



In *Stylin'*, you learn to future-proof your site by separating the content from the presentation; you do this by creating pages of XHTML markup with only content in them, and then, using a single line of code, you link these pages to a separate file called a *style sheet*, which contains the presentation rules that define how the markup should be displayed.

The power of this church-and-state separation is this—you can have different style sheets for browsers, for PDAs, for cell phones, for printing, for screen readers for the visually impaired, and so on; each style sheet causes the content to be presented in the best possible way for that use, but you only ever need *one* version of the XHTML content markup. As you will see, an XHTML page can automatically select the correct style sheet for each environment in which it finds itself. In this way, your write-once, use-many content becomes truly portable, flexible, and ready for whatever presentational requirements the future may bring its way. Note, however, that like any great vision of the future, there are still some current realities that we need to deal with.

## XHTML and How to Write It

Because CSS is a mechanism for styling XHTML, you can't start using CSS until you have a solid grounding in XHTML. And what, exactly, is XHTML? XHTML is a reformulation of HTML as XML—didja get that? Put (very) simply, XHTML is based on the free-form structure of XML, where tags can be named to actually describe the content they contain; for example, `<starname>Cher</starname>`. This very powerful capability of XML means that when you develop your set of custom tags for your XML content, you also must create a second document, known as a *DTD* (document type definition) or a similarly formatted XML schema, to explain to the device that is interpreting the XML for how to handle those tags.

XML has been almost universally adopted in business, and the fact that the same X (for eXtensible) is now in XHTML emphasizes the unstoppable movement toward the separation of presentation and content.

The rest of this chapter is dedicated to the latest, completely reformulated, totally-modern, and altogether more flexible version of HTML. Ladies and gentlemen, please welcome... XHTML!



If you want more than this rather simplistic description of XML, check out the XML tutorial at the SpiderPro Web site ([www.spiderpro.com/bu/buxxmlm001.html](http://www.spiderpro.com/bu/buxxmlm001.html)).

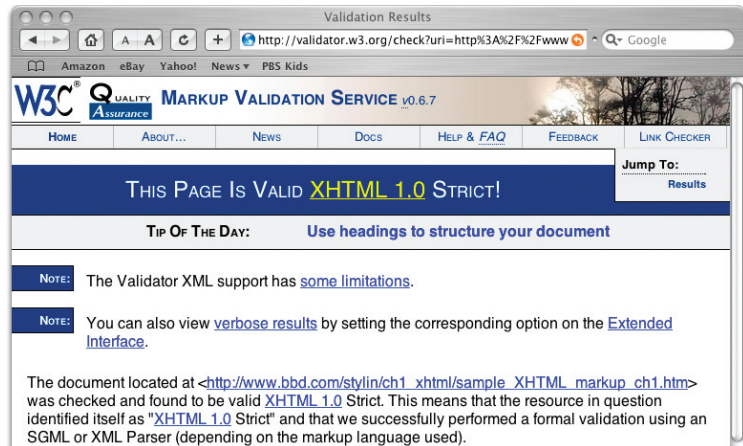
## XHTML Markup Rules

Correctly written XHTML markup gives you the best chance that your pages will display correctly in a broad variety of devices for years to come. The clean, easy-to-write, and flexible nature of XHTML produces code that loads fast, is easy to understand when editing, and prepares your content for use in a variety of applications.

You can easily determine if your site complies with Web standards—if your markup is *well-formed and valid XHTML*, and your style sheet is *valid CSS*, then it will comply. (Whether it's well designed or not is a rather more subjective matter, but we will consider that as we go along.)

*Well formed* means that the XHTML is structured correctly, according to the markup rules described in this chapter. *Valid* means the markup contains only XHTML, with no meaningless tags, tags that are not closed properly, or *deprecated* (phased out, but still operational) HTML tags. You can check to see if your page meets these criteria by uploading the page onto a server and then going to <http://validator.w3.org> and entering the page's URL. Press Submit, and in a few seconds you are presented with either a detailed list of the page's errors or the very satisfying "This Page Is Valid XHTML!" message (**Figure 1.2**). CSS can be validated in the same way at <http://jigsaw.w3.org/css-validator>.

