

1

Creating Structured Documents

In this chapter, you meet the first technologies you need to learn in order to write web pages: HTML and XHTML. In fact, what you will really be learning is XHTML—although I will be explaining the differences between HTML and XHTML as we go along. (As I already mentioned, you can consider XHTML simply to be the latest version of HTML.)

The main goal of this chapter is to demonstrate how the primary role of XHTML is to describe the structure of your documents.

In this chapter, then, you:

- ☐ Learn the difference between tags, elements, and attributes
- ☐ See how a web page uses markup to describe how the page should be structured
- ☐ Meet the elements that allow you to mark up text such as headings and paragraphs
- ☐ Learn many other elements that can add additional presentation information and phrasing to your documents
- ☐ See how to add bulleted and numbered lists to documents
- ☐ Are introduced to some core concepts that distinguish different types of elements in XHTML

By the end of the chapter you will have a good idea of how to structure a page in XHTML and will have written your first web pages.

A Web of Structured Documents

Every day, you come across all kinds of printed documents—newspapers, train timetables, insurance forms. The Web is like a sea of documents all linked together; these documents bear a strong similarity to the documents that you meet in everyday life. So let's think for a moment about the structure of some of the documents we see around us, and how they compare to web pages.

Chapter 1: Creating Structured Documents

Every morning I used to read a newspaper. A newspaper is made up of several stories or articles (and probably a fair smattering of advertisements, too). Each story has a headline and then some paragraphs, perhaps a subheading, and then some more paragraphs; it may also include a picture or two.

I don't buy a daily paper anymore, as I tend to look at news online, but the structure of articles on news web sites is very similar to the structure of articles in newspapers. Each article is made up of headings, paragraphs of text, and the odd picture. The parallel is quite clear; the only real difference is that each story gets its own page on a web site, and that page is accessed by clicking on a headline or a brief summary either on the site's main home page or one of the home pages for a subsection of the site (such as the politics, sports, or entertainment sections).

Consider another example: Say I'm catching a train to see a friend, so I check the schedule to see what time the trains go that way. The main part of the schedule is a *table* telling me what times trains arrive and when they depart from different stations. In the same way that a lot of documents have headings and paragraphs, a lot of other documents use tables; from the stocks and shares pages in the financial supplement of your paper to the TV listings at the back, you come across tables of information every day—and these are often recreated on the Web.

Another kind of document you often come across is a form. For example, I have a form sitting on my desk (which I really must mail) from an insurance company. This form contains fields for me to write my name, address, and the amount of coverage I want, and boxes I have to check to indicate the number of rooms in the house and what type of lock I have on my front door. Indeed, there are lots of forms on the Web, from a simple search box that asks what you are looking for to the registration forms you are required to go through before you can place an online order for books or CDs.

As you can see, there are many parallels between the structure of printed documents you come across every day and pages you see on the Web. So you will hardly be surprised to learn that when it comes to writing web pages, your code tells the web browser the structure of the information you want to display—what text to put in a heading, or in a paragraph, or in a table, and so on—so that the browser can present it properly to the user.

In order to tell a web browser the structure of a document—how to make a heading, a paragraph, a table, and so on—you need to learn HTML and XHTML.

Introducing XHTML

XHTML, or Extensible Hypertext Markup Language, and its predecessor HTML, are the most widely used languages on the Web. As its name suggests, XHTML is a *markup language*, which may sound complicated, until you realize that you come across markup every day.

When creating a document in a word processor, you can add styles to the text to explain the document's structure. For example, you can distinguish headings from the main body of the text using a heading style (usually with a larger font). You can use the Enter (or Return) key to start a new paragraph. You can insert tables into your document to hold data, or create bulleted lists for a series of related points, and so on. While this does affect the presentation of the document, the key purpose of this kind of markup is to provide a structure that makes the document easier to understand.

When marking up documents for the Web, you are performing a very similar process, except you do it by adding things called *tags* to the text. With XHTML the key thing to remember is that you are adding

Chapter 1: Creating Structured Documents

the tags to indicate the *structure* of the document, which part of the document is a heading, which parts are paragraphs, what belongs in a table, and so on. Browsers such as Internet Explorer, Firefox, and Safari will use this markup to help present the text in a familiar fashion, similar to that of a word processor (headings are bigger than the main text, there is space between each paragraph, lists of bullet points have a circle in front of them). However the way these are presented is up to the browser; the XHTML specification does not say which font should be used or what size that font should be.

While earlier versions of HTML allowed you to control the presentation of a document—things like which typefaces and colors a document should use—XHTML markup is not supposed to be used to style the document; that is the job of CSS, which you meet in Chapter 7.

Let's have a look at a very simple web page. As I mentioned in the introduction, you don't need any special programs to write web pages—you can simply use a text editor such as Notepad on Windows or TextEdit on a Mac, and save your files with an .html file extension. You can download this example along with all the code for this book from the Wrox web site at www.wrox.com; the example is in the Chapter 1 folder and is called `ch01_eg01.html`.

```
<html>
  <head>
    <title>Popular Websites: Google</title>
  </head>
  <body>
    <h1>About Google</h1>
    <p>Google is best known for its search engine, although
      Google now offers a number of other services.</p>
    <p>Google's mission is to organize the world's
      information and make it universally accessible and
      useful.</p>
    <p>Its founders Larry Page and Sergey Brin started
      Google at Stanford University.</p>
  </body>
</html>
```

This may look a bit confusing at first, but it will all make sense soon. As you can see, there are several sets of angle brackets with words or letters between them, such as `<html>`, `<head>`, `</title>`, and `</body>`. These angle brackets and the words inside them are known as *tags*, and these are the markup we have been talking about. Figure 1-1 illustrates what this page would look like in a web browser.



Figure 1-1

Chapter 1: Creating Structured Documents

As you can see, this document contains the heading “About Google” and a paragraph of text to introduce the company. Note also that it says “Popular Websites: Google” in the top-left of the browser window; this is known as the *title* of the page.

To understand the markup in this first example, you need to look at what is written between the angle brackets and compare that with what you see in the figure, which is what you will do next.

Tags and Elements

If you look at the first and last lines of the code for the last example, you will see pairs of angle brackets containing the letters `<html>`. The two brackets and all of the characters between them are known as a *tag*, and there are lots of tags in the example. All the tags in this example come in pairs; there are *opening tags* and *closing tags*. The closing tag is always slightly different from the opening tag in that it has a forward slash after the first angled bracket `</html>`.

A pair of tags and the content these include are known as an *element*. In Figure 1-2, you can see the heading for the page of the last example.

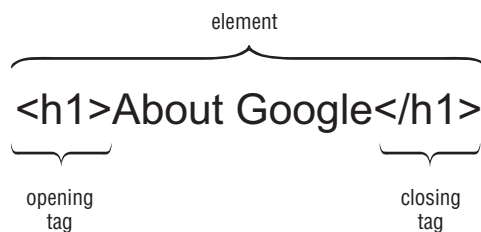


Figure 1-2

The opening tag says “This is the beginning of a heading” and the closing tag says “This is the end of a heading.” Like most of the tags in XHTML, the text inside the angled brackets explains the purpose of the tag—here `h1` indicates that it is a level 1 heading (or top-level heading). As you will see shortly, there are also tags for subheadings (`<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`). Without the markup, the words “About Google” in the middle of the tags would just be another bit of text; it would not be clear that they formed the heading.

Now look at the three paragraphs of text about the company; each one is held between an opening `<p>` tag and a closing `</p>` tag. And, you guessed it, the `p` stands for paragraph.

Because this basic concept is so important to understand, I think it bears repeating: *tags* are the angle brackets and the letters and numbers between them, whereas *elements* are tags and anything between the opening and closing tags.

Chapter 1: Creating Structured Documents

As you can see, the markup in this example actually describes what you will find between the tags, and the added meaning the tags give is describing the structure of the document. Between the opening `<p>` and closing `</p>` tags are paragraphs, and between the `<h1>` and `</h1>` tags is a heading. Indeed, the whole document is contained between opening `<html>` and closing `</html>` tags.

You will often find that terms from a family tree are used to describe the relationships between elements. For example, an element that contains another element is known as the *parent*, while the element that is between the parent element's opening and closing tags is called a *child* of that element. So, the `<title>` element is a child of the `<head>` element, the `<head>` element is the parent of the `<title>` element, and so on. Furthermore, the `<title>` element can be thought of as a grandchild of the `<html>` element.

XHTML tags should always be written in lowercase letters.

Separating Heads from Bodies

Whenever you write a web page in XHTML, the whole of the page is contained between the opening `<html>` and closing `</html>` tags, just as it was in the last example. Inside the `<html>` element, there are two main parts to the page:

- ❑ **The `<head>` element:** Often referred to as the head of the page, this contains information *about* the page (this is not the main content of the page). It is information such as a title and a description of the page, or keywords that search engines can use to index the page. It consists of the opening `<head>` tag, the closing `</head>` tag, and everything in between.
- ❑ **The `<body>` element:** Often referred to as the body of the page, this contains the information you actually see in the main browser window. It consists of the opening `<body>` tag, closing `</body>` tag, and everything in between.

Inside the `<head>` element of the first example page, you can see a `<title>` element:

```
<head>
  <title>Popular Websites: Google</title>
</head>
```

Between the opening and closing `title` tags are the words `Popular Websites: Google`, which is the title of this web page. If you remember Figure 1-1, which showed the screenshot of this page, I brought your attention to the words right at the top of the browser window. This is where browsers like Internet Explorer, Firefox, and Safari display the title of a document; it is also the name they use when you save a page in your favorites.

The real content of your page is held in the `<body>` element, which is what you want users to read, and is shown in the main browser window.

Chapter 1: Creating Structured Documents

The **head** element contains information about the document, which is not displayed within the main page itself. The **body** element holds the actual content of the page that is viewed in your browser.

You may have noticed that the tags in the example you have been looking at appear in a symmetrical order. If you want to have one element inside another, then both the element's opening and closing tags must be inside the containing element. For example, the following is allowed:

```
<p> This paragraph contains some <em>emphasized text.</em></p>
```

Whereas the following is wrong because the closing `</em>` tag is not inside the paragraph element:

```
<p> This paragraph contains some <em>emphasized text. </p></em>
```

In other words, if an element is to contain another element, it must wholly contain that element. This is referred to as *nesting* your elements correctly.

Attributes Tell Us About Elements

What really differentiates web documents from standard documents are the links (or hyperlinks) that take you from one web page to another. Let's take a look at an example of a link by adding one to the example you just looked at. Links are created using an `<a>` element (the *a* stands for anchor).

Here we will add a link from this page to Google in a new paragraph at the end of the document. There is just one new line in this example (code sample `ch01_eg02.html`) and that line is highlighted:

```
<html>
  <head>
    <title>Popular Websites: Google</title>
  </head>
  <body>
    <h1>About Google</h1>
    <p>Google is best known for its search engine, although Google now offers a
      number of other services.</p>
    <p>Google's mission is to organize the world's information and make it
      universally accessible and useful.</p>
    <p>Its founders Larry Page and Sergey Brin started Google at Stanford
      University.</p>
    <p><a href="http://www.Google.com/">Click here to visit Google's Web
      site.</a></p>
  </body>
</html>
```

Inside this new paragraph is the `<a>` element that creates the link. Between the opening `<a>` tag and the closing `</a>` tag is the text that you can click on, which says "Click here to visit Google's Web site."

Figure 1-3 shows you what this page looks like in a browser.

Chapter 1: Creating Structured Documents



Figure 1-3

If you look closely at the opening tag of the link, it carries something called an *attribute*. In this case it's the `href` attribute; this is followed by an equal sign, and then the URL for Google's web site in quotation marks. In this case, the `href` attribute is telling you where the link should take you. You look at links in greater detail in the next chapter, but for the moment this illustrates the purpose of attributes.

Attributes are used to say something about the element that carries them, and they always appear on the opening tag of the element that carries them. All attributes are made up of two parts: a *name* and a *value*:

- ❑ The *name* is the property of the element that you want to set. In this example, the `<a>` element carries an attribute whose name is `href`, which you can use to indicate where the link should take you.
- ❑ The *value* is what you want the value of the property to be. In this example, the value was the URL that the link should take you to, so the value of the `href` attribute is `http://www.Google.com`.

