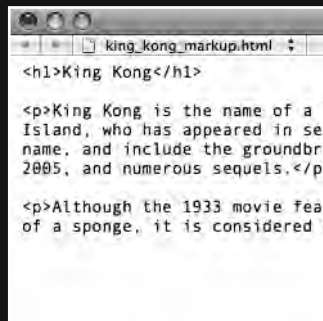


CHAPTER 2

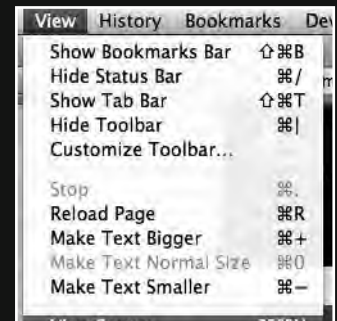
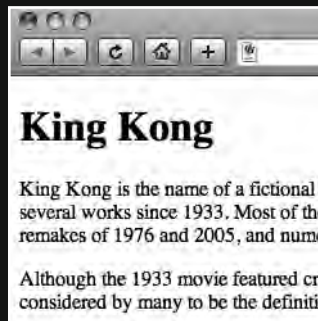
BUILDING BASIC WEB PAGES



```
<h1>King Kong</h1>

<p>King Kong is the name of a
Island, who has appeared in se
name, and include the groundbr
2005, and numerous sequels.</p>

<p>Although the 1933 movie fea
of a sponge, it is considered l
```



In this chapter, we get started with the hands-on aspects of our Web Standardistas' journey. First, we introduce you to HTML, the language that underpins the World Wide Web. We then build a few basic web pages to give you an understanding of how they are constructed and to introduce you to some fundamental concepts. This will form the cornerstone of the rest of the book, so we spend some time with you working through this systematically to ensure you have grasped a solid understanding of the principles of HTML and XHTML before moving on.

Once you've built your first web page, we show you how to use your browser's View Source command to learn from other designers' web sites by looking at their underlying source code. We also introduce the concept of HTML elements, looking at two key sections of your document: the `<head>` and the `<body>`, explaining what their purpose is and how they work.

Along the way, we cover nesting tags, commenting your markup, and the importance of using a well-written `<title>` tag.

HTML: Tags in action

You briefly met HTML in the last chapter when we covered the evolution of the Web. In this chapter, you get some hands-on use of it. In a nutshell, HTML provides basic formatting for words and images—our **content**—and we use it to build web pages and to give documents structure. HTML is used to describe the different elements that a web page can consist of, for example, headings (`<h1>`, `<h2>` . . .), paragraphs (`<p>`), and lists (``, `` . . .). The way we describe these elements and add this structure is through the use of what are known as **tags**.

What are tags?

HTML pages are in essence plain text files with the addition of tags that provide information on how your document is structured. The tags are distinguished from the rest of the content by being enclosed in angle brackets like this: `<...>`. Everything between the tags is intended for display within the browser; the different tags provide information on how the document should be interpreted and displayed.

A word of warning: we're working with XHTML 1.0 Strict throughout this book, and, as its name suggests, its rules are strict. All tags *must* be written in lowercase (i.e., `<h1>` is right, `<H1>` isn't). In addition to this key rule, there are some other important rules that we'll introduce to you when the time is right.

Let's look at an example. In Figure 2-1, we have opened a simple plain text file in our text editor and saved it as an HTML file.

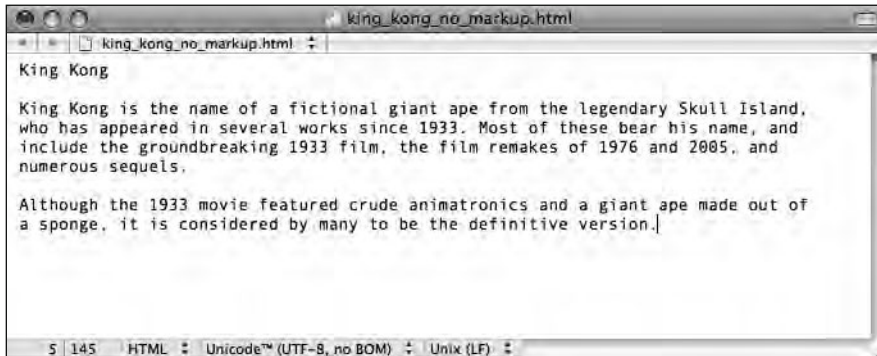


Figure 2-1. Our document, without HTML tags, as it appears in our plain text editor

Figure 2-2 shows the same page as it displays in our browser. Without any HTML tags to inform the browser how to structure the content of the page, the browser has no way of knowing how to display the content, and so simply displays it as a long line of text, wrapping to the width of the browser window. Note that all of the formatting and line breaks in our plain text example are ignored by the browser.