FIGURE 1.2 If your site complies with Web standards, you'll get the ever-gratifying This Page Is Valid XHTML message from the W3C validator.





For a list of deprecated tags that you should abandon and replace with their XHTML equivalents, refer to the About.com Web site (<http://Webdesign.about.com/library/tags/bltags-deprecatedtags.htm>).

Here's the complete (and mercifully, short) list of the coding requirements for XHTML compliance:

1. **Declare a DOCTYPE.** The DOCTYPE goes before the opening `html` tag at the top of the page and tells the browser whether the page contains HTML, XHTML, or a mix of both, so that it can correctly interpret the markup. There are three main DOCTYPEs that let the browser know what kind of markup it is dealing with:

**Strict:** All markup is XHTML compliant.

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

**Transitional:** This states that markup is a mix of XHTML and deprecated HTML. Many sites are currently using this one, so their old HTML code works as well (in this context, “as well” means “also” rather than “equally well”) as the XHTML they are now adding.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
```

**Frameset:** This is the same as transitional but in this case frames, which are deprecated under XHTML, are OK too.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
    "http://www.w3.org/TR/html4/frameset.dtd">
```

It is important to specify a DOCTYPE. Browsers that don't see a DOCTYPE in the markup assume that the site page was coded for browsers developed long before Web standards.

Without a DOCTYPE, many browsers go into what is known as *Quirks mode*, a backwards-compatibility feature supported by Mozilla, Internet Explorer 6 for Windows, and Internet Explorer 5 for Macintosh. In Quirks mode, the browser functions as if it has no knowledge of the modern DOM (document object model), and pretends it has never heard of Web standards. This ability to switch modes depending on the DOCTYPE, or lack thereof, enables browsers to do the best possible job of interpreting the code of both standards-compliant and non-compliant sites.

Note that for some weird reason, the `DOCTYPE` tag does not need to be closed with a backslash and DOCTYPE is always in caps. This entirely contradicts XHTML rules 4 and 7 below. Go figure.



You can learn more about Quirks mode at the Dive Into Mark Web site ([http://diveintomark.org/archives/2002/05/29/quirks\\_mode](http://diveintomark.org/archives/2002/05/29/quirks_mode)).



Because the DOCTYPE (and the XML namespace and content type discussed in items 2 and 3) are a pain to type, they are in the page templates on the Stylin' Web site ([www.bbd.com/stylin](http://www.bbd.com/stylin)). You can use these as a starting point for your own XHTML documents. Just pick whichever of the three (strict, transitional, or frames) DOCTYPEs you want to use.



If you copy a DOCTYPE or namespace from some other site, make sure the URL is absolute (that is, it starts with `http://` followed by a complete path to the document). Some sites (including W3C, of course) host their own DOCTYPE and namespace files, and so they can use relative URLs to them. But if you use these URLs as is, with a different server that doesn't host these files, your page may behave unpredictably because the URLs aren't pointing at anything.

2. **Declare an XML namespace.** Note this line is your new `html` tag. Here's an example:

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

When a browser is handling an XHTML page and wants to know what's in the DTD, which lists and defines all the valid XHTML tags, here's where it can find it—buried away on the servers of the WC3.

In short, the DOCTYPE and namespace declarations ensure that the browser interprets your XHTML code as you intended.

3. **Declare your content type.** The content type declaration goes in the head of your document, along with any other meta tags you may add. The most common is

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

This simply states what character coding was used for the document. ISO-8859-1 is the Latin character set, used by all standard flavors of English, so if you are coding for an audience who uses the alphabet instead of, for example, Chinese or Farsi characters, this is the one you need. If your next site is going to be in Cyrillic or Hebrew, you can find the appropriate content types on Microsoft's site (<http://msdn.microsoft.com/workshop/author/dhtml/reference/charsets/charset4.asp>).