The value of the attribute should always be put in double quotation marks, and it is separated from the name by the equal sign. If you wanted the link to open in a new window, you could add a `target` attribute to the opening `<a>` tag as well, and give it a value of `_blank`:

```
<a href="http://www.Google.com" target="_blank">
```

This illustrates that elements can carry several attributes, although an element should never have two attributes of the same name.

All attributes are made up of two parts, the attribute's name and its value, separated by an equal sign. Values should be held within double quotation marks. All XHTML attribute names should be written in lowercase letters.

Chapter 1: Creating Structured Documents

The XML Declaration

Sometimes you will see something that is known as the XML Declaration at the beginning of an XHTML document. The XHTML language was actually written using another language called XML (Extensible Markup Language, which is used to create markup languages), and any XML document can begin with this optional XML declaration:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

If you include the XML declaration, it must be right at the beginning of the document; there must be nothing before it, not even a space. The `encoding` attribute indicates the encoding used in the document.

An encoding (short for character encoding) represents how a program or operating system stores characters that you might want to display. Because different languages have different characters, and indeed because some programs support more characters than others, there are several different encodings.

Document Type Declaration

As mentioned previously, XHTML is the successor to HTML—although you can just think of it as being the latest version. XHTML employs a stricter syntax than its predecessor HTML. For example, your element and attribute names in XHTML must all be written in lowercase (whereas earlier versions of HTML were not case-sensitive), every element that has some content must have a corresponding closing element, and some of the elements and attributes may be marked as deprecated—meaning that they were likely to be phased out in future versions of XHTML.

Each XHTML page should therefore begin with a `DOCTYPE` declaration to indicate to a browser (or any other program) the version of HTML or XHTML that is being used in that page.

While I have been talking about XHTML as one language, there were actually three versions or flavors of XHTML released—this was done to help existing web developers make the transition from HTML to XHTML:

- ❑ **Transitional XHTML 1.0**, which still allowed developers to use the deprecated markup from HTML 4.1 (which is likely to be phased out) but required the author to use the new stricter syntax.
- ❑ **Strict XHTML 1.0**, which was to signal the path forward for XHTML, without the deprecated stylistic markup and obeying the new stricter syntax.
- ❑ **Frameset XHTML 1.0**, which is used to create web pages that use a technology called *frames* (you meet frames in Chapter 6).

If by now you are feeling a little overwhelmed by all the different versions of HTML and XHTML, don't be! Throughout this book, you will be primarily learning Transitional XHTML 1.0. In the process, you will learn which elements and attributes have been marked as deprecated and what the alternatives for using these are. If you avoid the deprecated elements and attributes, you will automatically be writing Strict XHTML 1.0.

The `DOCTYPE` declaration goes before the opening `<html>` tag in a document, and after the optional XML Declaration if you have used it.

Chapter 1: Creating Structured Documents

If you are writing Transitional XHTML 1.0 (and include stylistic markup in your document), then your DOCTYPE declaration should look like this:

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

If you are writing Strict XHTML 1.0, your DOCTYPE declaration will look like this:

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

For frameset documents (discussed in Chapter 6), your DOCTYPE declaration would look like this:

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

A Strict XHTML document *must* contain the DOCTYPE declaration before the root element; however, you are not required to include the DOCTYPE declaration if you are creating a transitional or frameset document.

Having learned Transitional XHTML 1.0, you should be able to understand older versions of HTML and be safe in the knowledge that (unless specifically warned), your XHTML code will work in the majority of browsers used on the Web today.

Core Elements and Attributes

Now that you understand how the contents of a web page are marked up using elements that describe the structure of the document, the next step is to learn all the elements you can use to describe the structure of the various kinds of document you might wish to display on the Web. The rest of this chapter, and much of the next few chapters, will introduce you to all these elements.

As each element is introduced, I will be quite thorough about how it may be used, and which attributes it may take. This allows the book to act as a complete reference once you have learned how to write web pages. But, when you are first going through it, if you feel you understand what an element is used for, feel free to skip further ahead in that chapter if you want to—you can always come back later and read about it again.

Let's start by taking a closer look at the four main elements that form the basic structure of every document: `<html>`, `<head>`, `<title>`, and `<body>`. These four elements should appear in every XHTML document that you write, and you will see them referred to throughout this book as the *skeleton* of the document.

The `<html>` Element

The `<html>` element is the containing element for the whole XHTML document. After the optional XML declaration and required DOCTYPE declaration, each XHTML document should have an opening `<html>` tag and each document should end with a closing `</html>` tag.

Chapter 1: Creating Structured Documents

If you are writing Strict XHTML 1.0, the opening tag must also include something known as a *namespace identifier* (this indicates that the markup in the document belongs to the XHTML 1.0 namespace). Therefore the opening tag should look like this:

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

While it is not strictly required in Transitional XHTML documents, it is a good practice to use it on all XHTML documents.

Only two elements appear as direct children of an `<html>` element: `<head>` and `<body>` (although the `<head>` and `<body>` elements will usually contain many more elements).

The `<html>` element can also carry the following attributes, which you will meet in the “Attribute Groups” section later in this chapter:

```
id dir lang xml:lang
```

You may sometimes come across the use of the `version` attribute in HTML 4.1 and earlier to indicate which version of HTML the document uses, although it is usually left off. XHTML documents should use the DOCTYPE declaration along with the `xmlns` attribute instead to indicate which version of XHTML they use.

The `<head>` Element

The `<head>` element is just a container for all other header elements. It should be the first thing to appear after the opening `<html>` tag.

Each `<head>` element should contain a `<title>` element indicating the title of the document, although it may also contain any combination of the following elements, in any order:

- ☐ `<base>`, which you will meet in Chapter 2.
- ☐ `<object>`, which is designed to include images, JavaScript objects, Flash animations, MP3 files, QuickTime movies, and other components of a page. It is covered in Chapter 3.
- ☐ `<link>` to link to an external file, such as a style sheet or JavaScript file, as you will see in Chapter 7.
- ☐ `<style>` to include CSS rules inside the document; it is covered in Chapter 7.
- ☐ `<script>` for including script in the document, which you’ll see in Chapter 11.
- ☐ `<meta>`, which includes information about the document such as keywords and a description, which are particularly helpful for search applications; this is covered in Chapter 13.

The `profile` attribute is not actually in use yet, although it was included so it could be used in the future to specify a URL for something known as a *profile* that would describe the content of the document. The other attributes are covered in the “Attribute Groups” section later in this chapter.

Chapter 1: Creating Structured Documents

The opening `<head>` tag can carry the following attributes:

```
id dir lang xml:lang profile
```

The `<title>` Element

You should specify a title for every page that you write. It lives inside the `<title>` element (which, as you saw earlier in the chapter, is a child of the `<head>` element). It is used in several ways:

- ☐ At the very top of a browser window (as you saw in the first example and Figure 1-1)
- ☐ As the default name for a bookmark in browsers such as IE, Firefox, and Safari
- ☐ By search engines that use its content to help index pages

Therefore, it is important to use a title that really describes the content of your site. For example, the home page of our site should not just say “Home Page”; rather it should describe what your site is about. For example, rather than just saying Wrox Home Page, it is more helpful to write:

```
<title>Wrox: Books for programmers written by programmers</title>
```

The test for a good title is whether a visitor can tell what she will find on that page just by reading the title, without looking at the actual content of the page.

The `<title>` element should contain only the text for the title; it may not contain any other elements. The `<title>` element can carry the following attributes, which are covered in the “Attribute Groups” section later in the chapter:

```
id dir lang xml:lang
```

The `<body>` Element

The `<body>` element appears after the `<head>` element and contains the part of the web page that you actually see in the main browser window, which is sometimes referred to as *body content*. It may contain anything from a couple of paragraphs under a heading to more complicated layouts containing forms and tables, and is likely to constitute the majority of any XHTML document. Most of what you will be learning in this and the following four chapters will be written between the opening `<body>` tag and closing `</body>` tag.

The `<body>` element may carry all of the attributes from the *attribute groups* you are about to meet in the next section. If you are using Transitional XHTML or HTML 4.1, you can use any of the following deprecated attributes on the `<body>` element (which are covered in Appendix I):

```
background bgcolor alink link vlink text
```

There are also several browser specific attributes that you might see used on the `<body>` element; these also are covered in Appendix I:

```
language, topmargin, bottommargin, leftmargin, rightmargin, scroll,
bgproperties, marginheight, marginwidth
```

Chapter 1: Creating Structured Documents

Attribute Groups

As you have seen, attributes live on the opening tag of an element and provide extra information about the element that carries them. All attributes consist of a *name* and a *value*; the name reflects a property of the element the attribute is describing, and the value is a value for that property. For example, the `xml:lang` attribute describes the language used within that element; a value such as `EN-US` would indicate that the language used inside the element is U.S. English. Many of the elements in XHTML can carry some or all of the attributes you will meet in this section.

There are three groups of attributes that many of the XHTML elements can carry (as you have already seen, the `<html>`, `<head>`, `<title>`, and `<body>` elements share some of these attributes). Don't worry if they seem a little abstract at the moment; they will make more sense as you read on, but because they are used by so many elements I have grouped them here to avoid having to repeat them each time they come up. As I say, don't worry if they do not make complete sense at the moment, as long as you remember where you read this. You can keep referring back to them when you need to. The three attribute groups are:

- ❑ **Core attributes:** The `class`, `id`, and `title` attributes
- ❑ **Internationalization attributes:** The `dir`, `lang`, and `xml:lang` attributes
- ❑ **UI events:** Attributes associated with events `onclick`, `ondblclick`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onkeypress`, `onkeydown`, and `onkeyup` (these are covered in more detail in Chapter 11)

Together, the core attributes and the internationalization attributes are known as the *universal attributes*.

Core Attributes

The four core attributes that can be used on the majority of XHTML elements (although not all) are:

```
id title class style
```

Where these attributes occasionally have special meaning for an element that differs from the description given here, I revisit them; otherwise their use can generally be described as you see in the subsections that follow.

The *id* Attribute

The `id` attribute can be used to uniquely identify any element within a page. You might want to uniquely identify an element so that you can link to that specific part in the document, or to specify the element so that you can associate a CSS style or JavaScript to the content of that one element within the document.

The syntax for the `id` attribute is as follows (where *string* is your chosen value for the attribute):

```
id="string"
```

Chapter 1: Creating Structured Documents

For example, the `id` attribute could be used to distinguish between two paragraph elements, like so:

```
<p id="accounts">This paragraph explains the role of the accounts department.</p>
<p id="sales">This paragraph explains the role of the sales department.</p>
```

Note that there are some special rules for the value of the `id` attribute. It must:

- ❑ Begin with a letter (A–Z or a–z) and can then be followed by any number of letters, digits (0–9), hyphens, underscores, colons, and periods (you may not start the value with a digit, hyphen, underscore, colon, or period).
- ❑ Remain unique within that document; no two `id` attributes may have the same value within that XHTML document.

Before the `id` attribute was introduced, the `name` attribute served a similar purpose in HTML documents, but its use was deprecated in HTML 4.01, and now you should generally use the `id` attribute in XHTML documents. If you need to use the `name` attribute, it is available in Transitional XHTML, but not Strict XHTML (you might want to use the `name` attribute if you are dealing with older browsers that were written before the `id` attribute was introduced).

The class Attribute

Although the `id` attribute uniquely identifies a particular element, the `class` attribute is used to specify that an element belongs to a *class* of element. It is commonly used with CSS, so you will learn more about the use of the `class` attribute in Chapter 7, which introduces CSS. The syntax of the `class` attribute is as follows:

```
class="className"
```

The value of the attribute may also be a space-separated list of class names. For example:

```
class="className1 className2 className3"
```

The title Attribute

The `title` attribute gives a suggested title for the element. The syntax for the `title` attribute is as follows:

```
title="string"
```

The behavior of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip or while the element is loading.

Not every element that *can* carry a `title` attribute really needs one, so when we meet an element that particularly benefits from use of this attribute, I will show you the behavior it has when used with that element.

The style Attribute (deprecated)

The `style` attribute allows you to specify CSS rules within the element. You meet CSS in Chapter 7, but for the moment here is an example of how it might be used:

```
<p style="font-family:arial; color:#FF0000;">Some text</p>
```

Chapter 1: Creating Structured Documents

As a general rule, however, it is best to avoid the use of this attribute. This attribute is marked as deprecated in XHTML 1.0 (which means it will be removed from future versions of XHTML). If you want to use CSS rules to govern how an element appears, it is better to use a separate style sheet instead. You will see each of these techniques in Chapter 7, which introduces CSS.

