responding to the user	361
Using Concatenation to Build Better Greetings.....	362
Comparing literals and variables	363
Including spaces in your concatenated phrases	364
Understanding the String Object	364
Introducing object-based programming (and cows).....	364
Investigating the length of a string	365
Using string methods to manipulate text	366
Understanding Variable Types	368
Adding numbers.....	369
Adding the user's numbers	370
The trouble with dynamic data.....	370
The pesky plus sign	371
Changing Variables to the Desired Type.....	372
Using variable conversion tools	373
Fixing the addInput code	373

Chapter 2: Talking to the Page	375
Understanding the Document Object Model	375
Previewing the DOM	375
Getting the blues, JavaScript-style	377
Writing JavaScript code to change colors	378
Managing Button Events	379
Adding a function for more ... functionality	381
Making a more flexible function	382
Embedding quotes within quotes	384
Writing the changeColor function	384
Managing Text Input and Output	384
Introducing event-driven programming	385
Creating the HTML form	386
Using getElementById to get access to the page	387
Manipulating the text fields	388
Writing to the Document	388
Preparing the HTML framework	390
Writing the JavaScript	390
Finding your innerHTML	391
Working with Other Text Elements	391
Building the form	392
Writing the function	393
Understanding generated source	395
What if you're not in Chrome?	397
 Chapter 3: Decisions and Debugging	 399
Making Choices with If	399
Changing the greeting with if	401
The different flavors of if	402
Conditional operators	403
Nesting your if statements	403
Making decisions with switch	405
Managing Repetition with for Loops	406
Setting up the web page	407
Initializing the output	408
Creating the basic for loop	409
Introducing shortcut operators	410
Counting backwards	411
Counting by fives	412
Understanding the Zen of for loops	413
Building While Loops	413
Making a basic while loop	413
Getting your loops to behave	415
Managing more complex loops	416

Managing Errors with a Debugger	418
Debugging with the interactive console	420
Debugging strategies	422
Resolving syntax errors	422
Squashing logic bugs	424
Chapter 4: Functions, Arrays, and Objects.	429
Breaking Code into Functions	429
Thinking about structure	430
Building the antsFunction.html program	431
Passing Data to and from Functions	432
Examining the makeSong code	434
Looking at the chorus	434
Handling the verses	435
Managing Scope	437
Introducing local and global variables	437
Examining variable scope	437
Building a Basic Array	439
Accessing array data	440
Using arrays with for loops	441
Revisiting the ants song	442
Working with Two-Dimension Arrays	444
Setting up the arrays	446
Getting a city	447
Creating a main() function	448
Creating Your Own Objects	449
Building a basic object	449
Adding methods to an object	450
Building a reusable object	452
Using your shiny new objects	453
Introducing JSON	454
Storing data in JSON format	454
Building a more complex JSON structure	455
Chapter 5: Getting Valid Input	459
Getting Input from a Drop-Down List	459
Building the form	460
Reading the list box	461
Managing Multiple Selections	462
Coding a multiple selection select object	462
Writing the JavaScript code	463
Check, Please: Reading Check Boxes	465
Building the check box page	466
Responding to the check boxes	467
Working with Radio Buttons	468
Interpreting Radio Buttons	469

Working with Regular Expressions	470
Introducing regular expressions.....	473
Using characters in regular expressions	475
Marking the beginning and end of the line	476
Working with special characters	476
Conducting repetition operations	477
Working with pattern memory.....	478
New HTML5/CSS3 Tricks for Validation	479
Adding a pattern	481
Marking a field as required	481
Adding placeholder text.....	481
Chapter 6: Drawing on the Canvas	483
Canvas Basics.....	483
Setting up the canvas	484
How <canvas> works	485
Fill and Stroke Styles	486
Colors	486
Gradients.....	487
Patterns	489
Drawing Essential Shapes.....	491
Rectangle functions	491
Drawing text	492
Adding shadows.....	494
Working with Paths	496
Line-drawing options.....	498
Drawing arcs and circles.....	500
Drawing quadratic curves.....	502
Building a Bézier curve	503
Images	505
Drawing an image on the canvas	505
Drawing part of an image.....	507
Manipulating Pixels	508
Chapter 7: Animation with the Canvas	511
Transformations	511
Building a transformed image	512
A few thoughts about transformations	514
Animation	515
Overview of the animation loop.....	515
Setting up the constants	516
Initializing the animation	517
Animate the current frame	517
Moving an element.....	519
Bouncing off the walls	520
Reading the Keyboard.....	521
Managing basic keyboard input.....	522
Moving an image with the keyboard	523