4. **Close every tag, whether enclosing or nonenclosing.**

Enclosing tags have content within them, like this

```
<p>This is a paragraph of text inside paragraph tags. To be XHTML compliant, it must, and in this case does, have a closing tag.</p>
```

Non-enclosing tags do not go around content but still must be closed, using space-slash at the end, like this

```

```

5. **All tags must be nested correctly.** If a tag opens before a preceding one closes, it must be closed before that preceding one closes. For example

```
<p>It's <strong>very important</strong> to nest tags correctly.</p>
```

Here, the `strong` tag is correctly placed inside the `<p>`; it closes before the containing `p` tag is closed. A tag enclosed inside another in this way is said to be *nested*.

This is wrongly nested

```
<p>The nesting of these tags is <strong>wrong</p></strong>
```

Multiple elements can be nested inside a containing element; a list nests multiple `li` elements inside a single `ul` or `ol` element, like this

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Because CSS relies on proper nesting in order to target styles to elements, you have to get this right or your code won't validate.

- 6. Inline tags can't contain block level tags.** Block-level tags are tags that provide visual structure to your document, such as `p` (paragraph) and `div` (division). Block-level elements stack on top of one another on the page—if you have two paragraphs, the second paragraph appears by default under the previous one; no line breaks are required. By contrast, inline tags, such as `a` (anchor, a hyperlink) and `em` (emphasis, usually displayed as italics) occur in the normal flow of text, and don't force a new line.

We discuss block and inline elements in detail later in Chapter 4, but for now, just remember that if you nest a block element, such as a paragraph `p`, inside an inline element, such as a link `a`, your code won't validate.

Also, some block-level elements can't contain other block-level elements either; for instance, a `h1-6` (heading) tag can't contain a paragraph. Besides using validation, you can let common sense be your guide to avoid these problems; you wouldn't put an entire paragraph inside a paragraph heading when you were writing on paper or in Word, so don't do illogical things like that in your XHTML either, and you won't go far wrong.

- 7. Write tags entirely in lowercase.** Self-explanatory—no capital letters at all. I've always done this myself, but if you haven't, the days of `P` are over; now it has to be `p`. Sorry.

- 8. Attributes must have values and must be quoted.** Some tags' attributes don't need values in HTML, but in XHTML, all attributes must have values. For example, if you previously used the `select` tag to create a drop-down menu in HTML and wanted to choose which item showed by default when the page loaded, you might have written something like this

```
<SELECT NAME=ANIMALS>
<OPTION VALUE=Cats>Cats</OPTION>
<OPTION VALUE=Dogs SELECTED>Dogs</OPTION>
</SELECT>
```

which would have given you a drop-down menu with Dogs displayed by default.

The equivalent valid XHTML is this

```
<select name="animals">
<option value="cats">Cats</option>
<option value="dogs" selected="selected">Dogs</option>
</select>
```

Note that in this revised version, all the attribute names are in lowercase and all the values are in quotes.



*Quoted attribute values don't have to be lowercase, but it's good practice to write everything lower case; then you can't go wrong.*

### What Are Attributes?

*Attributes* can be added to a tag and can help further define that tag. Each attribute comprises two pieces: the *attribute name* and the *attribute value*, in the format `name="value"`. For example, this image tag

```

```

has two attributes: the image source (its relative location on the server), which has the value `"images/myPicture.jpg"`, and an alternative text description whose value is a string of text that appears onscreen if the image fails to load, or could be read by a screen reader.

Before Web standards, it was common practice to load up tags with presentational attributes. Now we can move all presentational information into the style sheet and thereby greatly reduce the complexity of our markup.