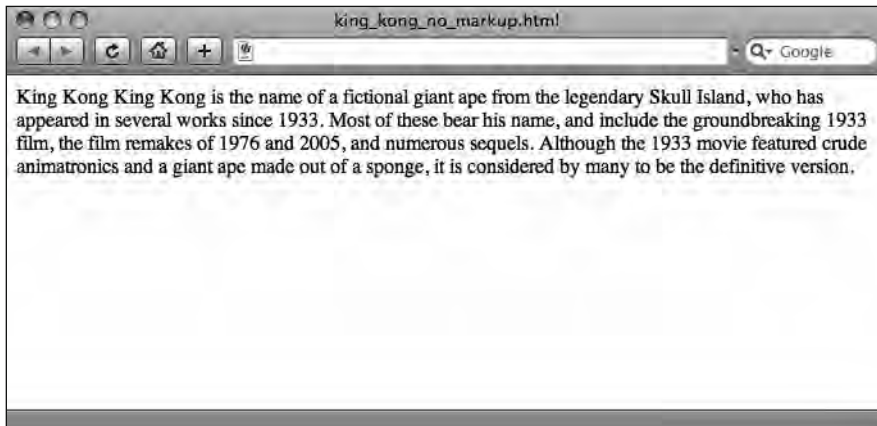
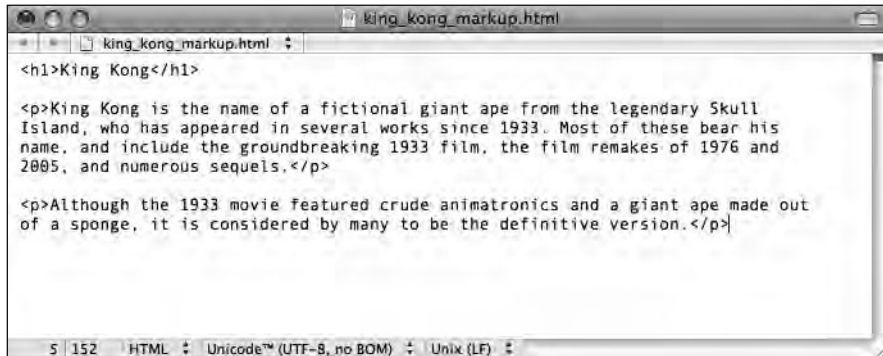


Figure 2-2. Our document, without tags, as it displays in a browser

Compare this to the example in Figure 2-3, where we have added some basic tags to the HTML document. We've marked up the first line—King Kong—to be a header using `<h1>` tags and marked up the two paragraphs that follow using `<p>` tags to divide the text into two distinct paragraphs.


 A screenshot of a plain text editor window titled 'king_kong_markup.html'. The editor shows the following HTML code:


```
<h1>King Kong</h1>

<p>King Kong is the name of a fictional giant ape from the legendary Skull
Island, who has appeared in several works since 1933. Most of these bear his
name, and include the groundbreaking 1933 film, the film remakes of 1976 and
2005, and numerous sequels.</p>

<p>Although the 1933 movie featured crude animatronics and a giant ape made out
of a sponge, it is considered by many to be the definitive version.</p>
```

 The status bar at the bottom indicates '5 | 152 | HTML | Unicode™ (UTF-8, no BOM) | Unix (LF)'.

Figure 2-3. Our document, with HTML tags added, in our plain text editor

Figure 2-4 shows how this marked-up version of our document displays in the browser. Note the difference from Figure 2-2: the browser now displays some basic formatting and gives a sense of the underlying document structure.

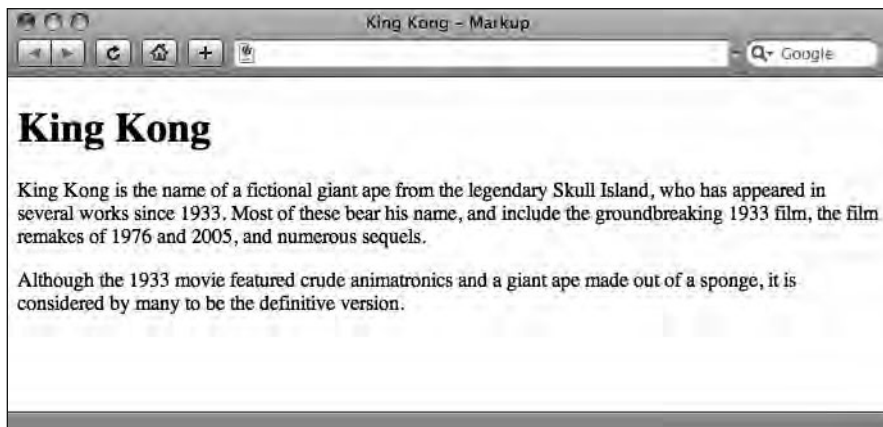


Figure 2-4. Our document, with basic tags added, as it displays in a browser

As you can see, the tags are not directly visible in the browser; instead, the tags inform the browser how the text is structured and how to display the content.

Tags come in pairs, usually

In XHTML, apart from a few exceptions, once opened, all tags need to be closed. Most tags come in pairs consisting of an **opening tag** and a **closing tag** (sometimes called a **start tag** and an **end tag**, respectively). For example, as shown in Figure 2-3 earlier, a paragraph opens with a `<p>` tag and closes with a `</p>` tag (the slash, `/`, after the opening angle bracket distinguishes the closing tag from the opening tag).

In the earliest days of the World Wide Web, as HTML evolved, browsers were quite forgiving. It was possible to write invalid markup and for the browser to “do its best” to second-guess what you were trying to achieve. However, as web standards become increasingly embraced, writing valid, well-formed markup from the outset makes it easier for browsers to display your pages consistently. Well-formed markup can help to reduce the amount of time spent trying to debug a page that doesn’t display the way you intended.