Internationalization

There are three internationalization attributes that help users write pages for different languages and character sets, and they are available to most (although not all) XHTML elements (which is important in multi-lingual documents).

```
dir lang xml:lang
```

Even in current browsers, support for these attributes is still very patchy, and you are best off specifying a character set that will create text in the direction you require, although the `xml:lang` attribute could be used by other XML-aware applications.

Here is the web address of a helpful W3C document that describes internationalization issues in greater detail, although we will briefly look at each of these attributes next:

```
http://www.w3.org/TR/i18n-html-tech/
```

The internationalization attributes are sometimes referred to as the i18n attributes, an odd name that comes from the draft-ietf-html-i18n specification in which they were first defined.

The `dir` Attribute

The `dir` attribute allows you to indicate to the browser the direction in which the text should flow. When you want to indicate the directionality of a whole document (or the majority of the document), it should be used with the `<html>` element rather than the `<body>` element for two reasons: the `<html>` element has better support in browsers, and it will then apply to the header elements as well as those in the body. The `dir` attribute can also be used on elements within the body of the document if you want to change the direction of a small portion of the document.

The `dir` attribute can take one of two values, as you can see in the table that follows.

Value	Meaning
ltr	Left to right (the default value)
rtl	Right to left (for languages such as Hebrew or Arabic that are read right to left)

The `lang` Attribute

The `lang` attribute allows you to indicate the main language used in a document, but this attribute was kept in XHTML only for backwards compatibility with earlier versions of HTML. It has been replaced by the `xml:lang` attribute in new XHTML documents (which is covered in the next section). However, the

Chapter 1: Creating Structured Documents

XHTML recommendation suggests that you use both the `lang` and the `xml:lang` attributes on the `<html>` element in your XHTML 1.0 documents (to achieve maximum compatibility across different browsers).

The `lang` attribute was designed to offer language-specific display to users, although it has little effect in the main browsers. The real benefit of using the `lang` attribute is with search engines (which can tell the user which language the document is authored in), screen readers (which might need to pronounce different languages in different ways), and applications (which can alert users when they either do not support that language or it is a different language than their default language). When used with the `<html>` element it applies to the whole document, although it can be used on other elements, in which case it just applies to the content of those elements.

The values of the `lang` attribute are ISO-639 standard two-character language codes. If you want to specify a dialect of the language, you can follow the language code with a dash and a subcode name. The table that follows offers some examples.

Value	Meaning
ar	Arabic
en	English
en-us	U. S. English
zh	Chinese

A list of language codes for most of the main languages in use today can be found in Appendix G.

The `xml:lang` Attribute

The `xml:lang` attribute is the XHTML replacement for the `lang` attribute. It is an attribute that is available in all languages that are written in XML (you may remember earlier in the chapter that I mentioned that XHTML was written in XML), which is why it is prefixed by the characters `xml:`. The value of the `xml:lang` attribute should be an ISO-639 country code like those listed in the previous section; a full list appears in Appendix G.

While it has no effect in the main browsers, other XML-aware applications and search engines may use this information, and it is good practice to include the `xml:lang` attribute in your documents. When used with the `<html>` element, it applies to the whole document, although it can be used on other elements, in which case it just applies to the content of those elements.

UI Events

The UI event attributes allow you to associate an *event*, such as a key press or the mouse being moved over an element, with a script (a portion of programming code that runs when the event occurs). For example, when someone moves a mouse over the content of a certain element you might use a script to make it change color.

Chapter 1: Creating Structured Documents

You will meet the UI events in more detail in Chapter 14, although their names indicate quite clearly what event they are associated with; for example, `onclick` fires when a user clicks on that element's content, `onmousemove` fires when a mouse moves, and `onmouseout` fires when a user moves the mouse out of the content of a particular element.

There are ten events, known collectively as *common events*:

```
onclick, ondoubleclick, onmousedown, onmouseup, onmouseover, onmousemove,
onmouseout, onkeypress, onkeydown, onkeyup
```

The `<body>` and `<frameset>` elements also have the following events for when a page opens or is closed:

```
onload onunload
```

Finally, there are a number of events that work with forms only (which are mentioned in Chapter 5 and again in Chapter 11):

```
onfocus, onblur, onsubmit, onreset, onselect, onchange
```

Now that you have made your way through the preliminaries and learned about the elements that make up the skeleton of an XHTML document, it's time to get down to business marking up the text that will appear on your web pages.

Basic Text Formatting

You've seen the skeleton structure of an XHTML document and the core attributes, so it is now time to get back to looking at how you mark up text in order to describe its structure. Because almost every document you create will contain some form of text, the elements you are about to meet are the fundamental building blocks of most pages.

While going through this section it is important to remember that, while one browser might display each of these elements in a certain way, another browser could display very different results; the font sizes (and therefore the amount of space a section of text takes up) will change between browsers, as will the typefaces used. You will not really be learning how to control the appearance (typefaces, colors, and font sizes) of text until Chapter 7.

In this section, you learn how to use what are known as *basic text formatting elements*:

```
h1, h2, h3, h4, h5, h6
p, br, pre
```

If you want people to read what you have written, then structuring your text well is even more important on the Web than when writing for print. People have trouble reading long, wide paragraphs of text on web sites unless they are broken up well (as you will see in Chapter 9), so getting into good habits from the start of your web development career will help ensure that your pages get the attention they deserve.

Chapter 1: Creating Structured Documents

Before you get started on the elements that you will use to mark up your text, it helps to know how text is displayed by default (it is up to you to tell the browser if you want it to treat text differently).

White Space and Flow

Before you start to mark up your text, it is best to understand what XHTML does when it comes across spaces and how browsers treat long sentences and paragraphs of text.

You might think that if you put several consecutive spaces between two words, the spaces would appear between those words onscreen, but this is not the case; by default, only one space will be displayed. This is known as *white space collapsing*. Similarly, if you start a new line in your source document, or you have consecutive empty lines, these will be ignored and simply treated as one space, as will tab characters. For example look at the following paragraph (taken from `ch01_eg03.html` in the code samples):

```
<p>This paragraph shows how multiple spaces between words are  
treated as a single space. This is known as white space collapsing, and  
the big spaces between some of the words will not appear in the  
browser.
```

```
It also demonstrates how the browser will treat multiple carriage returns  
(new lines) as a single space, too.</p>
```

As you can see in Figure 1-4, the browser treats the multiple spaces and several carriage returns (where text appears on a new line) as if there were only one single space.

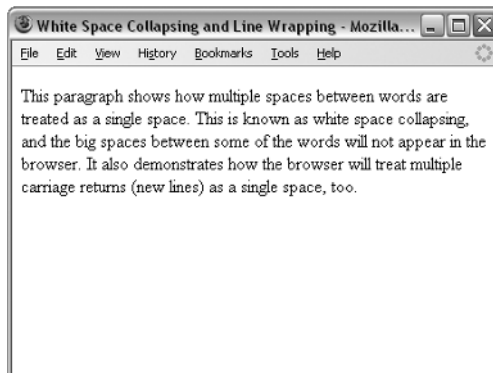


Figure 1-4

As Figure 1-4 also shows, when a browser displays text it will automatically *wrap* the text onto new lines when it runs out of space. If you look again at the code for this example, and look at where each new line starts, the results are different on the screen than they are in the code. You can try this out for yourself, as all of the examples are available with the download code for this book; just try resizing the browser window (making it smaller and larger) and notice how the text wraps at new places on the screen.

Chapter 1: Creating Structured Documents

This can be particularly helpful because it allows you to add spaces to your code that will not show up in the actual document, and these spaces can be used to indent your code, which makes it easier to read. The first two examples in this chapter demonstrated indented code, where child elements are indented from the left to distinguish them from their parent elements. This is something that I do throughout this book to make the code more readable. (If you want to preserve the spaces in a document, you need to use either the `<pre>` element, which you learn about later in the chapter or the ` ` entity reference, which you learn about in Appendix F.)

It is therefore extremely important that you learn how to use the elements in the rest of this chapter to break up and control the presentation of your text.

Creating Headings Using *h*n Elements

No matter what sort of document you are creating, most documents have headings in some form or other. Newspapers use headlines, a heading on a form tells you the purpose of the form, the title of a table of sports results tells you the league or division the teams play in, and so on.

In longer pieces of text, headings can also help structure a document. If you look at the table of contents for this book, you can see how different levels of headings have been arranged to add structure to the book, with subheadings under the main headings.

XHTML offers six levels of headings, which use the elements `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`. While browsers *can* display headings differently, they tend to display the `<h1>` element as the largest of the six and `<h6>` as the smallest, CSS can be used to override the size and style of any of the elements. The levels of heading would look something like those in Figure 1-5 (`ch01_eg04.html`).

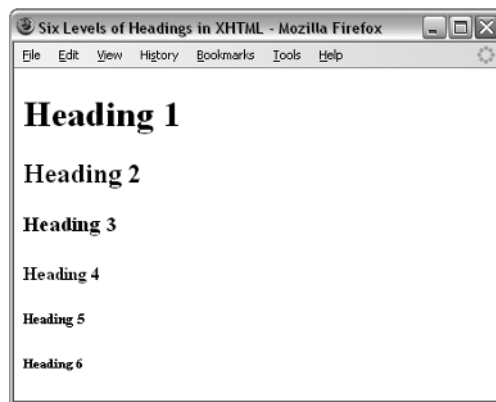


Figure 1-5

By default, most browsers display the contents of the `<h1>`, `<h2>`, and `<h3>` elements larger than the default size of text in the document. The content of the `<h4>` element would be the same size as the default text, and the content of the `<h5>` and `<h6>` elements would be smaller.

Chapter 1: Creating Structured Documents

Here is another example of how you might use headings to structure a document (ch01_eg05.html), where the <h2> elements are subheadings of the <h1> element (this actually models the structure of this section of the chapter):

```
<h1>Basic Text Formatting</h1>
<p> This section is going to address the way in which you mark up text.
Almost every document you create will contain some form of text, so this
will be a very important section. </p>
<h2>Whitespace and Flow</h2>
<p> Before you start to mark up your text, it is best to understand what
XHTML does when it comes across spaces and how browsers treat long sentences
and paragraphs of text.</p>
<h2>Creating Headings Using hn Elements</h2>
<p> No matter what sort of document you are creating, most documents have
headings in some form or other...</p>
```

Figure 1-6 shows how this will look.

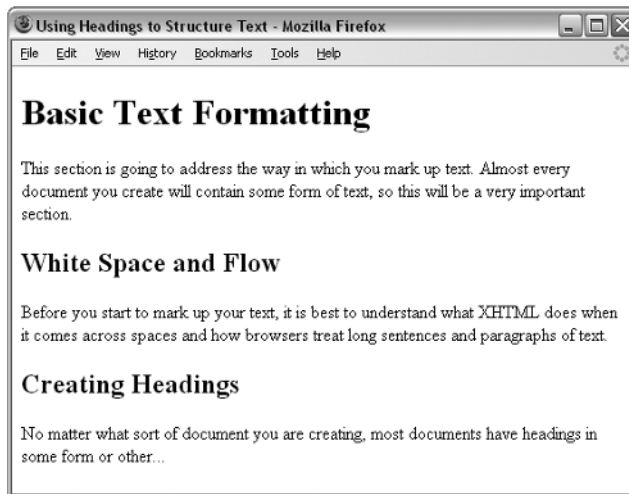


Figure 1-6

The six heading elements can all carry the universal attributes as well as a deprecated attribute called `align`:

```
align class id style title dir lang xml:lang
```

The align Attribute (deprecated)

The deprecated `align` attribute indicates whether the heading appears to the left, center, or right of the page (the default is the left). It can take the three values discussed in the table that follows.

Chapter 1: Creating Structured Documents

Value	Meaning
left	The heading is displayed to the left of the browser window (or other containing element if it is nested within another element). This is the default value if the align attribute is not used.
center	The heading is displayed in the center of the browser window (or other containing element if it is nested within another element).
right	The heading is displayed to the right of the browser window (or other containing element if it is nested within another element).

I mention the align attribute here because you are likely to see it used on various elements. It has been marked as deprecated because it does not help describe the structure of the document—rather it is used to affect the presentation of the page, which should now be done using CSS. Here is an example of using the deprecated align attribute (ch01_eg06.html):

```
<h1 align="left">Left-Aligned Heading</h1>
<p>This heading uses the align attribute with a value of left.</p>
<h1 align="center">Centered Heading</h1>
<p>This heading uses the align attribute with a value of center.</p>
<h1 align="right">Right-Aligned Heading</h1>
<p>This heading uses the align attribute with a value of right.</p>
```

Figure 1-7 shows the effect of the align attribute in a browser.