*Various tools take your old HTML markup and convert it to XHTML. Of these, HTML Tidy is considered the best. The Infohound site (<http://infohound.net/tidy>) has an online version of HTML Tidy and links to downloadable versions and documentation. After the conversion is complete, you always have some final hand cleanup to do, but HTML Tidy and others can save you hours of work.*

- 9. Use the encoded equivalents for a left angle bracket and ampersand within content.** When XHTML encounters a left angle bracket, `<` (also known as the less-than symbol), it quite reasonably assumes you are starting a tag. But what if you actually want that symbol to appear in your content? The answer is to encode it using an entity. An *entity* is a short string of characters that represents a single character; when used, this string causes XHTML to interpret and display the character correctly and not to confuse it with markup. The entity for the left angle-bracket/less-than symbol is `&lt;`;—remember LT stands for less than.

Entities not only help avoid parsing errors like the one just mentioned, but they also enable certain symbols to be displayed at all, such as `&copy;` for the copyright symbol (©). Every symbolic entity begins with an ampersand (&) and ends with a semicolon (;). Because of this, you probably aren't surprised to find out that XHTML regards ampersands in your code as the start of entities, and so you must also encode ampersands as entities when you want them to appear in your content; the ampersand entity is `&amp;`.

A good rule of thumb is that if a character you want to use is not printed on the keys of your keyboard (such as é, ®, ©, or £), you need to use an entity in your markup.

There are some 50,000 entities total, which encompass the character sets of most of the world's major languages, but you can find a shorter list of the commonly used entities at the Web Design Group site ([www.htmlhelp.com/reference/html40/entities](http://www.htmlhelp.com/reference/html40/entities)).

And those are the rules of XHTML markup; they are relatively simple, but you must follow them exactly if you want your pages to validate (and you do).

## Understanding Markup

Here is a sample unstyled but valid XHTML page that illustrates the rules of XHTML (**Figure 1.3**):

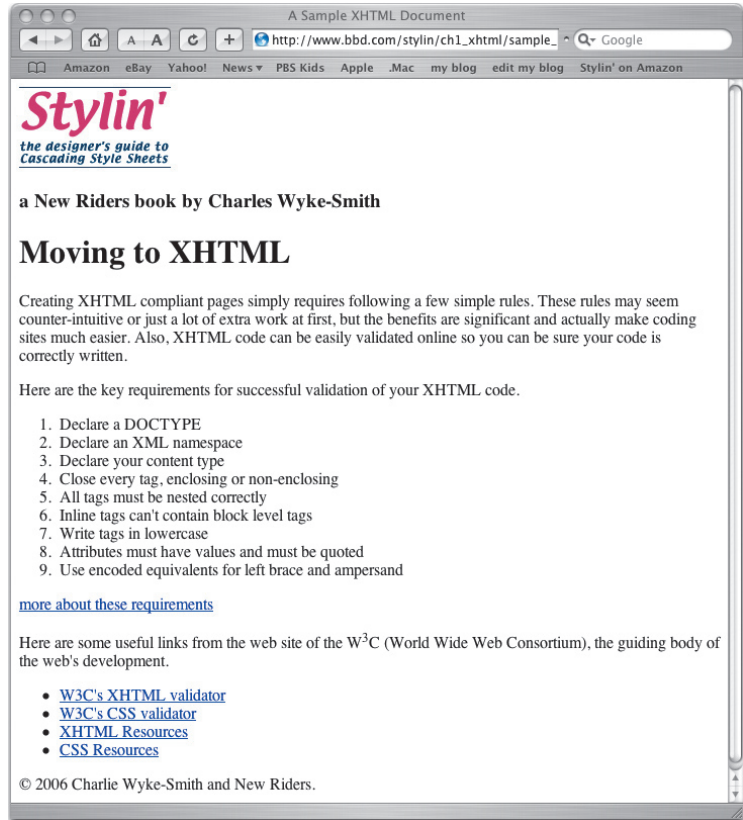
FIGURE 1.3 This unstyled but valid XHTML isn't visually interesting, but it is definitely usable.