Following the simple rules we introduce in this chapter can save you a significant amount of time—which you would otherwise spend debugging and fixing problems—in the long run. More importantly, it will result in your writing markup that is the envy of your peers. It’s not difficult to write well-formed markup; it’s simply a matter of diligent attention to detail, something every aspiring Web Standardista should strive for.

The following example shows how to correctly close tags. The `<p>` opening the paragraph has been closed with a `</p>`:

```
<p>I am a paragraph by a well-trained Web Standardista.  
I have been closed correctly.</p>
```

In the following example, however, the paragraph has not been closed properly:

```
<p>I am a paragraph by a lazy programmer. I haven't been closed.
```

Remember, once opened, all tags should be closed.

Getting into the practice of writing well-formed markup—in particular closing all the tags that you open—can help resolve display issues down the line. It is not a difficult habit to get into, indeed, not getting into the habit can result in more difficulty down the line. Follow a few simple rules, and you’re well and truly on your way.

It’s an element, my dear Watson

The opening tag, the closing tag, and the content within these tags are known collectively as an **HTML element**. Figure 2-5 illustrates the structure of an element. As you can see, the opening tag, the tag’s content, and the closing tag *combined* make up the element.

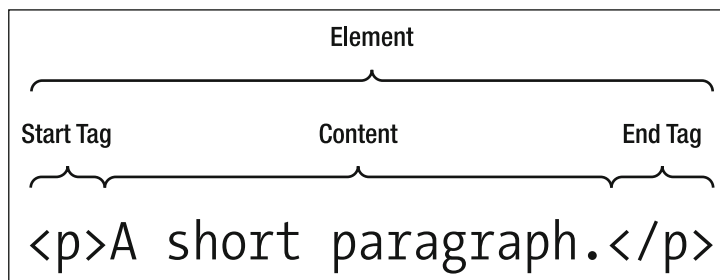


Figure 2-5. The structure of an HTML element

We will regularly refer to opening tags, closing tags, and elements throughout this chapter, so it’s worth ensuring that you have an understanding of how an element is constructed.

Your first web page: Hello World!

In this section, you’ll build your first web page. Nothing too complicated, but your first web page nonetheless. Although it might at first glance seem a little simplistic, this page will highlight a number of key principles that form the majority of the learning for this chapter.

When learning any new programming language, it’s tradition to write a short program to display the words “Hello World!” Our first web page will be no different. It will introduce you to the basic concepts of HTML and provide stage one of your journey through web standards.

Let’s get started on your first web page. Launch your text editor and create a new document. Save it as `hello_world.html`. This will be your first web page.

Before we write the web page, let’s take a look at how we’ve named it. Naming files is important and can cause issues down the line if done incorrectly, so it’s worth spending some time on file-naming conventions now. Let’s look at the file name we just specified in a little more detail.

The `.html` part is important: it is a suffix, referred to as an **extension**, that tells the browser the document is a web page. (`.html` stands for HyperText Markup Language, but then by now you know that, don’t you?) An alternative suffix, heralding from the days when certain software could only handle three-letter extensions, is `.htm`. Some people prefer to name their files using `.html`, others using `.htm`. Regardless of which extension you prefer to use, it is best to be consistent. In this book, we’re using `.html`.

The `_` (underscore) is also important; we’ve used it instead of a space, as you can see in Figure 2-6. Spaces, along with a few other characters, are not allowed in URLs. (As you probably know, a **URL**, short for Universal Resource Locator, is the address you type into the browser’s address bar when you want to load a specific page on the Web.) Since spaces are not allowed, the browser will convert it to `%20`, as shown in Figure 2-7.

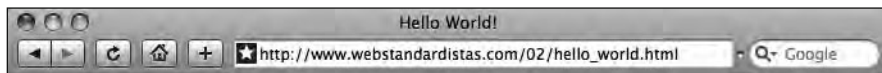


Figure 2-6. Our “Hello World!” page’s URL with an underscore replacing the space is easy to read.



Figure 2-7. Our “Hello World!” page’s URL with the space in the file name converted to `%20` is confusing.

So the file name that makes sense to you and is easily read as a URL:

```
hello world.html
```

is converted by the browser to the following:

```
hello%20world.html
```

Although the page would still load, the URL will have an unsightly %20 in it. Not only is this rather ugly and hard to read, but it is also a major Web Standardista faux pas. Although we have used underscores in our file names, an alternative is to use hyphens to replace spaces as in the following example:

```
hello-world.html
```

Whether you use underscores or hyphens is largely a matter of taste. Some people prefer the look of underscores, some the look of hyphens. It is worth mentioning, however, that Google tends to interpret hyphens as spaces, so using hyphens might help Google to index your page. Regardless, here is our First Golden Rule:

Don't use spaces in your file names. Ever.

Finally, upper- and lowercase can be interpreted differently on different systems. Windows systems are case insensitive whereas Linux systems are not. The following two file names are seen as different documents by a computer running Linux:

```
hello_world.html  
Hello_World.html
```

As such, they'll be treated as different web pages by your browser. It's time to introduce another golden rule so you'll never need to worry about this again. Here is our Second Golden Rule:

Use only lowercase letters when naming files.

Follow this rule, and you won't run the risk of your page not being found because of the difference in upper- and lowercase letters in the file name.