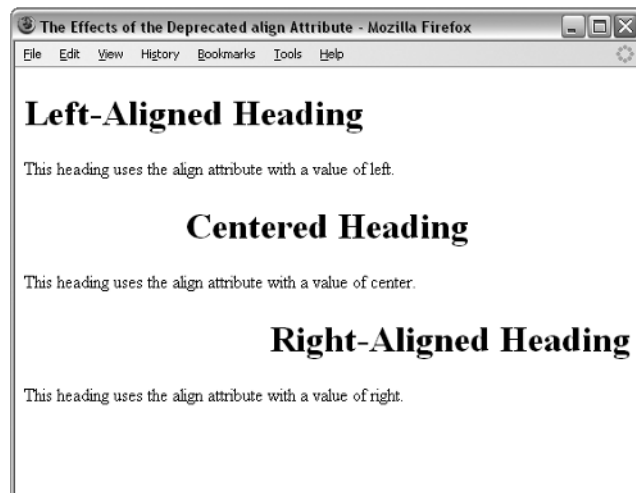


Figure 1-7

Chapter 1: Creating Structured Documents

The `align` attribute has been replaced with the `text-align` property in CSS and the ability to float block-level elements (as you will see in Chapter 7). The `align` attribute is covered in more detail in Appendix I.

Creating Paragraphs Using the `<p>` Element

The `<p>` element offers another way to structure your text. Each paragraph of text should go in between an opening `<p>` and closing `</p>` tag, as in this example (`ch01_eg07.html`):

```
<p>Here is a paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p>Here is a third paragraph of text.</p>
```

When a browser displays a paragraph, it usually inserts a new line before the next paragraph and adds a little bit of extra vertical space, as in Figure 1-8.

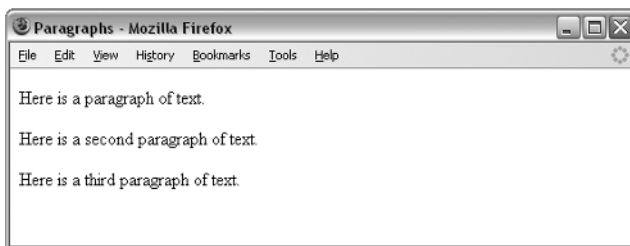


Figure 1-8

The `<p>` element can carry all of the universal attributes and the deprecated `align` attribute:

```
align class id style title dir lang xml:lang
```

Creating Line Breaks Using the `<br />` Element

Whenever you use the `<br />` element, anything following it starts on the next line. The `<br />` element is an example of an *empty element*, where you do not need opening *and* closing tags, because there is nothing to go in between them.

The `<br />` element has a space between the characters `br` and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, whereas if you miss the forward slash character and just use `<br>`, it is not valid XHTML.

Most browsers allow you to use multiple `<br />` elements to push text down several lines, and many designers use two line breaks between paragraphs of text rather than using the `<p>` element to structure text, as follows:

```
Paragraph one<br /><br />
Paragraph two<br /><br />
Paragraph three<br /><br />
```

Chapter 1: Creating Structured Documents

While this creates a similar effect to using the paragraph element, if you do not use the `<p>` element itself for each paragraph then the document is no longer describing where each paragraph starts and finishes. Furthermore, in Strict XHTML the `<br />` element can be used only within what are known as block-level elements. These are elements such as the `<p>` element—elements that tend to naturally act as though they have a line break before and after them. You learn more about block-level elements near the end of the chapter.

Avoid using `<br />` elements just to position text; such usage can produce unexpected results because the amount of space created when you do so depends upon the size of the font. Instead, you should use CSS, which you learn about in Chapter 7.

Here you can see an example of the `<br />` element in use within a paragraph (ch01_eg08.html):

```
<p>When you want to start a new line you can use the &lt;br /> element.  
So, the next<br />word will appear on a new line.</p>
```

Figure 1-9 shows you how the line breaks after the words “next” and “do” look.

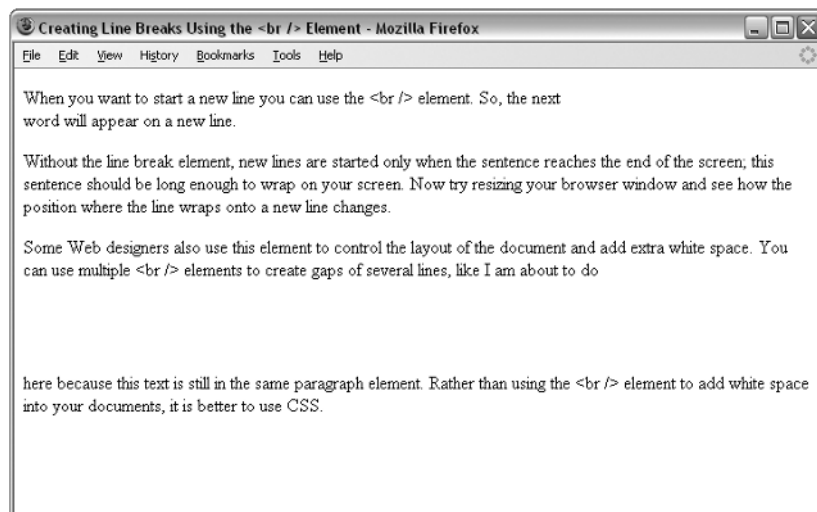


Figure 1-9

The `<br />` element can carry the core attributes as well as an attribute called `clear`, which can be used with images, and is covered in Appendix I.

```
clear class id style title
```

Creating Preformatted Text Using the `<pre>` Element

Sometimes you want your text to follow the exact format of how it is written in the XHTML document—you don’t want the text to wrap onto a new line when it reaches the edge of the browser; you don’t want it to ignore multiple spaces; and you want the line breaks where you put them.

Chapter 1: Creating Structured Documents

Any text between the opening `<pre>` tag and the closing `</pre>` tag will preserve the formatting of the source document. You should be aware, however, that most browsers would display this text in a monospaced font by default. (Courier is an example of a monospaced font, because each letter of the alphabet takes up the same width. In non-monospaced fonts, an *i* is usually narrower than an *m*.)

Two of the most common uses of the `<pre>` element are to display tabular data without the use of a table (in which case you must use the monospaced font or columns will not align correctly) and to represent computer source code. For example, the following shows some JavaScript inside a `<pre>` element (ch01_eg09.html):

```
<pre>
function testFunction(strText){
    alert (strText)
}
</pre>
```

You can see in Figure 1-10 how the content of the `<pre>` element is displayed in the monospaced font; more important, you can see how it follows the formatting shown inside the `<pre>` element—the white space is preserved.

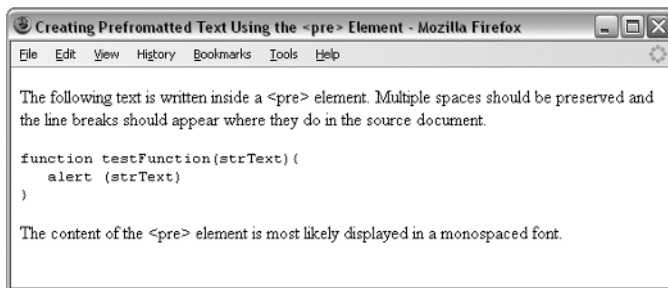


Figure 1-10

While tab characters can have an effect inside a `<pre>` element, and a tab is supposed to represent eight spaces, the implementation of tabs varies across browsers, so it is advisable to use spaces instead.

You will come across more elements that can be used to represent code later in this chapter in the section “Phrase Elements,” which covers the `<code>`, `<kbd>`, and `<var>` elements.

Firefox, IE, and Safari support an extension to the XHTML recommendation that prevents line breaks: the `<nobr>` element. (This retains the normal style of its containing element and does not result in the text being displayed in a monospaced font.) Because it is an extension, it is not valid XHTML. The `<nobr>` element is covered in Appendix I.

Try It Out Basic Text Formatting

Now that you’ve seen the basic elements that you will be using to format your text—headings and paragraphs—it’s time to try putting that information to work.

Chapter 1: Creating Structured Documents

In this example, you create a new page for a site about jazz legends, and this page tells people about Miles Davis. So, start up your text editor or web page authoring tool and follow these steps:

1. You will be creating a Strict XHTML document, so add the XML declaration and a DOCTYPE declaration to indicate that you will be writing Strict XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

2. Add the skeleton of the document: the `<html>`, `<head>`, `<title>`, and `<body>` elements. The root `<html>` element carries the `xmlns` attribute to indicate that the markup belongs to the XHTML namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <title>Jazz Legends - Miles Davis</title>
  </head>
  <body>
  </body>
</html>
```

3. Your page will have a main heading and some level 2 headings, which show the general structure of the page people will see:

```
<body>
  <h1>Jazz Legends - Miles Davis</h1>
  <h2>Styles of Miles</h2>
  <h2>Davis the Painter</h2>
</body>
```

4. You can now fill out the page with some paragraphs that follow the headings:

```
<body>
  <h1>Jazz Legends - Miles Davis</h1>
  <p>Miles Davis is known to many as one of the world's finest jazz musicians
    and an outstanding trumpet player. He also earned great respect in the
    world of music as an innovative bandleader and composer.</p>
  <h2>Styles of Miles</h2>
  <p>Miles Davis played and wrote in a variety of styles throughout his
    career, from tunes that have become jazz standards to his more
    experimental improvisational work. </p>
  <p>In the 1950s Miles was known for a warm, rich, wispy sound and was able
    to vary the color of his sound, pitch. He was also adept in using a Harmon
    mute. In the 1960s Miles began to play more in the upper register. In 1969
    he even incorporated the use of electronic instruments in his music.</p>
  <h2>Davis the Painter</h2>
  <p>Miles' love was not only for music; he is also considered a fine
    painter. Inspired by a Milan-based design movement known as Memphis,
    Miles painted a series of abstract paintings in 1988.</p>
</body>
</html>
```

Chapter 1: Creating Structured Documents

5. Save the file as `miles.html` and then open it in a web browser. The result should look something like Figure 1-11.



Figure 1-11

How It Works

The opening line of this page is the optional XML declaration. Because this is a Strict XHTML document (and therefore is an XML document), it has been included here. The next line is the `DOCTYPE` declaration, which is required in Strict XHTML documents. The `DOCTYPE` declaration indicates which version of XHTML the document conforms to.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The entire page is then contained in the root `<html>` element. The opening `<html>` tag carries the namespace identifier, which is just another way of indicating that the markup your document contains is XHTML. The `<html>` element also carries the `lang` attribute, which indicates the language that the document is written in. Our web page is written in English, so it uses the two-letter ISO code for English (the full list of country codes can be found in Appendix G). While the `lang` attribute has little practical use at the moment, it will help future-proof your documents.

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

The `<html>` element can contain only two child elements: the `<head>` element and `<body>` element. The `<head>` element contains the title for the page, and you should be able to tell from the title of the page the type of information the page will contain.