*The minimal styling that you do see in the page such as the different sizes of header text, the bulleted lists, links, and so on is due to the fact that the browser has an internal style sheet that styles valid XHTML markup, and the only reason for you to style it differently is if you want it to look different.*



*The browser's default styles are the baseline design for your markup; if the browser doesn't read your style sheet for some reason, these default styles are your fall back, so it's worth making sure that your marked-up but unstyled page displays meaningfully in the browser before you start on the CSS—if it's valid and semantic XHTML, it will.*



The page isn't pretty, but it is certainly usable. And, this page's markup is lean and simple. There is no presentational code and this XHTML passes muster with the W3C HTML validator. In Chapter 3, I'll begin teaching you how to turn this unstyled markup into a more attractive-looking page using CSS.

Now let's get into more detail on the XHTML rules by taking a look at the markup that created the page shown in Figure 1.3 line by line.

### LINES 1 - 2

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

Here the DOCTYPE is set to XHTML 1.0 Strict. In this case, you're indicating that code will be interpreted as pure, non-backward-compatible XHTML.

I focus on the strict DOCTYPEs throughout this book, which means I do not use any deprecated HTML. If you need to support deprecated HTML tags such as frames, you need a different DOCTYPE (see "XHTML Markup Rules," #3 earlier in this chapter).

#### LINE 3

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

Next is the opening `html` tag, which did not have attributes in the past. Now it has a URL that points to the *namespace* (the collection of XML declarations and attributes) of this document.

As mentioned earlier in "XHTML Markup Rules", the DOCTYPE and namespace declarations ensure that the browser understands what flavor of (X)HTML you are using, so it interprets your code as you intended.

#### LINE 4

```
<head>
```

This tag opens the document head. The head of your document, which is sandwiched between the `head` and `/head` tags), contains information that, with the exception of the title, is not displayed to the viewer. Besides the essential `head` tags I list next (Lines 5–9), optionally there can be others: `meta` tags can contain all kinds of information (page descriptions, keywords, author names, etc.) used by search engines and other indexing software that might visit your site.

There can also be style tags that contain JavaScript and CSS that relate to, and can only be used by, the page they are on.

#### LINE 5

```
<title>A Sample XHTML Document</title>
```

Technically, you don't *have* to use a `title` tag for your page to validate, but if you don't add it, the validator will encourage you to add it, and after you read the "About Title Tags" sidebar, you always will.



Learn more about metatags by visiting the [Webdeveloper.com](http://www.webdeveloper.com/html/html_metatags.html) site ([www.webdeveloper.com/html/html\\_metatags.html](http://www.webdeveloper.com/html/html_metatags.html)).

## About Title Tags

It's easy to miss the title of a page because it is displayed at the very top of the browser window, but title tags carry tremendous weight with search engines—for example, the pages that get listed on page one of Google's results almost always have some or all of your search terms present in their titles, which are also displayed as the titles of each of the results. So make sure your page title contains keywords that your users might use to search with and is written so it entices clicks when it appears in results. Don't waste your title tag with the useless and all-too common "Welcome to our Home Page."

### LINES 6 - 7

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

```
<meta http-equiv="Content-Language" content="en-us" />
```

These two required `meta` head tags provide information that helps the browser and server properly manage and display the page.

XHTML insists that you provide character encoding information, which ensures that the browser is displaying the pages with an appropriate character set. Here, in the first `meta` tag, 8859-1 is the code for Latin-1, the alphabet and associated symbols used in writing English and some other languages (see "XHTML Markup Rules," #3 earlier in this chapter). Note that as nonenclosing tags, they are both closed with the space-slash-angle bracket construction.

Language information is also required. In the second `meta` tag, I state that the language is U.S. English; a language type such as Chinese causes the browser to display text from right to left.