The bottom line is to be consistent, because it will save problems later. We recommend naming all files in lowercase and using _ (underscores) or - (hyphens) instead of spaces throughout. Follow our rules, and you'll save yourself a lot of trouble down the line.

To mark up a web page, you just type

After that brief, but important, digression on file-naming conventions, it's back to your first web page. In your new document, type the following:

```
<html>  
<head>  
<title>Hello World!</title>
```

```

</head>
<body>
<p>Hello World!</p>
</body>
</html>

```

Save the file and open it within your browser. If you are using a Mac, you can do this by locating your file and dragging it into an open browser window, or by opening your browser, selecting File ► Open File, browsing to the `hello_world.html` file you just saved, and clicking Open. The page will then load into a browser window. (If you are using Windows or Linux, the procedure is similar, but the commands will vary slightly.) The web page is a simple one, but a web page nonetheless. It should look something like Figure 2-8.



Figure 2-8. Our “Hello World!” web page that we prepared earlier, displayed in a browser

Congratulations! You’ve just built your first web page. It might not seem like much, but it marks the first step on your journey to becoming a Web Standardista.

The markup makes the web page

Let’s look at the preceding markup and break it down a little; doing so will give you an understanding of a web page’s basic construction and a solid foundation on which to build as we progress through the chapter. In our “Hello World!” example, we used the following tags: `<html>`, `<head>`, `<title>`, `<body>`, and `<p>`.

These tags provide the basic underlying structure:

- The `<html>` tag tells the browser we’re opening a new HTML document; it primes the browser, telling it, “Hey, get ready to receive some HTML goodness!”
- The `<head>` tag tells the browser we’re providing some information *about* the page; this is where we put information like the title of the page.
- The `<title>` tag tells the browser the title of the page (you’ll see it at the top of the browser in your “Hello World!” page).

- The `<body>` tag tells the browser we're starting information that we want to display on the page itself.
- The `<p>` tag surrounds our first—admittedly short—paragraph.

That's it. Five tags might not seem like much, but as you can see, they're enough to create a web page, and so begin your journey toward Web Standardista happiness.

Learning from others: How to view source

Now that you have a general understanding of how to use tags to organize and format content, we can start to explore how other designers use tags to structure *their* web pages. The best way to do this is to load up a web page and use your browser's View Source feature to see the original behind-the-scenes code that underlies the page.

At first glance, the source code might look unintelligible, but given time, experience, and persistence, looking at the markup of different web pages will teach you a huge amount about web design and, equally importantly, will enable you to troubleshoot your own web pages when things go wrong.

The wonderful thing about the Web, and what makes it easy to learn from others' Web pages, is the ability to use the View Source menu command in your browser to view the underlying source code for almost any web page. On Safari, the browser we're using, you'll find this command under the View menu (View ► View Source) as you can see in Figure 2-9. Most browsers have an equivalent command.

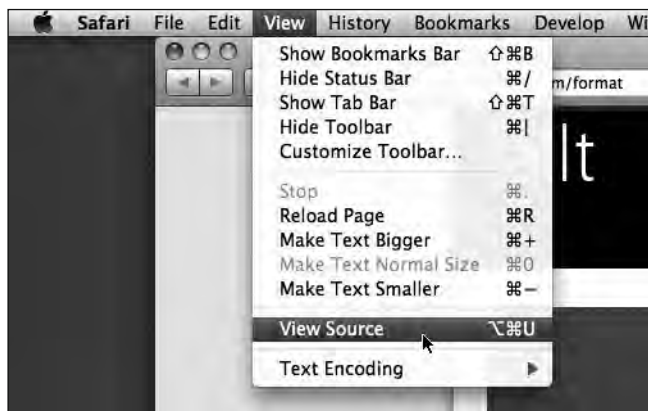


Figure 2-9. Using Safari's View Source feature

Use View Source to take a look at the underlying source code of two web pages we've provided, the URLs of which follow. By now, you should know enough about the structure of HTML to work out what's going on. As you work through the chapters, we encourage you to look behind the scenes of the homework web pages and try and work out what's going on. Experiment by copying and pasting some of our markup into your web pages and seeing what happens. The best way to learn is by doing.

We’ve uploaded our two example web pages for you here:

www.webstandardistas.com/02/king_kong_junior.html
www.webstandardistas.com/02/king_kong_senior.html

*It is important to stress that although you can look at the source of other designers’ web pages and figure out how they are put together and learn from this, it is **not** permitted to copy and paste other peoples’ markup and use it as your own, unless the designer has specifically stated that you are allowed to do so.*

Every page has a <head> and a <body>

HTML pages are broken into two key elements: the head and body elements, as illustrated in Figure 2-10. Both handle different types of information about the web page itself, and both are essential.

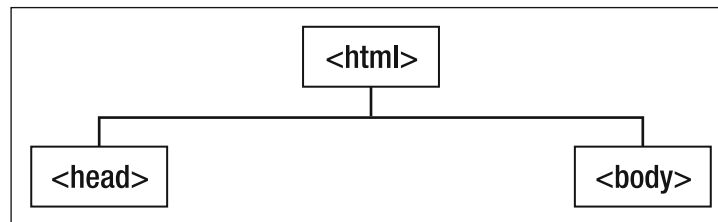


Figure 2-10. The head and body elements are both a part of every HTML page

The head element—everything contained within and including the opening <head> and the closing </head> tag pair—handles information about the document and other data that is not considered part of the document’s actual content (i.e., what appears within the viewer’s browser window). This includes the page’s title, its meta tags (tags which, among other things, can be used to provide information about the page to search engines), its style sheets, and any scripts. In short, anything about or affecting the content of the page itself. Information in the head element isn’t seen by the everyday user, only information in the body element is.