```
<head>
  <title>Jazz Legends: Miles Davis</title>
</head>
```

Chapter 1: Creating Structured Documents

Meanwhile, the `<body>` element contains the main part of the web page—the part that viewers will actually see in the main part of the web browser. Note how this page contains headings to structure the information on the page just as you would find in a word-processed document.

There are different levels of headings to help enforce structure. In this example, there is a main heading introducing Miles Davis—the main topic for this page—and then subheadings, each containing specific information about his music and other interests.

Don't forget the closing `</html>` tag at the end—after all, you must close every element correctly.

Presentational Elements

If you use a word processor, you are familiar with the ability to make text bold, italic, or underlined; these are just three of the ten options available to indicate how text can appear in HTML and XHTML. The full list is bold, italic, monospaced, underlined, strikethrough, teletype, larger, smaller, superscripted, and subscripted text.

Technically speaking, these elements affect only the presentation of a document, and the markup is of no other use, but they remain in both Transitional and Strict XHTML 1.0. As you will see later in the chapter, there are dedicated elements for indicating things like emphasis within a piece of text, and these will result in a similar presentation of the information.

All of the following presentational elements can carry the universal attributes and the UI event attributes you met earlier in the chapter.

You should also be aware that you can use CSS to get similar results, as you will see in Chapter 7.

The `<b>` Element

Anything that appears in a `<b>` element is displayed in **bold**, like the word bold here:

The following word uses a `<b>bold</b>` typeface.

This does not necessarily mean the browser will use a boldface version of a font. Some browsers use an algorithm to take a font and make the lines thicker (giving it the appearance of being bold), while others (if they cannot find a boldface version of the font) may highlight or underline the text.

This `<b>` element has the same effect as the `<strong>` element, which you will meet later, and is used to indicate that its contents have strong emphasis.

The `<i>` Element

The content of an `<i>` element is displayed in *italicized* text, like the word italic here:

The following word uses an `<i>italic</i>` typeface.

Chapter 1: Creating Structured Documents

This does not necessarily mean the browser will look for an oblique or italicized version of the font. Most browsers use an algorithm to put the lines on a slant to simulate an italic font.

The `<i>` element has the same effect as the `<em>` element, which you will meet later, and which is used to indicate that its contents have emphasis.

The `<u>` Element (deprecated)

The content of a `<u>` element is *underlined* with a simple line:

The following word would be `<u>underlined</u>`

The `<u>` element is deprecated in HTML 4 and XHTML 1.0, although it is still supported by current browsers. The preferred method is to use CSS to achieve this effect, which you'll learn about in Chapter 7.

The `<s>` and `<strike>` Elements (deprecated)

The content of an `<s>` or `<strike>` element is displayed with a *strikethrough*, which is a thin line through the text (`<s>` is just the abbreviated form of `<strike>`).

The following word would have a `<s>strikethrough</s>`.

Both the `<s>` and `<strike>` elements are deprecated in HTML 4.1 and Transitional XHTML 1.0, and were removed from Strict XHTML 1.0, although they are still supported by current browsers. The preferred method is to use CSS to achieve this effect, which you learn about in Chapter 7.

The `<tt>` Element

The content of a `<tt>` element is written in *monospaced* font.

The following word will appear in a `<tt>monospaced</tt>` font.

Figure 1-12 shows the use of the `<b>`, `<i>`, `<u>`, `<s>`, and `<tt>` elements (ch01_eg10.html).

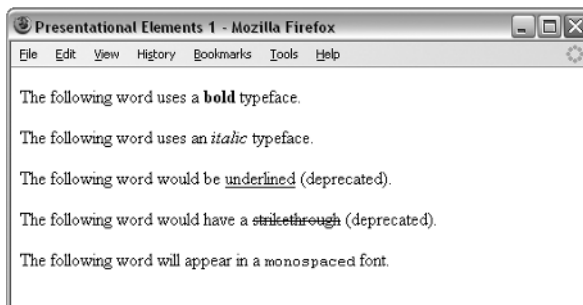


Figure 1-12

Chapter 1: Creating Structured Documents

The `<sup>` Element

The content of a `<sup>` element is written in *superscript*; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

Written on the 31st February.

The `<sup>` element is especially helpful in adding exponential values to equations, and adding the *st*, *nd*, *rd*, and *th* suffixes to numbers such as dates. However, in some browsers, you should be aware that it can create a taller gap between the line with the superscript text and the line above it.

The `<sub>` Element

The content of a `<sub>` element is written in *subscript*; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

The EPR paradox₂ was devised by Einstein, Podolsky, and Rosen.

The `<sub>` element is particularly helpful when combined with the `<a>` element (which you meet in the next chapter) to create footnotes.

The `<big>` Element

The content of the `<big>` element is displayed one font size larger than the rest of the text surrounding it. If the font is already the largest size, it has no effect. You can nest several `<big>` elements inside one another, and the content of each will get one size larger for each element.

The following word should be `<big>bigger</big>` than those around it.

In general, you should use CSS rather than the `<big>` element for formatting purposes.

The `<small>` Element

The content of the `<small>` element is displayed one font size smaller than the rest of the text surrounding it. If the font is already the smallest, it has no effect. You can nest several `<small>` elements inside one another, and the content of each gets one size smaller for each element.

The following word should be `<small>smaller</small>` than those around it.

In general, you should use CSS rather than the `<small>` element for formatting purposes.

The `<hr />` Element

The `<hr />` element creates a horizontal rule across the page. It is an empty element, rather like the `<br />` element.

`<hr />`

This is frequently used to separate distinct sections of a page where a new heading is not appropriate.

Chapter 1: Creating Structured Documents

Figure 1-13 shows the use of the `<sup>`, `<sub>`, `<big>`, `<small>`, and `<hr />` elements (ch01_eg11.html).

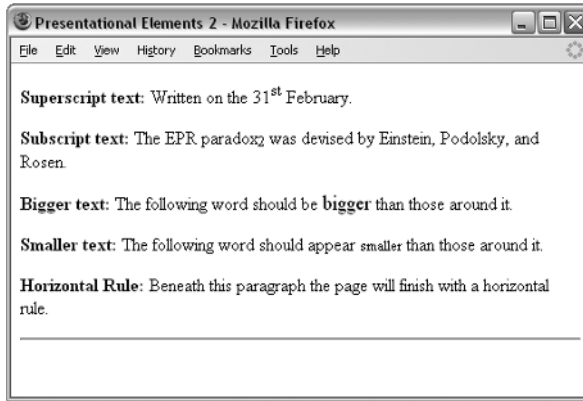


Figure 1-13

Phrase Elements

The following elements are not used as widely as the elements you have met so far. As the element names indicate, they are designed to describe their content:

- ☐ `<em>` and `<strong>` for emphasis
- ☐ `<blockquote>`, `<cite>`, and `<q>` for quotations and citations
- ☐ `<abbr>`, `<acronym>`, and `<dfn>` for abbreviations, acronyms, and key terms
- ☐ `<code>`, `<kbd>`, `<var>`, and `<samp>` for computer code and information
- ☐ `<address>` for addresses

While some of these phrase elements are displayed in a manner similar to the `<b>`, `<i>`, `<pre>`, and `<tt>` elements you have already seen, they are designed for specific purposes. For example, the `<em>` and `<strong>` elements give text emphasis and strong emphasis respectively and there are several elements for marking up quotes.

It is tempting to ignore these elements and just use the presentational elements you just met to create the same visual effect, but you should be aware of them and preferably get into the habit of using them where appropriate. For example, where you want to add emphasis to a word within a sentence you should use the `<em>` and `<strong>` elements rather than the presentational elements you just met; there are several good reasons for this, such as:

- ☐ Applications such as screen readers (which can read pages to web users with visual impairments) could add suitable intonation to the reading voice so that users with visual impairments could hear where the emphasis should be placed.
- ☐ Automated programs could be written to find the words with emphasis and pull them out as keywords within a document, or specifically index those words so that a user could find important terms in a document.

Chapter 1: Creating Structured Documents

As you can see, appropriate use of these elements adds more information to a document (such as which words should have emphasis, which are parts of programming code, which parts are addresses, and so on) rather than just saying how it should be presented visually.

All of the following phrase elements can carry the universal attributes and the UI event attributes you met earlier in the chapter.

The `<em>` Element Adds Emphasis

The content of an `<em>` element is intended to be a point of emphasis in your document, and it is usually displayed in italicized text. The kind of emphasis intended is on words such as “must” in the following sentence:

```
<p>You <em>must</em> remember to close elements in XHTML.</p>
```

You should use this element only when you are trying to add emphasis to a word, not just because you want to make the text appear italicized. If you just want italic text for stylistic reasons—without adding emphasis—you can use either the `<i>` element or CSS.

The `<strong>` Element Adds Strong Emphasis

The `<strong>` element is intended to show strong emphasis for its content—stronger emphasis than the `<em>` element. As with the `<em>` element, the `<strong>` element should be used only when you want to add strong emphasis to part of a document. Rather than being rendered in an italic font, most visual browsers display the strong emphasis in a bold font.

```
<p><em>Always</em> look at burning magnesium through protective colored  
glass as it <strong>can cause blindness</strong>.</p>
```

Figure 1-14 shows how the `<em>` and `<strong>` elements are rendered in Firefox (ch01_eg12.html).

You need to remember that how the elements are presented (italics or bold) is largely irrelevant. You should use these elements to add emphasis to phrases, and therefore give your documents greater meaning, rather than to control how they appear visually. As you will see in Chapter 7, it is quite simple with CSS to change the visual presentation of these elements—for example to highlight any words inside an `<em>` element with a yellow background and make them bold rather than italic.

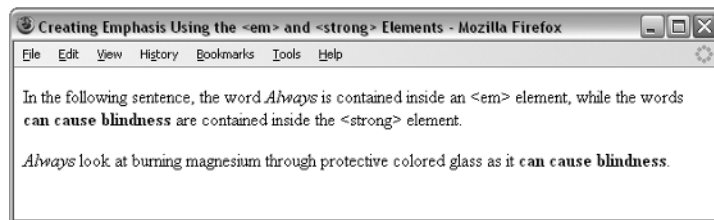


Figure 1-14

Chapter 1: Creating Structured Documents

The `<abbr>` Element Is for Abbreviations

You can indicate when you are using an abbreviated form by placing the abbreviation between opening `<abbr>` and closing `</abbr>` tags.

When possible, consider using a `title` attribute whose value is the full version of the abbreviations. If you are abbreviating a foreign word, you can also use the `xml:lang` attribute in XHTML (or the `lang` attribute in HTML).

For example, if you want to indicate that Bev is an abbreviation for Beverly, you can use the `<abbr>` element like so:

```
I have a friend called <abbr title="Beverly">Bev</abbr>.
```

The `<acronym>` Element Is for Acronym Use

The `<acronym>` element allows you to indicate that the text between an opening `<acronym>` and closing `</acronym>` tags is an acronym.

When possible use a `title` attribute whose value is the full version of the acronyms on the `<acronym>` element, and if the acronym is in a different language, include an `xml:lang` attribute in XHTML documents (or a `lang` attribute in HTML documents).

For example, if you want to indicate that XHTML was an acronym, you can use the `<acronym>` element like so (ch01_eg13.html):

```
This chapter covers marking up text in <acronym title="Extensible Hypertext
Markup Language">XHTML</acronym>.
```

As you can see from Figure 1-15, Firefox gives the `<abbr>` and `<acronym>` elements a dashed-underline, and when you hover your mouse over the word, the value of the title attribute shows as a tooltip. Internet Explorer 7 does not change the appearance of the element, although it does show the title as a tooltip.

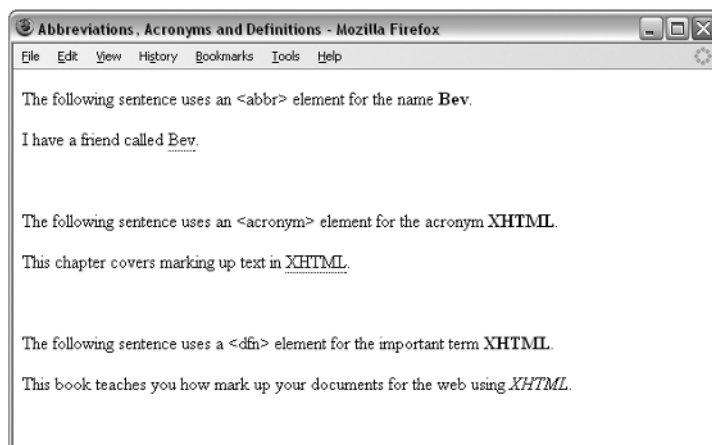


Figure 1-15

Chapter 1: Creating Structured Documents

The `<dfn>` Element Is for Special Terms

The `<dfn>` element allows you to specify that you are introducing a special term. Its use is similar to the words that are in italics in the midst of paragraphs in this book when new key concepts are introduced.

Typically, you would use the `<dfn>` element the first time you introduce a key term and only in that instance. Most recent browsers render the content of a `<dfn>` element in an italic font.

For example, you can indicate that the term “XHTML” in the following sentence is important and should be marked as such:

```
This book teaches you how mark up your documents for the Web using
<dfn>XHTML</dfn>.
```

Figure 1-15, on the previous page, shows the use of the `<dfn>` element (ch01_eg13.html).

The `<blockquote>` Element Is for Quoting Text

When you want to quote a passage from another source, you should use the `<blockquote>` element. Note that there is a separate `<q>` element for use with smaller quotations, as discussed in the next section. Here's ch01_eg14.html:

```
<p>The following description of XHTML is taken from the W3C Web site:</p>
<blockquote> XHTML 1.0 is the W3C's first Recommendation for XHTML,
following on from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML
2.0. </blockquote>
```

Text inside a `<blockquote>` element is usually indented from the left and right edges of the surrounding text, and sometimes uses an italicized font (but it should be used only for quotes; if you simply want this effect on a paragraph of text, you should use CSS). You can see what this looks like in Figure 1-16.

Using the `cite` Attribute with the `<blockquote>` Element

You can use the `cite` attribute on the `<blockquote>` element to indicate the source of the quote. The value of this attribute should be a URL pointing to an online document, if possible the exact place in that document. Browsers will not actually do anything with this attribute, but it means the source of the quote is there should you need it in the future—it could also be used by other processing applications (ch01_eg14.html).