### LINES 8 - 9

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

The `link` tag links the XHTML markup to a CSS style sheet, which is a separate file located using the `href`. (I show you how to create a linked CSS style sheet later in this book so in this case the browser does not find the file and simply ignores this line.) The `link` tag isn't required, but linking is how you relate a style sheet to your markup, and by adding the same style sheet link to each page of your site, you can enable the pages to all share the same set of styles. You can also use the `@import` rule to link to a style sheet, and I'll show you

both of these linking methods and when you might use one or the other or both, later in the book.

Make sure you close the document head using the `/head` tag.

#### LINE 10

```
<body>
```

Start the document body. The body contains the content that displays on your page.

#### LINE 11

```
<!--header-->
```

This is a comment. It is not displayed; it is just here to make the code more understandable. Note that in XHTML you can only use two dashes, instead of the unlimited number allowed by HTML, at the start and end of each comment.

#### LINE 12

```
<div id="logo"> 
```

### Comments on Comments

The browser does not display comments, but the visitor can see them if he views the source code of your page. I comment my code heavily, especially with what I call “start comments” that show me where each section of my document starts. Doing so helps me find my way around long documents and helps others understand my page if they have to edit it later.

Often, I write comments first to get an overview of my page structure; I then start adding the tags between them. Remember, comments are your friend and, while they may add a few minutes to the initial coding, they can save you hours later.

Note in the code samples that I also write a corresponding end comment for each start comment. Sometimes the opening and closing `<div>` tags that are often used to define a content block can be hundreds of lines apart and can have other divs nested close by inside them—it gets confusing fast. If you comment the beginning and end of any div that is likely to have more than a few lines of code in it, you can quickly and confidently make major edits as you organize your markup. If you don't get the value of this yet, keep it in mind as we proceed—I'll show you plenty of examples.

Divs divide the page into rectangular, box-like areas. These areas are invisible unless you turn their borders on or color their backgrounds. This `div` tag has an `id` attribute with the value of "logo"; you can use this ID name to target CSS styles at this div to set its position, size, background color, and much more; furthermore, the div allows you to position all the content within it as a group and use its ID to target styles at each of the tags it contains.

The logo image tag (`img`) is a nonenclosing element and is therefore closed with a slash before the closing brace. Note the `alt` tag, which displays if the graphic doesn't load or, can be spoken by a screen reader. You must use `alt` tags on every image, even if the value is "" (that is, two quotes with nothing, not even a space, in between). Only do this if the image serves no informational purpose. You can leave the `alt` tags blank on everything, but such tags will be flagged by an XHTML validator. Also, this isn't very user friendly and does not aid accessibility. Note that all attribute values (such as the 150 and 80 in this example) must now be in quotes. Yes, really.

### Naming Classes and IDs

IDs and class attributes are identifiers you can add to your tags. You can add a class or an ID attribute to any tag, although most commonly, you add them to block-level elements. IDs and classes help you accurately target your CSS at a specific element or set of elements. I get into the uses for (and differences between) IDs and classes later, but for now, it's helpful to know that the attribute value must be a single word, although you can make compound words that the browser sees as single words using underscores, such as `class="navigation_links"`.

Because the browser can misinterpret attribute names made of bizarre strings of characters, my advice is to start the word with a letter, not a number or a symbol. Because the only purpose of a class or ID is to give an element a name that you can reference in your style sheet (or JavaScript code), the value can be a word of your own choosing. That said, it's good practice to name classes and IDs something meaningful such as `class="navigationbar"` rather than `class="deadrat"`. Although the deadrat class might provide a moment of levity during a grueling programming session, the humor may be lost on you when you are editing your code at some point in the future. Don't save time with abbreviated names either; call the class "footer" rather than "fr" or you are apt to waste your time (or someone else's) later trying to figure out what you meant. Do yourself a favor and take the time to give classes and IDs nonambiguous and descriptive names.