The body element—everything within and including the opening <body> and the closing </body> tag pair—is where everything the user will see within the browser is contained. Any text, links, or images you want to be displayed within the web browser reside within it. As we progress through the next few chapters, we’ll provide you with additional useful tags to ensure your <body> is well fed.

The importance of using the title element

When you open a book—this one for example—you expect there to be some chapter headings and for the information within the chapters to be broken down into sections

with titles and subtitles. Let's face it, it would be difficult to find this book on your bookshelf if it didn't have a title. Web pages are no different.

The head element must contain a `title` element, which is typically displayed in the browser's title bar and, if you are using Windows, in the task bar when the browser is minimized. The `title` is also the name saved when you bookmark a page or save it to your favorites.

If you don't include a `title`, the browser will usually display the name of the file (e.g., `index.html`) in the title bar. This is confusing for the user and certainly embarrassing for any budding Web Standardista.

Spending a little extra time and inserting a proper page title not only improves usability, but also helps improve search engine rankings. Try to use meaningful keywords within your page title; the upcoming examples demonstrate the importance of a meaningful title.

In the following example, the title is clear and informative. It gives us an understanding of the contents of the page.

```
<head>
  <title>Burger Flipper | ACME Widgets </title>
</head>
```

The following example is less useful. Which company's product page are we looking at? Is this the Burger Flipper page or another page altogether?

```
<head>
  <title>Product Page</title>
</head>
```

How many times have you looked back through your browsing history to try and retrace your steps? Some sites are easy to find, their titles clearly displayed in your history; others are impossible to find—all you can see is Untitled Document, `index.html`, or Product Page. Providing a well-considered title helps resolve this confusion. In the preceding example, `Burger Flipper | ACME Widgets` is clear and unambiguous.

In the next example, we have put the product names, for example “Burger Flipper,” *before* the company name, “ACME Widgets.” This means that when you have the entire ACME Widgets product catalog open in tabs in your browser, you can see the product names of each page, as illustrated in Figure 2-11. Putting the company name first and product name second might result in tabs reading as follows: `ACME Widge . . .`, `ACME Widge . . .`, `ACME Widge . . .`, again and again and again. Less useful and quite frustrating.

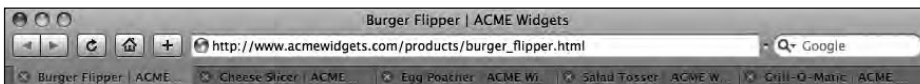


Figure 2-11. The ACME Widgets product catalog, open in tabs

Defining your document type

As we mentioned in Chapter 1, there are several types of HTML and XHTML: HTML 4.01 Strict, HTML 4.01 Transitional, and XHTML 1.0 Strict, to mention but a few.

In order to process your markup correctly, so it displays the way you intend it to, a web browser needs to know which set of rules to use when interpreting your document. For example, if you’ve created an HTML 4.01 Transitional document, the web browser needs to use different rules than if you’ve created an XHTML 1.0 Strict document.

But how does the browser know which document type you are using?

It all starts with a DOCTYPE

As we mentioned in the previous chapter, the different versions of HTML and XHTML are defined by the W3C, but they are also defined in something known as a **Document Type Definition**, or **DTD** for short. The DTD is written not for humans, but for tools that process (X)HTML documents. As a consequence, if it looks a little daunting at first sight, it is because it is intended for machines, not people.

The DTD is often referred to as a **DOCTYPE**. The DOCTYPE informs the browser which flavor of HTML or XHTML you’re using. Throughout this book, we will be using XHTML 1.0 Strict. The XHTML 1.0 Strict DOCTYPE looks like this:

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

The DOCTYPE is an additional, but important, part of your web page that tells the browser how to display the page and what language has been used to mark it up. The DOCTYPE needs to come before your opening `<html>` tag, as you’ll see later in the section “Hello World!: DOCTYPE edition.” Failure to include a DOCTYPE at the start of your web page triggers what’s known as **Quirks Mode**, implying to the browser that your web page was written using old-fashioned, invalid, and quirky markup. “What is Quirks Mode?” we hear you ask. Let’s take a look.

A short Quirks Mode interlude

Quirks Mode was conceived at a time when browsers were starting to pay proper attention to web standards. Millions of pages out there were created when the implementation of CSS was less than stellar. The authors of these pages had built them to work in older browsers, writing CSS that matched those browser’s implementations.

If the new browsers were completely standards compliant, a lot of these old pages—written in the bad old days—would render as broken. As a consequence, browser vendors looked for a solution that would allow old web pages to continue to be rendered using the old rules, and the new pages built with web standards to be rendered using the new, compliant rules. The trigger that would tell the browser to use the compliant, strict mode instead of Quirks Mode was the inclusion of a DOCTYPE. Adding the DOCTYPE to the top of your page was an indication that you knew what you were doing, and that you wanted your pages to be interpreted using the new, strict rules.

As an aspiring Web Standardista, you know how important it is to be standards-compliant. In the next section, we show you where to place the DOCTYPE so that the browsers your markup meets know what to expect, but first a couple of other important additions.