***Part V: Server-Side Programming with PHP* 527**

Chapter 1: Getting Started on the Server 529

Introducing Server-Side Programming.....	529
Programming on the server.....	529
Serving your programs.....	530
Picking a language	531
Installing Your Web Server.....	532
Inspecting phpinfo().....	533
Building HTML with PHP	536
Coding with Quotation Marks	539
Working with Variables PHP-Style.....	540
Concatenation	541
Interpolating variables into text	542
Building HTML Output.....	543
Using double quote interpolation.....	543
Generating output with heredocs.....	544
Switching from PHP to HTML.....	546

Chapter 2: PHP and HTML Forms 549

Exploring the Relationship between PHP and HTML.....	549
Embedding PHP inside HTML	550
Viewing the results	551
Sending Data to a PHP Program.....	552
Creating a form for PHP processing	552
Receiving data in PHP	555
Choosing the Method of Your Madness	556
Using get to send data.....	557
Using the post method to transmit form data	559
Getting data from the form	560
Retrieving Data from Other Form Elements.....	563
Building a form with complex elements	563
Responding to a complex form	565

Chapter 3: Using Control Structures 569

Introducing Conditions (Again).....	569
Building the Classic if Statement	570
Rolling dice the PHP way	571
Checking your six.....	571
Understanding comparison operators.....	574
Taking the middle road	574
Building a program that makes its own form	576
Making a switch	578
Looping with for	581
Looping with while	584

Chapter 4: Working with Arrays	587
Using One-Dimensional Arrays	587
Creating an array	587
Filling an array	588
Viewing the elements of an array	588
Preloading an array	589
Using Loops with Arrays	590
Simplifying loops with foreach	591
Arrays and HTML	593
Introducing Associative Arrays	594
Using foreach with associative arrays	595
Introducing Multidimensional Arrays	597
We're going on a trip	597
Looking up the distance	599
Breaking a String into an Array	600
Creating arrays with explode	601
Creating arrays with preg_split	602
Chapter 5: Using Functions and Session Variables	605
Creating Your Own Functions	605
Rolling dice the old-fashioned way	606
Improving code with functions	607
Managing variable scope	610
Returning data from functions	610
Managing Persistence with Session Variables	611
Understanding session variables	613
Adding session variables to your code	614
Chapter 6: Working with Files and Directories	617
Text File Manipulation	617
Writing text to files	618
Writing a basic text file	620
Reading from the file	625
Using Delimited Data	626
Storing data in a CSV file	627
Viewing CSV data directly	629
Reading the CSV data in PHP	630
Working with File and Directory Functions	633
opendir()	633
readdir()	634
chdir()	634
Generating the list of file links	635

Chapter 7: Exceptions and Objects	639
Object-Oriented Programming in PHP	639
Building a basic object	640
Using your brand-new class	642
Protecting your data with access modifiers	644
Using access modifiers	645
You've Got Your Momma's Eyes: Inheritance	647
Building a critter based on another critter	648
How to inherit the wind (and anything else)	649
Catching Exceptions	650
Introducing exception handling	650
Knowing when to trap for exceptions	652

Part VI: Managing Data with MySQL **653**

Chapter 1: Getting Started with Data	655
Examining the Basic Structure of Data	655
Determining the fields in a record	657
Introducing SQL data types	657
Specifying the length of a record	658
Defining a primary key	659
Defining the table structure	659
Introducing MySQL	660
Why use MySQL?	661
Understanding the three-tier architecture	662
Practicing with MySQL	662
Setting Up phpMyAdmin	663
Changing the root password	665
Adding a user	670
Using phpMyAdmin on a remote server	672
Implementing a Database with phpMyAdmin	674
Chapter 2: Managing Data with MySQL	679
Writing SQL Code by Hand	679
Understanding SQL syntax rules	680
Examining the buildContact.sql script	680
Dropping a table	681
Creating a table	681
Adding records to the table	682
Viewing the sample data	683
Running a Script with phpMyAdmin	683
Using AUTO_INCREMENT for Primary Keys	686