**LINES 13 - 15**

```
<h3>a New Riders book by Charles Wyke-Smith</h3>
</div>
<!--end header-->
```

A size 3 text heading is a block-level element and therefore it occurs on a new line, or more precisely, under the previous element. No `br /` tags are required.

```
</div>
```

Remember to close the header division using the `/div` tag and make a comment that the header ends here.

**LINES 16 - 20**

```
<!--main content-->
<div class="contentarea">
  <h1>Moving to XHTML</h1>
  <p>Creating XHTML compliant pages simply requires following
  a few simple rules. These rules may seem counter-intuitive
  or just a lot of extra work at first, but the benefits are
  significant and actually make coding sites much easier. Also,
  XHTML code can be easily validated online, so you can be
  sure your code is correctly written.</p>
  <p>Here are the key requirements for successful validation of
  your XHTML code.</p>
```

The content area starts with a `div`, which is a block-level element. The main header is size 1 text. Next, are two paragraphs. Paragraph tags, like all enclosing tags, must be closed with a forwardslash tag; in this case, `/p`. Note that paragraphs are block-level elements and have a default amount of space around them, top and bottom.

**LINES 21 - 31**

```
<ol>
  <li>Declare a DOCTYPE.</li>
  <li>Declare an XML namespace.</li>
  <li>Declare your content type.</li>
  <li>Close every tag, enclosing or non-enclosing.</li>
```

```
<li>All tags must be nested correctly.</li>
<li>Inline tags can't contain block-level tags.</li>
<li>Write tags in lowercase.</li>
<li>Attributes must have values and must be quoted.</li>
<li>Use encoded equivalents for left brace and
ampersand.</li>
</ol>
```

This is an ordered list; each list item has a number by default. (Unordered lists ([ul](#)) have bullets by default rather than numbers).

#### LINE 32

```
<a href="more.htm">more about these requirements</a>
```

This is a hyperlink to a page named `more.htm` in the same folder as the current page.

#### LINES 33 - 34

```
</div>
<!--end main content-->
```

This closes the content area `div`. The comment is, of course, optional.

#### LINES 35 - 37

```
<!--navigation-->
<div id="navigation">
  <p>Here are some useful links from the web site of the
  <acronym title="World Wide Web Consortium">W3C</acronym>
  (World Wide Web Consortium), the guiding body of the web's
  development.</p>
```

It's good practice to style acronyms in a way that differentiates them from the text around them. Internet Explorer does not provide any default styling for acronyms; Safari will put them in italics (such as in Figure 1.3). If you add a `title` tag to an acronym, a tool tip containing the text from the title attribute pops up when a user mouses over it. It's also good practice to indicate the tool-tip's availability by underlining the acronym with a dotted line; this is achieved by styling the acronym element with a dotted border-bottom. Don't make the underline solid, which by convention would indicate the text is a link. These same markup techniques can also be applied to the `abbr` (abbreviation) tag.

**LINES 38 - 45**

```

    <ul>
      <li><a href="http://validator.w3.org">W3C's XHTML
validator</a></li>
      <li><a href="http://jigsaw.w3.org/css-validator/">W3C's
CSS validator</a></li>
      <li><a href="http://www.w3.org/MarkUp/">XHTML Resources</
a></li>
      <li><a href="http://www.w3.org/Style/CSS/">CSS
Resources</a></li>
    </ul>
</div>
<!--end navigation-->

```

This navigation aid is constructed as a list in which each list item is a link. All of this is inside a `div` block with an ID that enables you to reference it accurately from the style sheet. Note that there is no line break (which here would be purely presentational markup) at the end of each link; none is needed. By default, links appear in a row because they are inline elements, but here, because they are contained within list items, which are block-level elements, they display stacked.

**LINES 46 - 50**

```

<!--footer-->
<div id="homepagefooter">
  <p>&copy; 2004 Charlie Wyke-Smith and New Riders.</p>
</div>
<!--end footer-->

```

The last element of the page is a `div` that contains the footer text inside a paragraph tag.