It's all in a namespace

A second attribute that is required when creating valid XHTML Strict web pages is the inclusion of what's known as an **XML namespace declaration**. Essentially, we replace the opening `<html>` tag from our simple "Hello World!" web page with the following:

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

The `xmlns="http://www.w3.org/1999/xhtml"` attribute is required when writing web pages in XHTML. Added to the opening `<html>` tag, it ensures that your page validates.

Just one more thing

As Columbo would say, "Just one more thing . . ."

The last thing we need to add to ensure our pages validate is a character encoding. Like the DOCTYPE and namespace, you don't need to know exactly how this works; you just need to know that you must include it for your web pages to validate. You simply add the following immediately after your opening `<head>` tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

The curious among you might want to skip ahead to Chapter 13, where we introduce the wonders of character encoding fully, although this isn't required reading at this point.

You don't have to memorize all this

Don't worry, you won't have to learn or memorize the exact syntax of the DOCTYPE, namespace declaration, or character encoding. For now, you can simply use a template file we've provided for you to ensure your pages validate and render correctly.

To save you a lot of painstaking typing, and to ensure against mistyping, we've uploaded an XHTML file with the correct DOCTYPE, namespace declaration, and character encoding to the book's companion web site, where you can simply download it, and copy and paste it. Use [View Source](#) to find it here:

```
www.webstandardistas.com/02/template.html.
```

Apart from helping your browser determine how to interpret your document, the DOCTYPE also enables you to use tools like The W3C Markup Validation Service (<http://validator.w3.org/>) to check the syntax of your documents, helping you to ensure that your code is valid.

Hello World!: DOCTYPE edition

Earlier in this chapter, we created a very simple “Hello World!” web page to give you an idea of a web page’s basic structure. We’ll now develop this by adding the DOCTYPE, namespace declaration, and character encoding as described previously. Let’s have a look at our new and improved “Hello World!” web page:

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

Although viewing this new and improved version of your web page in a browser will appear to make little difference, trust us, behind the scenes the browser is being informed that you’re building standards-compliant pages. The DOCTYPE, namespace declaration, and character encoding are critical parts of every web page you build, and you should include them from now on.

Tags have structure too: Nested elements

HTML elements can be nested, a feature which you have already seen in action in our simple “Hello World!” web page. Figure 2-12 shows the basic structure of this page as a diagram known as a **document tree**.

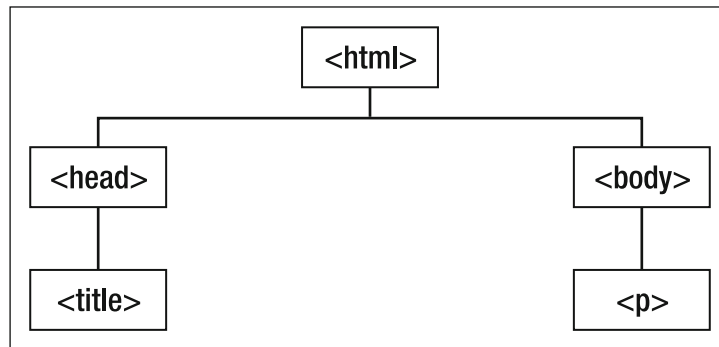


Figure 2-12. A simple diagram of the document tree

Nesting can simply be described as placing one element inside another. In our “Hello World!” web page, illustrated in Figure 2-12, both the head and body elements are placed, or nested, within the html element. The title element is nested within the head element, and the p element is nested within the body.

An easy way to grasp this concept is to think of your web page as an inverted tree. The tree’s trunk is the html element; from this trunk have grown two branches, the head and body elements. From each of these branches further elements grow.

An important rule to remember when writing valid XHTML markup is that elements must be nested properly. Think of a Russian nested doll, which is really a set of dolls, each nested within another: tags are no different.

To nest elements in the right order, you need to make sure that you close your tags in the reverse order that they were opened. An easy way to remember this is to use the mnemonic “First In, Last Out.” Following is an example of nesting tags in action; in this case, we’ve added another layer of structure to a paragraph, using (strong emphasis, by default displayed in bold in graphical browsers) and (emphasis, by default displayed in italics). Figure 2-13 shows this markup in action.

```
<p><strong><em>I am a paragraph, I am not only emphasized,  
I am also strongly emphasized.</em></strong></p>
```

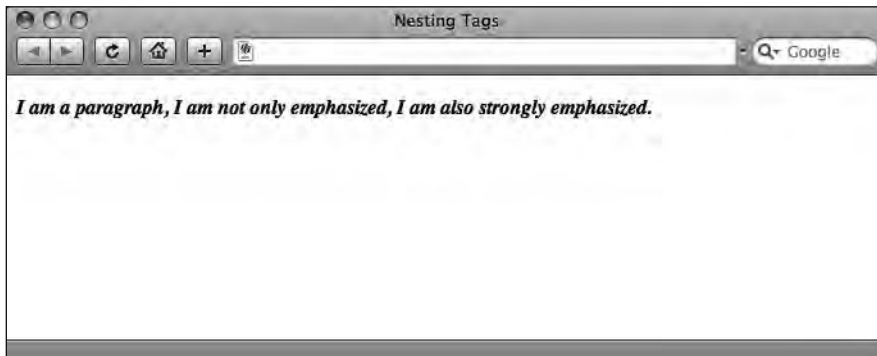


Figure 2-13. The nested markup displayed in a browser

Although web browsers are forgiving and would render the preceding HTML page the same regardless of whether you nested your tags correctly, as mentioned before, making sure your pages are valid is important in the long run. When building more complex pages—and adding CSS to the mix—incorrectly nested tags can lead to inconsistent display of your web page across browsers.