```
<blockquote cite="http://www.w3.org/markup/"> XHTML 1.0 is the W3C's first
Recommendation for XHTML, following on from earlier work on HTML 4.01, HTML
4.0, HTML 3.2 and HTML 2.0.</blockquote>
```

At the time of this writing, some validators had trouble with the `cite` attribute, such as the W3C validator, which does not recognize the presence of the `cite` attribute on the `<blockquote>` element.

Chapter 1: Creating Structured Documents

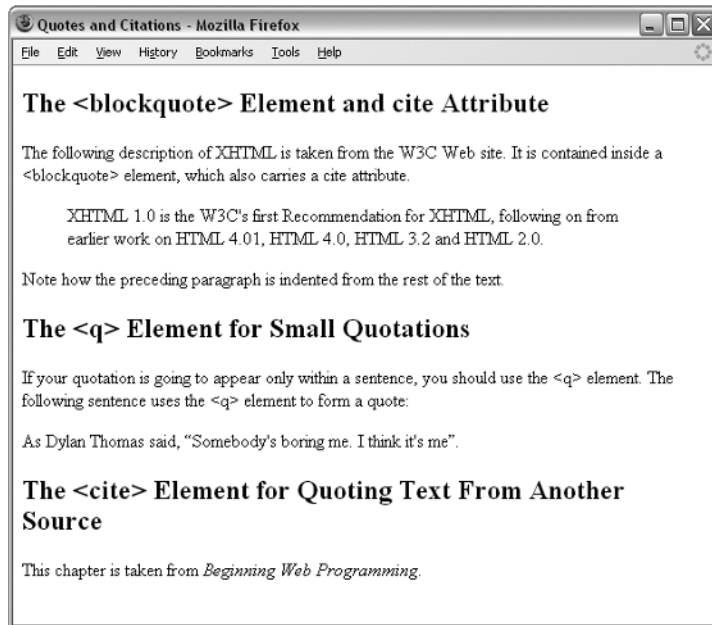


Figure 1-16

The `<q>` Element Is for Short Quotations

The `<q>` element is intended to be used when you want to add a quote within a sentence rather than as an indented block on its own (ch01_eg14.html):

```
<p>As Dylan Thomas said, <q>Somebody's boring me. I think it's me</q>.</p>
```

The HTML and XHTML recommendations say that the text enclosed in a `<q>` element should begin and end in double quotes. Firefox inserts these quotation marks for you, whereas IE7 does not. So, if you want your quote to be surrounded by quotation marks, be warned that inserting them in the document will result in two sets of quotes in Firefox. Neither IE nor Firefox changes the appearance of this element in any other way.

The `<q>` element can also carry the `cite` attribute. The value should be a URL pointing to the source of the quote.

The `<cite>` Element Is for Citations

If you are quoting a text, you can indicate the source by placing it between an opening `<cite>` tag and closing `</cite>` tag. As you would expect in a print publication, the content of the `<cite>` element is rendered in italicized text by default (ch01_eg12.html).

```
This chapter is taken from <cite>Beginning Web Development</cite>.
```

Chapter 1: Creating Structured Documents

If you are referencing an online resource, you should place your `<cite>` element inside an `<a>` element, which, as you'll see in Chapter 2, creates a link to the relevant document.

There are several applications that potentially could make use of the `<cite>` element. For example, a search application could use `<cite>` tags to find documents that reference certain works, or a browser could collect the contents of `<cite>` elements to generate a bibliography for any given document, although at the moment it is not widely enough used for either feature to exist.

You can see the `<blockquote>`, `<q>`, and `<cite>` elements in Figure 1-16.

The `<code>` Element Is for Code

If your pages include any programming code (which is not uncommon on the Web), the following four elements will be of particular use to you. Any code to appear on a web page should be placed inside a `<code>` element. Usually the content of the `<code>` element is presented in a monospaced font, just like the code in most programming books (including this one).

Note that you cannot just use the opening and closing angle brackets inside these elements if you want to represent XHTML markup. The browser could mistake these characters for actual markup. You should use `<` instead of the left-angle bracket `<`, and you should use `>` instead of the right-angle bracket `>`. A list of all these character entities is in Appendix F.

Here you can see an example of using the `<code>` element to represent an `<h1>` element and its content in XHTML (`ch01_eg15.html`):

```
<p><code>&lt;h1&gt;This is a primary heading&lt;/h1&gt;</code></p>
```

Figure 1-17 shows you how this would look in a browser.

The use of the `<code>` element could theoretically allow search applications to look at the content of `<code>` elements to help them find a particular code segment. The `<code>` element is often used in conjunction with the `<pre>` element so that the formatting of the code is retained.

The `<kbd>` Element Is for Text Typed on a Keyboard

If, when talking about computers, you want to tell a reader to enter some text, you can use the `<kbd>` element to indicate what should be typed in, as in this example (`ch01_eg15.html`):

```
<p>Type in the following: <kbd>This is the kbd element</kbd>.</p>
```

The content of a `<kbd>` element is usually represented in a monospaced font, rather like the content of the `<code>` element. Figure 1-17 shows you what this would look like in a browser.

Chapter 1: Creating Structured Documents

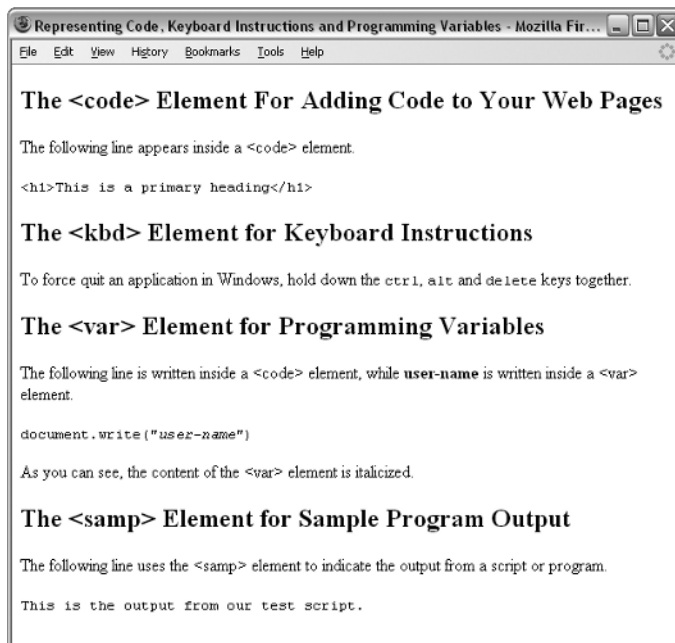


Figure 1-17

The <var> Element Is for Programming Variables

The `<var>` element is another of the elements added to help programmers. It is usually used in conjunction with the `<pre>` and `<code>` elements to indicate that the content of that element is a variable that can be supplied by a user (`ch01_eg15.html`).

```
<p><code>document.write("<var>user-name</var>")</code></p>
```

Typically the content of a `<var>` element is italicized, as you can see in Figure 1-17.

If you are not familiar with the concept of variables, they are covered in Chapter 11.

The <samp> Element Is for a Program Output

The `<samp>` element indicates sample output from a program, script, or the like. Again, it is mainly used when documenting programming concepts. For example (`ch01_eg15.html`):

```
<p>If everything worked you should see the result <samp>Test completed  
OK</samp>.</p>
```

This tends to be displayed in a monospaced font, as you can see in Figure 1-17.

Chapter 1: Creating Structured Documents

The `<address>` Element Is for Addresses

Many documents need to contain a snail-mail address, and there is a special `<address>` element that is used to contain addresses. For example, here is the address for Wrox, inside an `<address>` element (ch01_eg16.html):

```
<address>Wrox Press, 10475 Crosspoint Blvd, Indianapolis, IN 46256</address>
```

A browser can display the address differently than the surrounding document, and IE, Firefox, and Safari display it in italics, as you can see in Figure 1-18 (although you can override this with CSS).

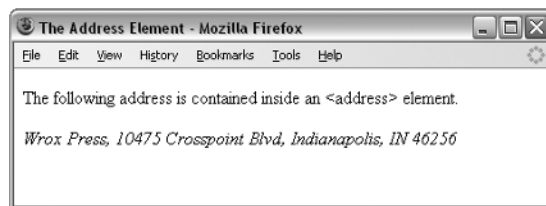


Figure 1-18

Indicating who wrote a document or who is responsible for it adds credibility to a document that is otherwise anonymous. The `<address>` element is a good way to add this at the end of the document. It can also help automated applications read addresses from documents.

That brings you to the end of the phrase elements, but not quite the end of all the text elements.

Lists

There are many reasons why you might want to add a list to your pages, from putting your five favorite albums on your home page to including a numbered set of instructions for visitors to follow (like the steps you follow in the Try It Out examples in this book).

You can create three types of lists in XHTML:

- ☐ **Unordered lists**, which are like lists of bullet points
- ☐ **Ordered lists**, which use a sequence of numbers or letters instead of bullet points
- ☐ **Definition lists**, which allow you to specify a term and its definition

I'm sure you will think of more uses for the lists as you meet them and start using them.

Using the `<ul>` Element to Create Unordered Lists

If you want to make a list of bullet points, you write the list within the `<ul>` element (which stands for unordered list). Each bullet point or line you want to write should then be contained between opening `<li>` tags and closing `</li>` tags (the `li` stands for *list item*).

Chapter 1: Creating Structured Documents

You should always close the `<li>` element, even though you might see some HTML pages that leave off the closing tag. This is a bad habit you should avoid.

If you want to create a bulleted list, you can do so like this (`ch01_eg17.html`):

```
<ul>
  <li>Bullet point number one</li>
  <li>Bullet point number two</li>
  <li>Bullet point number three</li>
</ul>
```

In a browser, this list would look something like Figure 1-19.

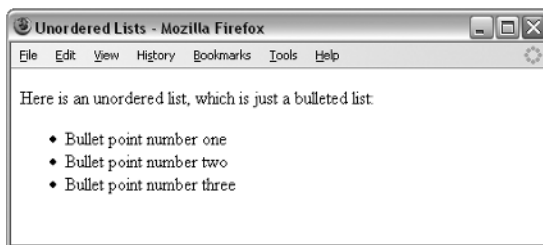


Figure 1-19

The `<ul>` and `<li>` elements can carry all the universal attributes and UI event attributes.

The `<ul>` element could also carry an attribute called `compact` in HTML 4.1—which is still allowed in Transitional XHTML but not in Strict XHTML 1.0—the purpose of which was to make the bullet points vertically closer together. Its value should also be `compact`, like so:

```
<ul compact="compact">
  <li>Item one</li>
  <li>Item two</li>
  <li>Item three</li>
</ul>
```

Ordered Lists

Sometimes, you want your lists to be ordered. In an ordered list, rather than prefixing each point with a bullet point, you can use either numbers (1, 2, 3), letters (A, B, C), or Roman numerals (i, ii, iii) to prefix the list item.

An ordered list is contained inside the `<ol>` element. Each item in the list should then be nested inside the `<li>` element and contained between opening `<li>` and closing `</li>` tags (`ch01_eg18.html`).

```
<ol>
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

Chapter 1: Creating Structured Documents

The result should be similar to what you see in Figure 1-20.

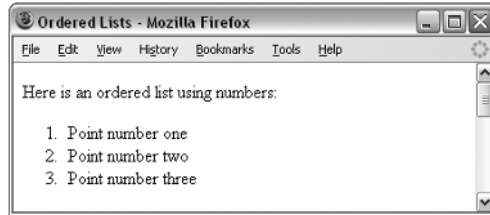


Figure 1-20

If you would rather have letters or Roman numerals than Arabic numbers, you must use the now-deprecated `type` attribute on the `<ol>` element.