**LINES 51 - 53**

```

</body>
</html>
<!--end of sample doc-->

```

Now you just close out the body and the page, and you're done. Any questions? No? Good! Moving right along . . .

## Document Hierarchy: Meet the XHTML Family

OK, the *document hierarchy* is one more important concept you need to understand before you can get to CSS. The document hierarchy is like a family tree or an organizational chart based on the nesting of a page's XHTML tags. A good way to learn to understand this concept is to take a snip of the body section of the markup we just discussed and strip out the content so that you can better see the organization of the tags. Here's the stripped-down header

```
<body>

    <!--header - this is just a comment, not code-->

    <div id="logo">
        <img />
        <h3> </h3>
    </div>

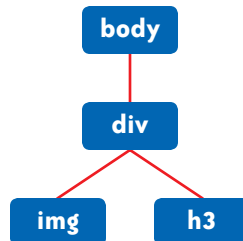
    <!--end header - remaining tags removed here for
    clarity-->

</body>
```

Now you can clearly see the relationships of the tags; for example, in the markup, you can see that the `body` tag contains (or nests) all the other tags. You can also see that the `div` tag (with the ID of "logo") contains two tags; an image tag and head 3 tag.

**Figure 1.4** shows another way to represent this structure—with a hierarchy diagram.

**Figure 1.4** You can clearly see the hierarchical structure in this diagram.



## XHTML Starter Kit

If this has whetted your appetite and you want jump right into coding standards-compliant sites, here's the framework of an XHTML page (this example is Strict, meaning it contains no frames or deprecated tags), and you can download this template, or a transitional version, at [www.bbd.com/stylin](http://www.bbd.com/stylin). For more HTML and XHTML templates, see The Web Standards site ([www.webstandards.org/learn/templates/index.html](http://www.webstandards.org/learn/templates/index.html)).

You just need to change the text in the `title` tag to the title of your page and start adding your own XHTML inside the `body` tag.

```
<!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" xml:lang="en">
  <head>
    <title>A Sample XHTML Document</title>
    <meta http-equiv="Content-type" content="text/html; charset=iso-8859-1" />
    <meta http-equiv="Content-Language" content="en-us" />
    <link href="demo_styles.css" rel="stylesheet" type="text/css" />
  </head>

  <body>
    <!--YOUR XHTML IN HERE-->
  </body>
</html>
```

When examining this hierarchical view, we can say that both the `img` tag and the `h3` tag are the *children* of the `div` tag, because it is the containing element of both. In turn, the `div` tag is the *parent* tag of both of them, and the `img` tag and the `h3` tag are *siblings* of one another because they both have the same parent tag. Finally, the `body` tag is an *ancestor* tag of the `img` and `h3` tags, because they are indirectly descended from it. In the same way, the `img` and `h3` tags (and the `div`, for that matter) are *descendants* of the `body` tag. To quote Sly Stone: "It's a family affair . . ."

In CSS, you write a kind of shorthand based on these relationships; for example

```
div#logo img {some CSS styling in here}
```

Such a CSS rule only targets `img` tags inside of (descended from) the `div` with the ID of "logo" (the `#` is the CSS symbol for an ID). This rule means "any image that is descended from the `div` with an ID of "logo"; other `img` tags in the page are unaffected by this rule because they aren't contained within the "logo" `div`. In this way, you can add a border around just this image or set its margin to move it away from surrounding elements.

We will get into learning to write CSS rules like this in great detail in the next chapter, but the important concept to understand is that every element within the body of your document is a descendant of the `body` tag, and, depending on its location in the markup, the element could be an ancestor, a parent, a child, or a sibling to other tags in the document hierarchy.

By creating rules that use (and often combine) references to IDs, classes, and the hierarchy structure, you have means by which you can accurately dictate which CSS rules affect which XHTML elements, and this is exactly what you will learn to do next.