In the following two examples, we show you the right way to nest elements using the First In, Last Out approach. These tags are nested correctly:

```
<p><strong><em>I have been nested properly by a Web  
Standardista.</em></strong></p>
```

In the following example, the tags are not nested properly:

```
<p><em><strong>I have not been nested properly, I've probably
been written by the same lazy programmer who didn't close his
tags properly a few pages back.</em></strong></p>
```

Making your markup easier to follow

By now, we've introduced quite a bit of complexity. We're using a variety of tags, we're nesting tags in the right order, we're using DOCTYPEs, and we never, ever forget to use a well-written title element. We're building web pages and using *View Source* to look at how other designers use XHTML. In short, there's a lot for us to remember as we move forward to the next chapter.

The good news is that we can use both the structure of our code—breaking it over different lines, using tabs and white space—and leave hidden comments within it to make our job that little bit easier.

In this section, we introduce the importance of both commenting your markup—leaving hidden comments within it—and putting some thought into how you format it within your chosen plain text editor—breaking it over different lines, using tabs and white space. This can make your life a great deal easier, especially when you return to a project after some time has elapsed.

Commenting your markup

Not everything we write in HTML displays within the browser. We've already introduced you to the head element, which is largely hidden from view within the main browser window. (X)HTML allows for the inclusion of hidden comments in both the head and body elements that can only be read when using *View Source* or looking at the `.html` file in question. Indeed, if you've been using your browser's *View Source* command to look at other web pages, you might have seen some examples of comments in use.

Comments open with a `<!--` and close with a `-->`; anything included between these markers is not displayed in the browser. This can be very useful for a number of reasons: a comment could serve as a note to remind you why you structured a document a particular way, a note to indicate when you changed the document, a note to a friend working on the same web page, or a means of hiding parts of the document itself. This latter use can be particularly useful when testing, enabling you to switch the display of elements on or off.

The easiest way to show you how comments work is to show you some in action. In the following example, we've included a comment between the first two paragraphs. We've also *commented out* the third paragraph, showing you how you can also use comments to hide parts of your markup.

As you can see in the screenshot in Figure 2-14, our comment between the first two paragraphs is hidden from the viewer, as is the third paragraph.

```
<h1>Comments in Action</h1>
<p>I am paragraph number one, I display in the browser.
The Heading 1 above also displays in the browser.</p>
<!-- I am a comment, I don't display in the browser. -->
<p>I am paragraph number two, I also display in the browser.</p>
<!-- <p>I am paragraph number three, I'm hidden.</p> -->
```

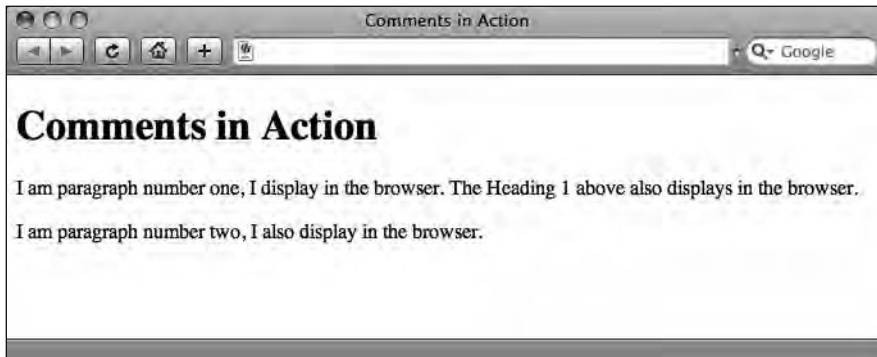


Figure 2-14. Our page with comments, hidden within the browser

Comments aren't restricted to single lines; they can also run over multiple lines. This can be useful when you want to hide a section of a document while you're testing your web pages during development. In our next example, we've hidden a number of lines using comments.

As you can see in Figure 2-15, only the first paragraph is displayed in the browser; the remaining paragraphs are hidden from the viewer.

```
<h1>Getting Clever With Comments</h1>
<p>I am paragraph number one, you met me in the last example.</p>
<!--
<p>I am paragraph number two, I won't display in the browser.</p>
<p>I am paragraph number three, I'm also hidden.</p>
-->
```

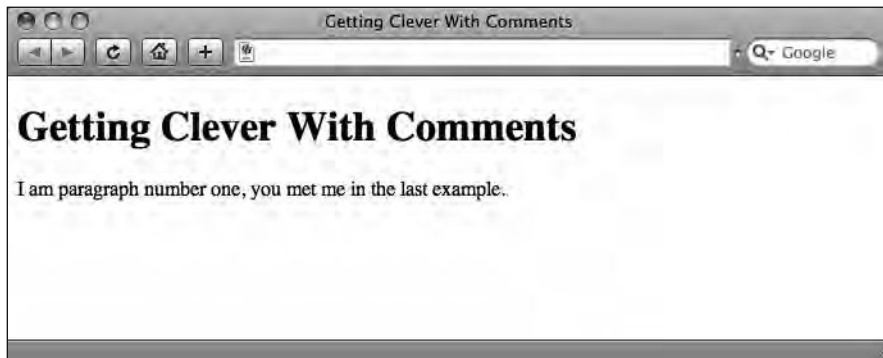


Figure 2-15. The previous example as seen in a browser—only the first, uncommented paragraph is visible.