Using the `type` Attribute to Select Numbers, Letters, or Roman Numerals in Ordered Lists (deprecated)

The `type` attribute on the `<ol>` element allows you to change the ordering of list items from the default of numbers to the options listed in the table that follows, by giving the `type` attribute the corresponding character.

Value for <code>type</code> Attribute	Description	Examples
1	Arabic numerals (the default)	1, 2, 3, 4, 5
A	Capital letters	A, B, C, D, E
a	Small letters	a, b, c, d, e
I	Large Roman numerals	I, II, III, IV, V
i	Small Roman numerals	i, ii, iii, iv, v

For example, here is an ordered list that uses small Roman numerals (`ch01_eg18.html`):

```
<ol type="i">
  <li>This is the first point</li>
  <li>This is the second point</li>
  <li>This is the third point</li>
</ol>
```

You can see what this might look like in Figure 1-21.

The `type` attribute was deprecated in HTML 4.1 in favor of the CSS `list-style-type` property; it will therefore work only in Transitional XHTML not Strict XHTML 1.0. The CSS replacement will work only in browsers since IE4 and Netscape 4 browsers.

Chapter 1: Creating Structured Documents

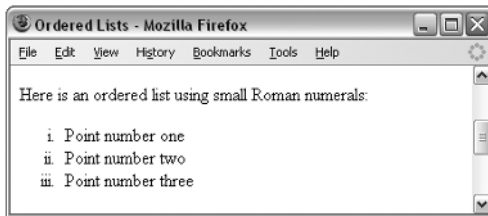


Figure 1-21

You used to be able to use the `type` attribute on `<li>` elements, which would override the value in the `<ol>` element, but it was deprecated in HTML 4.1 and its use should be avoided. All of the universal attributes and UI event attributes can be used with the `<ol>` elements, and also a special attribute `start`, to control the number a list starts at.

Using the `start` Attribute to Change the Starting Number in Ordered Lists (deprecated)

If you want to specify the number that a numbered list should start at, you can use the `start` attribute on the `<ol>` element. The value of this attribute should be the numeric representation of that point in the list, so a D in a list that is ordered with capital letters would be represented by the value 4 (`ch01_eg18.html`).

```
<ol type="i" start="4">
  <li>Point number one</li>
  <li>Point number two</li>
  <li>Point number three</li>
</ol>
```

You can see the result in Figure 1-22 .

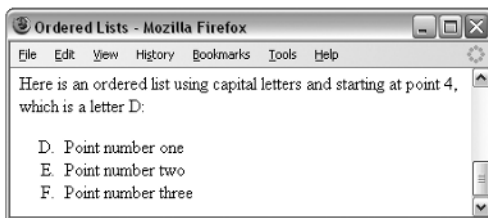


Figure 1-22

The `start` attribute was deprecated in HTML 4.1; it will therefore work in Transitional XHTML 1.0 but not in Strict XHTML 1.0.

Definition Lists

The definition list is a special kind of list for providing terms followed by a short text definition or description for them. Definition lists are contained inside the `<dl>` element. The `<dl>` element then contains alternating `<dt>` and `<dd>` elements. The content of the `<dt>` element is the term you will be defining.

Chapter 1: Creating Structured Documents

The `<dd>` element contains the definition of the previous `<dt>` element. For example, here is a definition list that describes the different types of lists in XHTML (`ch01_eg19.html`):

```
<dl>
  <dt>Unordered List</dt>
  <dd>A list of bullet points.</dd>
  <dt>Ordered List</dt>
  <dd>An ordered list of points, such as a numbered set of steps.</dd>
  <dt>Definition List</dt>
  <dd>A list of terms and definitions.</dd>
</dl>
```

In a browser, this would look something like Figure 1-23 (`ch01_eg19.html`).

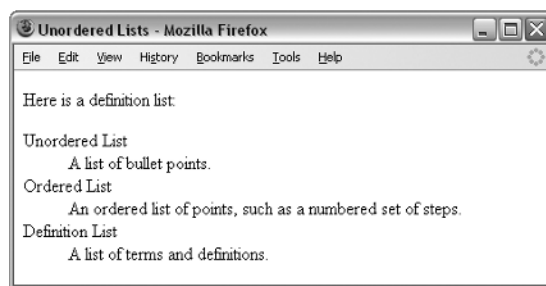


Figure 1-23

Each of these elements can carry the universal attributes and UI event attributes.

Nesting Lists

You can nest lists inside other lists. For example, you might want a numbered list with separate points corresponding to one of the list items. Each list will be numbered separately unless you specify otherwise using the `start` attribute. And each new list should be placed inside a `<li>` element (`ch01_eg20.html`):

```
<ol type="I">
  <li>Item one</li>
  <li>Item two</li>
  <li>Item three</li>
  <li>Item four
    <ol type="i">
      <li>Item 4.1</li>
      <li>Item 4.2</li>
      <li>Item 4.3</li>
    </ol>
  </li>
  <li>Item Five</li>
</ol>
```

Chapter 1: Creating Structured Documents

In a browser, this will look something like Figure 1-24.

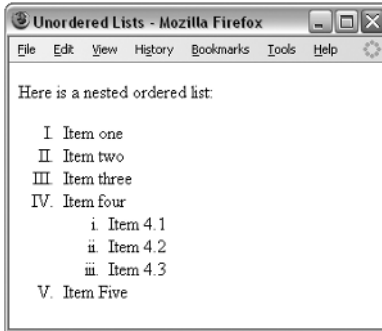


Figure 1-24

Try It Out Using Text Markup

Now that you've looked at the different elements and attributes you can use to mark up text, it is time to put the information into practice. In this example, you use a mixture of the text markup to create a page that displays a recipe. So, open up your text editor or web page authoring tool and follow these steps:

1. You will be writing this example in Transitional XHTML 1.0, so add the optional XML declaration, and the DOCTYPE declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

2. Add the skeleton elements for the document: `<html>`, `<head>`, `<title>`, and `<body>`. Don't forget to put the namespace identifier on the root element, along with an attribute to indicate the language of the document:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <title>Wrox Recipes - World's Best Scrambled Eggs</title>
  </head>
  <body>
  </body>
</html>
```

3. Add some appropriate heading elements into the body of the document:

```
<body>
  <h1>Wrox Recipes - World's Best Scrambled Eggs</h1>
  <h2>Ingredients</h2>
  <h2>Instructions</h2>
</body>
```

Chapter 1: Creating Structured Documents

4. After the `<h1>` element, there will be a bit of an explanation about the recipe (and why it is the World's Best). You can see that several of the elements you have met so far are tucked away in these two paragraphs.

```
<h1>Wrox Recipes - World's Best Scrambled Eggs</h1>
<p>I adapted this recipe from a book called
<cite cite=" http://www.amazon.com/exec/obidos/tg/detail/-
/0864119917/">Sydney Food</cite> by Bill Grainger. Ever since tasting
these eggs on my 1<sup>st</sup> visit to Bill's restaurant in Kings
Cross, Sydney, I have been after the recipe. I have since transformed
it into what I really believe are the <em>best</em> scrambled eggs
I have ever tasted.</p>
<p>This recipe is what I call a <q>very special breakfast</q>; just look at
the ingredients to see why. It has to be tasted to be believed.</p>
```

5. After the first `<h2>` element, you will list the ingredients in an unordered list:

```
<h2>Ingredients</h2>
<p>The following ingredients make one serving:</p>
<ul>
  <li>2 eggs</li>
  <li>1 tablespoon of butter (10g)</li>
  <li>1/3 cup of cream <i>(2 3/4 fl ounces)</i></li>
  <li>A pinch of salt</li>
  <li>Freshly milled black pepper</li>
  <li>3 fresh chives (chopped)</li>
</ul>
```

6. Add the instructions after the second `<h2>` element; these will go in a numbered list:

```
<h2>Instructions</h2>
<ol>
  <li>Whisk eggs, cream, and salt in a bowl.</li>
  <li>Melt the butter in a non-stick pan over a high heat <i>(taking care
not to burn the butter)</i>.</li>
  <li>Pour egg mixture into pan and wait until it starts setting around
the edge of the pan (around 20 seconds).</li>
  <li>Using a wooden spatula, bring the mixture into the center as if it
were an omelet, and let it cook for another 20 seconds.</li>
  <li>Fold contents in again, leave for 20 seconds, and repeat until
the eggs are only just done.</li>
  <li>Grind a light sprinkling of freshly milled pepper over the eggs
and blend in some chopped fresh chives.</li>
</ol>
<p>You should only make a <strong>maximum</strong> of two servings per
frying pan.</p>
```

7. Save this example as `eggs.html`. When you open it in a browser you should see something like Figure 1-25.

Chapter 1: Creating Structured Documents



Figure 1-25

How It Works

You have seen the XML declaration and the skeleton of this document enough times already, so now it's time to focus on the new elements you have available to mark up text.

After the main heading for the document, which is contained in the `<h1>` elements, you can see two paragraphs of text. Start by looking at the first paragraph.

In the first sentence, the `<cite>` element has been used to indicate a reference to the book this recipe is adapted from. The next sentence makes use of the `<sup>` element so you can write "1st" and use superscript text—although you will note that this makes the gap between the first line and the second line of text larger than the gap between the second and third lines of text (as the superscript letters poke above

Chapter 1: Creating Structured Documents

the line). In the final sentence there is emphasis on the word “best,” as these really are the *best* scrambled eggs I have ever tasted:

```
<h1>Wrox Recipes- World's Best Scrambled Eggs</h1>
<p>I adapted this recipe from a book called
  <cite cite="http://www.bills.com.au">Sydney Food</cite> by Bill Grainger.
  Ever since tasting these eggs on my 1<sup>st</sup> visit to Bill's
  restaurant in Kings Cross, Sydney, I have been after the recipe. I have
  since transformed it into what I really believe are the <em>best</em>
  scrambled eggs I have ever tasted. </p>
```

You can see another new element at work in the second element: the `<q>` element for quotes that are sprinkled into a sentence:

```
<p>Although this recipe may be what I call a <q>very special breakfast</q>,
  just look at the ingredients to see why, it has to be tasted to be
  believed.</p>
```

The ingredients (listed under an `<h2>` element) contain an unordered list, and an italicized alternative measure for the amount of cream required:

```
<ul>
  <li>2 eggs</li>
  <li>10g butter</li>
  <li>1/3 cup of cream <i>(2 3/4 fl ounces)</i></li>
  <li>a pinch of salt</li>
  <li>freshly milled black pepper</li>
  <li>3 fresh chives (chopped)</li>
</ul>
```

The instructions for cooking the eggs (listed under the second `<h2>` element) contain a numbered list and a couple of additional paragraphs. You might note that the numbered list contains an italicized comment about not burning the butter, and the final paragraph contains a strong emphasis that you should cook no more than two batches of these eggs in a pan.