Selecting Data from Your Tables	688
Selecting only a few fields	689
Selecting a subset of records	690
Searching with partial information	692
Searching for the ending value of a field	693
Searching for any text in a field	693
Searching with regular expressions	694
Sorting your responses	695
Editing Records	696
Updating a record	696
Deleting a record	697
Exporting Your Data and Structure	697
Exporting SQL code	700
Creating XML data	702
Chapter 3: Normalizing Your Data	705
Recognizing Problems with Single-Table Data	705
The identity crisis	706
The listed powers	706
Repetition and reliability	708
Fields with changeable data	709
Deletion problems	709
Introducing Entity-Relationship Diagrams	709
Using MySQL workbench to draw ER diagrams	709
Creating a table definition in Workbench	710
Introducing Normalization	713
First normal form	714
Second normal form	715
Third normal form	716
Identifying Relationships in Your Data	717
Chapter 4: Putting Data Together with Joins	719
Calculating Virtual Fields	719
Introducing SQL functions	720
Knowing when to calculate virtual fields	721
Calculating Date Values	721
Using DATEDIFF to determine age	722
Adding a calculation to get years	723
Converting the days integer into a date	723
Using YEAR() and MONTH() to get readable values	724
Concatenating to make one field	725
Creating a View	726
Using an Inner Join to Combine Tables	728
Building a Cartesian join and an inner join	729
Enforcing one-to-many relationships	731

Counting the advantages of inner joins	732
Building a view to encapsulate the join	733
Managing Many-to-Many Joins.....	733
Understanding link tables	735
Using link tables to make many-to-many joins.....	736

Chapter 5: Connecting PHP to a MySQL Database. 741

PHP and MySQL: A Perfect (but Geeky) Romance	741
Understanding data connections.....	744
Introducing PDO.....	745
Building a connection.....	745
Retrieving data from the database	747
Using HTML tables for output	748
Allowing User Interaction	751
Building an HTML search form	753
Responding to the search request.....	753

Part VII: Integrating the Client and Server with AJAX..... 759

Chapter 1: AJAX Essentials 761

AJAX Spelled Out	762
A is for asynchronous	763
J is for JavaScript	763
A is for ... and?	763
And X is for ... data.....	763
Making a Basic AJAX Connection	764
Building the HTML form.....	766
Creating an XMLHttpRequest object.....	767
Opening a connection to the server	768
Sending the request and parameters	769
Checking the status	769
All Together Now — Making the Connection Asynchronous	771
Setting up the program	772
Building the getAJAX() function	772
Reading the response.....	773

Chapter 2: Improving JavaScript and AJAX with jQuery 775

Introducing jQuery	776
Installing jQuery.....	777
Importing jQuery from Google	777
Your First jQuery App.....	778
Setting up the page	779
Meet the jQuery node object.....	780
Creating an Initialization Function	781

Using \$(document).ready()	782
Alternatives to document.ready	783
Investigating the jQuery Object	783
Changing the style of an element	783
Selecting jQuery objects	785
Modifying the style	785
Adding Events to Objects	786
Adding a hover event	787
Changing classes on the fly	788
Making an AJAX Request with jQuery	790
Including a text file with AJAX	791
Building a poor man's CMS with AJAX	791
Chapter 3: Animating jQuery	795
Playing Hide and Seek	795
Getting transition support	797
Writing the HTML and CSS foundation	799
Initializing the page	800
Hiding and showing the content	800
Toggling visibility	801
Sliding an element	801
Fading an element in and out	802
Changing Position with jQuery	802
Creating the framework	804
Setting up the events	805
Building the move() function with chaining	806
Building time-based animation with animate()	806
Move a little bit: Relative motion	808
Modifying Elements on the Fly	808
Building the basic page	813
Initializing the code	813
Adding text	813
Attack of the clones	814
It's a wrap	815
Alternating styles	816
Resetting the page	816
More fun with selectors and filters	817
Chapter 4: Using the jQuery User Interface Toolkit	819
What the jQuery User Interface Brings to the Table	819
It's a theme park	820
Using the themeRoller to get an overview of jQuery	820
Wanna drag? Making components draggable	823
Downloading the library	824
Writing the program	826
Resizing on a Theme	827
Examining the HTML and standard CSS	829