Although browsers hide everything between comments, your comments are still delivered to the user's browser along with the rest of the page's markup. As you now know, anyone can read these comments using *View Source*. Comments therefore aren't the best place to hide your secrets. The following, for example, isn't advisable:

```
<h1>Pinocchio - My Darkest Secrets</h1>
<p>I am a child made of wood. My darkest secret was revealed in
Shrek 2.</p>
<!-- I wear ladies underwear. -->
<p>I'm a good little boy.</p>
```

White space

In addition to using comments to assist in the process of writing markup, it's also worth considering the use of white space to enable you to visually indicate your document's structure within your plain text editor. As long as you've written your XHTML in a plain text editor with formatting switched off, you can use line breaks, tabs, and spaces (commonly referred to as **white space**) to separate the sections of your document to make your markup more readable. When viewed in the browser, these white spaces are ignored.

Take a look at the following two examples of our "Hello World!" page. Both display identically in the browser. In the first, we've used no line breaks or tabs:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Hello World!</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
```

In the following example, we've introduced line breaks and tabs. As you can see, this makes the markup much easier to read.

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Hello World!</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

You can see both of the preceding examples live here:

```
www.webstandardistas.com/02/no_whitespace.html
www.webstandardistas.com/02/whitespace.html
```

But what if you want to preserve line breaks, tabs, and extra spaces for a web page about poetry or one that is displaying examples of code within the page, for example? Have no fear, we will introduce a tag that specializes in just that in Chapter 4 when we show you how to present preformatted code examples on your web page.

Summary

So what have we covered? In this chapter, we got our hands dirty and started to build some web pages. We focused on the fundamental aspects of a web page's construction and highlighted some dos and don'ts. We also introduced the tricky topic of DOCTYPEs and how they inform the browser behind the scenes that you're a budding Web Standardista. Lastly, we looked at ways of making your code easier to read by using comments and white space.

In the next chapter, we start to add some good, old-fashioned, upper-class POSH markup.

Homework: Create your first space-monkey-themed XHTML page

This chapter's homework is to create a complete XHTML page using the plain text editor you downloaded and took for a test drive at the end of the last chapter. We've supplied some text for you; all you need to do is add some basic markup to this text to create a simple web page along the lines of the ones we've covered in the chapter.

1. Get started

Let's get started. Open your text editor and create a new file. The page you'll be making is about Albert I, the first-ever monkey astronaut. Following the advice on naming files, we suggest you save your new blank file as follows:

```
albert.html
```

We suggest you keep all your homework files organized in one place—for now, use a single folder.

We'll cover organizing files in folders later, as this will have an effect on how your different files relate to each other. At this point please keep all of your files in one location. We suggest you create a folder called homework where you save this and future homework files.

2. A basic web page

By now you should know that the basic, minimal structure of an XHTML document looks like the example displayed here:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title></title>
  </head>
  <body>
  </body>
</html>
```

At this point, you have two choices: you can either painstakingly type the preceding markup into your blank `albert.html` document, or you can go to the Web Standardistas web site and cheat a little by using copy and paste. You can find the preceding markup here:

```
www.webstandardistas.com/02/template.html
```

Using your browser's View Source feature, copy and paste the markup into your blank `albert.html` document and save it.

3. Find out about Albert I

Once you've got the basic markup in place, the next thing to do is to get some content for the page. Following the tradition of the best cooking shows, we've prepared this earlier and supplied you with some text about Albert I at the following location:

```
www.webstandardistas.com/02/albert.txt
```

4. Add a title

You know the importance of adding a descriptive title to your web pages. We've left this part up to you. After reading the text, add an appropriate title to your page between the `<title>` tags, and save your page.

5. Mark up the content

Copy and paste the text provided into the body of your `albert.html` page.

As our text currently stands, it has no markup or structure. Your task is to add some structure to the page using `<h1>` and `<p>` tags, a process similar to the one covered in Figures 2-3 and 2-4 of this chapter. With such a simple text file, this should be a relatively straightforward process. Add your markup and save the page.

6. Add a comment

Once you've completed the previous stage, add a comment on the page congratulating yourself on creating your very first space-monkey page using XHTML. Do this by leaving a comment in the markup as follows:

```
<!-- Add a comment like this. Well done! -->
```

7. Test your page

Now there's just one thing left to do—test. Save the file and open the page in your web browser to check that everything looks the way you would expect. At this point, we hope all's well; if not, you'll need to check your code thoroughly for any mistakes that may have crept in.

If you do run into problems, some things to consider might be the following: Is the text copied into the body of the page? Have you closed all the tags you've opened? To help you with troubleshooting, we have created a similarly structured page about Cheeta, famous for his role in the *Tarzan* movies. You can refer to this, using your browser's View Source command to see how we've structured the web page, here:

```
www.webstandardistas.com/02/cheeta.html
```

Assuming all's well, put the kettle on and enjoy a cup of *Lapsang Souchong* as you prepare yourself for the next chapter.