```
<h2>Instructions</h2>
<p>The following ingredients make one serving.</p>
<ol>
  <li>Whisk eggs, cream, and salt in a bowl.</li>
  <li>Melt the butter in a non-stick pan over a high heat <i>(taking care
    not to burn the butter)</i>.</li>
  <li>Pour egg mixture into pan, and wait until it starts setting
    around the edge of the pan (around twenty seconds).</li>
  <li>Using a wooden spatula, bring the mixture into the center as
    if it was an omelet, and let it cook for another 20 seconds.</li>
  <li>Fold contents in again, leave for 20 seconds, and repeat until
    the eggs are only just done.</li>
  <li>Grind a light sprinkling of freshly milled pepper over the eggs
    and blend in some chopped fresh chives.</li>
</ol>
<p>You should only make a <strong>maximum</strong> of two servings per
  frying pan.</p>
```

Chapter 1: Creating Structured Documents

The page then finishes up as usual with closing `</body>` and `</html>` tags. I hope you will enjoy the eggs—go on, you know you want to try them now.

Editing Text

When working on a document with others, it helps if you can see changes that another person has made. Even when working on your own documents, it can be helpful to keep track of changes you make. Two elements are specifically designed for revising and editing text:

- ☐ The `<ins>` element for when you want to add text
- ☐ The `<del>` element for when you want to delete some text

Here you can see some changes made to the following XHTML (`ch01_eg21.html`):

```
<h1>How to Spot a Wrox Book</h1>
<p>Wrox-spotting is a popular pastime in bookshops. Programmers like to find
the distinctive <del>blue</del><ins>red</ins> spines because they know that
Wrox books are written by <del>1000 monkeys</del><ins>Programmers</ins> for
Programmers.</p>
<ins><p>Both readers and authors, however, have reservations about the use
of photos on the covers.</p></ins>
```

This example would look something like Figure 1-26 in a browser.

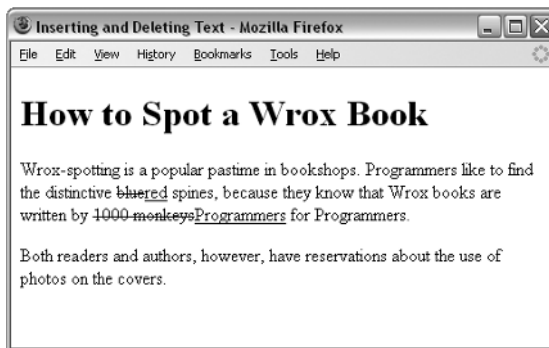


Figure 1-26

These features would also be particularly helpful in editing tools to note changes and modifications made by different authors.

If you are familiar with Microsoft Word, the `<ins>` and `<del>` elements are very similar to a feature called Track Changes (which you can find under the Tools menu). The track changes feature underlines new text additions and crosses through deleted text.

Chapter 1: Creating Structured Documents

You must be careful when using `<ins>` and `<del>` to ensure that you do not end up with a block-level element (such as a `<p>` or an `<h2>` element) inside an inline element such as a `<b>` or `<i>` element. You learn more about block-level elements and inline elements at the end of the chapter.

Using `<ins>` to Indicate New Additions to Text

Any text added to a document inside an `<ins>` element will be underlined to indicate that it is new text (refer to Figure 1-26).

```
<ins><p>This paragraph is contained inside an &lt;ins&gt; element.</p></ins>
```

You can use the `cite` attribute on the `<ins>` and `<del>` element to indicate the source or reason for a change, although this attribute is quite limiting as the value must be a URI.

You might also use the `title` attribute to provide information as to who added the `<ins>` or `<del>` element and why it was added or deleted; this information is offered to users as a tooltip in the major browsers.

The `<ins>` and `<del>` elements can also carry a `datetime` attribute whose value is a date and time in the following format:

```
YYYY-MM-DDThh:mm:ssTZD
```

This formula breaks down as follows:

- ☐ `YYYY` represents the year.
- ☐ `MM` represents the month.
- ☐ `DD` represents the day of the month.
- ☐ `T` is just a separator between the date and time.
- ☐ `hh` is the hour.
- ☐ `mm` is the number of minutes.
- ☐ `ss` is the number of seconds.
- ☐ `TZD` is the time zone designator.

For example, `2004-04-16T20:30-05:00` represents 8:30 p.m. on April 16, 2004, according to U.S. Eastern Standard Time.

The `datetime` attribute is likely to be entered only by a program or authoring tool, as the format is rather long to be entered by hand.

Using `<del>` to Indicate Deleted Text

If you want to delete some text from a document, you can place it inside a `<del>` element to indicate that it is marked to be deleted. Text inside a `<del>` element will have a line or strikethrough (refer to Figure 1-26).

```
<del><p>This paragraph is contained inside a &lt;del&gt; element.</p></del>
```

Chapter 1: Creating Structured Documents

The `<del>` element can carry the `cite`, `datetime`, and `title` attributes just like the `<ins>` element.

When you learn how to use CSS, you will see how it would be possible to show and hide the inserted and deleted content as required.

Using Character Entities for Special Characters

You can use most alphanumeric characters in your document and they will be displayed without a problem. There are, however, some characters that have special meaning in XHTML, and for some characters there is not an equivalent on the keyboard you can enter. For example, you cannot use the angle brackets that start and end tags, as the browser can mistake the following letters for markup. You can, however, use a set of different characters known as a *character entity* to represent these special characters. Sometimes you will also see character entities referred to as *escape characters*.

All special characters can be added into a document using the numeric entity for that character, and some also have named entities, as you can see in the table that follows.

Character	Numeric Entity	Named Entity
"	"	"
&	&	&
<	<	<
>	>	>

A full list of character entities (or special characters) appears in Appendix F.

Comments

You can put comments between any tags in your XHTML documents. Comments use the following syntax:

```
<!-- comment goes here -->
```

Anything after `<!--` until the closing `-->` will not be displayed. It can still be seen in the source code for the document, but it is not shown onscreen.

It is good practice to comment your code, especially in complex documents, to indicate sections of a document, and any other notes to anyone looking at the code. Comments help you and others understand your code.

Chapter 1: Creating Structured Documents

You can even comment out whole sections of code. For example, in the following snippet of code you would not see the content of the `<h2>` element. You can also see there are comments indicating the section of the document, who added it, and when it was added.

```
<!-- Start of Footnotes Section added 04-24-04 by Bob Stewart -->
<!-- <h2>Character Entities</h2> -->
<p><strong>Character entities</strong> can be used to escape special
characters that the browser might otherwise think have special meaning.</p>
<!-- End of Footnotes section -->
```

The `<font>` Element (deprecated)

You should be aware of the `<font>` element, which was introduced in HTML 3.2 to allow users more control over how text appears. It was deprecated in HTML 4.0, and has since been removed from XHTML. In its short life, however, it got a lot of use, and if you look at other people's code you will see it used a lot. If you want to read more about the `<font>` element, it is covered in Appendix I. You might see the `<font>` element used like so:

```
<h3>Using the &lt;font> element</h3>
<font face="arial, verdana, sans-serif" size="2" color="#666666">The
&lt;font> element has been deprecated since HTML 4.0. You should now use
CSS to indicate how text should be styled. </font>
```

Understanding Block and Inline Elements

Now that you have seen many of the elements that can be used to mark up text, it is important to make an observation about all of these elements that live inside the `<body>` element because each one can fall into one of two categories:

- ☐ Block-level elements
- ☐ Inline elements

This is quite a conceptual distinction, but it will have important ramifications for other features of XHTML (some of which you are about to meet).

Block-level elements appear on the screen as if they have a carriage return or line break before and after them. For example the `<p>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<ul>`, `<ol>`, `<dl>`, `<pre>`, `<hr />`, `<blockquote>`, and `<address>` elements are all block-level elements. They all start on their own new line, and anything that follows them appears on its own new line, too.

Inline elements, on the other hand, can appear within sentences and do not have to appear on a new line of their own. The `<b>`, `<i>`, `<u>`, `<em>`, `<strong>`, `<sup>`, `<sub>`, `<big>`, `<small>`, `<li>`, `<ins>`, `<del>`, `<code>`, `<cite>`, `<dfn>`, `<kbd>`, and `<var>` elements are all inline elements.

Chapter 1: Creating Structured Documents

For example, look at the following heading and paragraph. These elements start on their own new line and anything that follows them goes on a new line, too. Meanwhile the inline elements in the paragraph are not placed on their own new line. Here is the code (ch02_eg22.html):

```
<h1>Block-Level Elements</h1>
<p><strong>Block-level elements</strong> always start on a new line. The
<code>&lt;h1&gt;</code> and <code>&lt;p&gt;</code> elements will not sit
on the same line, whereas the inline elements flow with the rest of the
text.</p>
```

You can see what this looks like in Figure 1-27.



Figure 1-27

You should also be aware that in Strict XHTML, block-level elements can contain other block-level elements, and inline elements. However, inline elements can appear only within block-level elements, and they may not contain block-level elements (so you should not have a `<b>` element outside a block-level element).

Grouping Elements with `<div>` and `<span>`

The `<div>` and `<span>` elements allow you to group several elements to create sections or subsections of a page. On their own, they will not affect the appearance of a page, but they are commonly used with CSS to allow you to attach a style to a section of a page (as you will see in Chapter 7). For example, you might want to put all of the footnotes on a page within a `<div>` element to indicate that all of the elements within that `<div>` element relate to the footnotes. You might then attach a style to this `<div>` element so that they appear using a special set of style rules.

The `<div>` element is used to group block-level elements:

```
<div class="footnotes">
  <h2>Footnotes</h2>
  <p><b>1</b> The World Wide Web was invented by Tim Berners-Lee</p>
  <p><b>2</b> The W3C is the World Wide Web Consortium who maintain many Web
    standards</p>
</div>
```

Chapter 1: Creating Structured Documents

The `<span>` element, on the other hand, can be used to group inline elements only. So, if you had a part of a sentence or paragraph you wanted to group, you could use the `<span>` element. Here you can see that I have added a `<span>` element to indicate which content refers to an inventor. It contains both a bold element and some text:

```
<div class="footnotes">
  <h2>Footnotes</h2>
  <p><span class="inventor"><b>1</b> The World Wide Web was invented by Tim
    Berners Lee</span></p>
  <p><b>2</b> The W3C is the World Wide Web Consortium who maintain many Web
    standards</p>
</div>
```

On its own, this would have no effect at all on how the document looks visually, but it does add extra meaning to the markup, which now groups the related elements. This grouping can either be used by a processing application, or (as you will see in Chapter 7) can be used to attach special styles to these elements using CSS rules.

The `<div>` and `<span>` elements can carry all of the universal attributes and UI event attributes, as well as the deprecated `align` attribute (which is no longer available in Strict XHTML 1.0).

Summary

In this chapter you have seen how XHTML is used to add structure to the text that appears in a document.

You have learned that the contents of a web page is marked up using elements that describe the structure of the document. These elements consist of an opening tag, a closing tag, and some content between the opening and closing tags. In order to alter some properties of elements, the opening tag may carry attributes, and attributes are always written as name value pairs. You know that XHTML can be thought of as the latest version of HTML, and that there are three different flavors of XHTML—in order to tell the browser which you are using, you can use a DOCTYPE declaration.

You also met a lot of new elements and learned the attributes they can carry. You've seen how every XHTML document should contain at least the `<html>`, `<head>`, `<title>`, and `<body>` elements, and how the `<html>` element should carry a namespace identifier.

You then met some attributes: the core attributes (`class`, `id`, and `title`), the internationalization attributes (`dir`, `lang`, and `xml:lang`), and the UI event attributes, each of which will crop up regularly throughout the book, as most of the elements can support them.

The rest part of this chapter dealt with elements that describe the structure of text:

- ❑ The six levels of headings: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`
- ❑ Paragraphs `<p>`, preformatted sections `<pre>`, line breaks `<br />`, and addresses `<address>`
- ❑ Presentational elements `<b>`, `<i>`, `<u>`, `<s>`, `<tt>`, `<sup>`, `<sub>`, `<strike>`, `<big>`, `<small>`, and `<hr />`

Chapter 1: Creating Structured Documents

- ☐ Phrase elements such as `<em>`, `<strong>`, `<abbr>`, `<acronym>`, `<dfn>`, `<blockquote>`, `<q>`, `<cite>`, `<code>`, `<kbd>`, `<var>`, `<samp>`, and `<address>`
- ☐ Lists such as unordered lists using `<ul>` and `<li>`, ordered lists using `<ol>` and `<li>`, and definition lists using `<dl>`, `<dt>`, and `<dd>`
- ☐ Editing elements such as `<ins>` and `<del>`
- ☐ Grouping elements `<div>` and `<span>`

You will obviously use some of these elements more than others, but where an element fits the content you are trying to mark up, from paragraphs to addresses, you should try to use it. Structuring your text properly will help it last longer than if you just format it using line breaks and presentational elements.

You will come across many of these elements in later examples in this book, starting with the next chapter, which introduces you to the very important topic of linking between documents (and linking to specific parts of a document).

Finally, I should mention that you can learn a lot from looking at how other people have written their pages, and you can view the HTML or XHTML of pages on the Web by going to the View or Tools menu in your browser and selecting the Source (sometimes listed as View Source or Page Source) options. (You can also learn a lot of bad habits this way too—so you still need to read on in order to avoid them.)

Exercises

The answers to all of the exercises are in Appendix A.

1. Mark up the following sentence with the relevant presentational elements.

The 1st time the **bold** man wrote in *italics*, he underlined several key words.

2. Mark up the following list, with inserted and deleted content:

Ricotta pancake ingredients:

- ☐ 1 ~~1/2~~ 3/4 cups ricotta
- ☐ 3/4 cup milk
- ☐ 4 eggs
- ☐ 1 cup plain white flour
- ☐ 1 teaspoon baking powder
- ☐ ~~75g~~ 50g butter
- ☐ pinch of salt