Importing the files.....	829
Making a resizable element	830
Adding themes to your elements.....	830
Adding an icon	833
Dragging, Dropping, and Calling Back.....	834
Building the basic page.....	836
Initializing the page.....	836
Handling the drop	838
Beauty school dropout events	838
Cloning the elements.....	839
Chapter 5: Improving Usability with jQuery	841
Multi-Element Designs	841
Playing the accordion widget.....	842
Building a tabbed interface	845
Using tabs with AJAX.....	848
Improving Usability	849
Playing the dating game.....	851
Picking numbers with the slider	852
Selectable elements	854
Building a sortable list	855
Creating a custom dialog box.....	856
Chapter 6: Working with AJAX Data	859
Sending Requests AJAX Style.....	859
Sending the data	859
Building a Multipass Application.....	863
Setting up the HTML framework.....	864
Loading the select element.....	865
Writing the loadList.php program.....	866
Responding to selections.....	867
Writing the showHero.php script	868
Working with XML Data	870
Review of XML.....	871
Manipulating XML with jQuery	872
Creating the HTML.....	873
Retrieving the data	874
Processing the results.....	874
Printing the pet name.....	875
Working with JSON Data.....	876
Knowing JSON's pros	876
Reading JSON data with jQuery	877
Managing the framework	878
Retrieving the JSON data	879
Processing the results.....	879

Chapter 7: Going Mobile	883
Thinking in Mobile.....	883
Building a Responsive Site	885
Specifying a media type	885
Adding a qualifier.....	885
Making Your Page Responsive.....	888
Building the wide layout	891
Adding the narrow CSS	892
Using jQuery Mobile to Build Mobile Interfaces.....	894
Building a basic jQuery mobile page.....	894
Working with collapsible content.....	897
Building a multi-page document	900
Going from Site to App.....	905
Adding an icon to your program.....	906
Removing the Safari toolbar	906
Storing your program offline	907

Part VIII: Moving from Pages to Sites **909**

Chapter 1: Managing Your Servers	911
Understanding Clients and Servers.....	911
Parts of a client-side development system.....	912
Parts of a server-side system	913
Creating Your Own Server with XAMPP	914
Running XAMPP	915
Testing your XAMPP configuration	916
Adding your own files.....	916
Setting the security level	917
Compromising between functionality and security	919
Choosing a Web Host	920
Finding a hosting service	920
Connecting to a hosting service.....	922
Managing a Remote Site.....	922
Using web-based file tools	922
Understanding file permissions	924
Using FTP to manage your site.....	925
Using an FTP client	926
Naming Your Site.....	928
Understanding domain names	928
Registering a domain name	929
Managing Data Remotely	931
Creating your database.....	931
Finding the MySQL server name.....	932

Chapter 2: Planning Your Sites	933
Creating a Multipage Web Site	933
Planning a Larger Site	934
Understanding the Client.....	934
Ensuring that the client's expectations are clear	935
Delineating the tasks	936
Understanding the Audience	937
Determining whom you want to reach.....	937
Finding out the user's technical expertise	938
Building a Site Plan.....	939
Creating a site overview.....	940
Building the site diagram.....	941
Creating Page Templates	943
Sketching the page design	943
Building the HTML template framework.....	945
Creating page styles	947
Building a data framework.....	949
Fleshing Out the Project	950
Making the site live.....	950
Contemplating efficiency	951
Chapter 3: Introducing Content Management Systems.....	953
Overview of Content Management Systems.....	954
Previewing Common CMSs.....	955
Moodle.....	955
WordPress	956
Drupal	957
Building a CMS site with WebsiteBaker	958
Installing your CMS.....	958
Getting an overview of WebsiteBaker	962
Adding your content.....	962
Using the WYSIWYG editor.....	963
Changing the template	968
Adding additional templates	969
Building Custom Themes.....	971
Adding new functionality.....	970
Starting with a prebuilt template.....	971
Changing the info.php file.....	973
Modifying index.php.....	974
Modifying the CSS files	975
Packaging your template	976
Chapter 4: Editing Graphics	977
Using a Graphic Editor	977
Choosing an Editor	978

Introducing Gimp.....	979
Creating an image	980
Painting tools.....	980
Selection tools.....	982
Modification tools.....	984
Managing tool options.....	984
Utilities	985
Understanding Layers.....	986
Introducing Filters	988
Solving Common Web Graphics Problems.....	989
Changing a color	989
Building a banner graphic.....	990
Building a tiled background	992

Chapter 5: Taking Control of Content995

Building a “Poor Man’s CMS” with Your Own Code.....	995
Using Server Side Includes (SSIs)	995
Using AJAX and jQuery for client-side inclusion	998
Building a page with PHP includes	1000
Creating Your Own Data-Based CMS.....	1001
Using a database to manage content	1001
Writing a PHP page to read from the table.....	1004
Allowing user-generated content.....	1007
Adding a new block	1011
Improving the dbCMS design	1013

Index..... 1015