CHAPTER 2

HTML AND XHTML BASICS

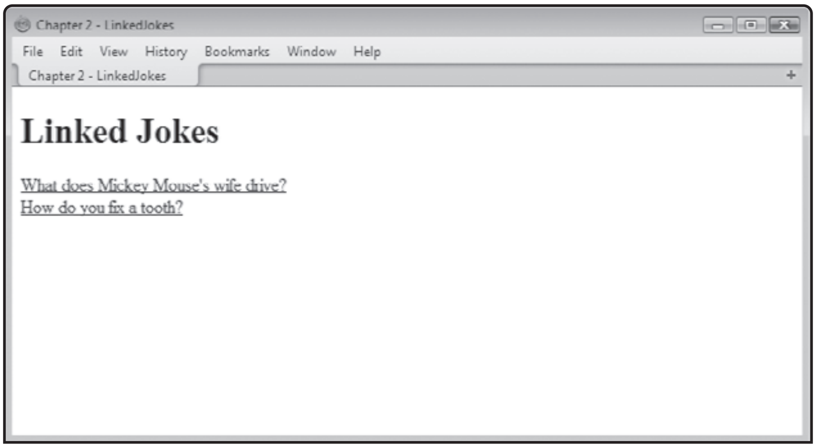
A key part of understanding how to develop web pages is to know the syntax rules and requirements of how to work with (X)HTML tags and their attributes. In this chapter, you will review the different HTML and XHTML standards available to web developers and will learn how to specify which version you plan to use in your web documents. You will learn the different ways in which elements are formulated and will review basic tag construction. You will also learn how to configure tag attributes and to work with standard tag attributes. This chapter will also explain how to improve the organization and presentation of statements in your web pages and will provide advice on what to look for when selecting a web host for your website.

Specifically, you will learn:

- About the six versions of (X)HTML and how to specify which version you are working with using the `html` element
- How to work with single tags and tag pairs
- How to validate your (X)HTML documents
- How to modify element attributes
- How to comment your (X)HTML markup and to make effective use of white space

Project Preview: Linked Jokes Application

In this chapter's web development project, you will learn how to create a new XHTML application named linkedjokes.html. This web application will consist of three web documents. The first document displays a pair of jokes, as shown in Figure 2.1.



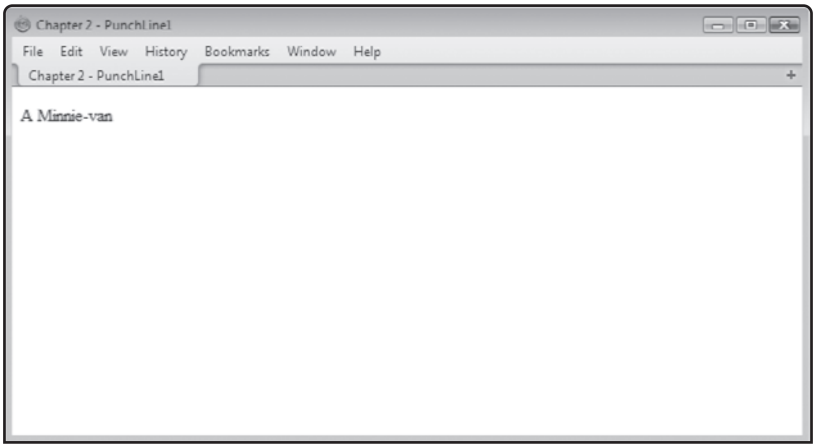
Linked Jokes

What does Mickey Mouse's wife drive?
How do you fix a tooth?

Figure 2.1

Using the Apple Safari browser to load the linkedjokes.html page.

Answers to each joke are provided in two separate web documents, one per joke. To view the answer for a given joke, the user must click on that joke's link. Figure 2.2 shows the web page that is loaded when the user clicks on the link for the first joke.

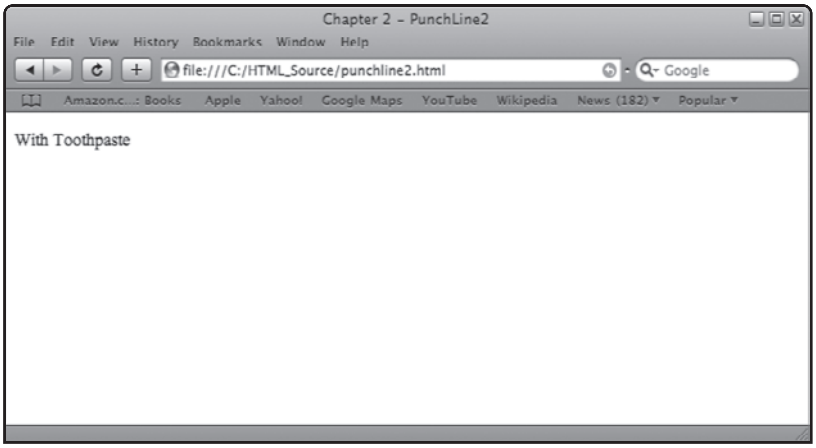


A Minnie-van

Figure 2.2

This page is automatically loaded when the user clicks on the first link on the page.

Likewise, to view the answer for the second joke, the user must click on that joke's link. Figure 2.3 shows the punch line that is displayed when the link for the second joke has been clicked.


Chapter 2 - PunchLine2
File Edit View History Bookmarks Window Help
file:///C:/HTML_Source/punchline2.html
Google
Amazon.c...: Books Apple Yahoo! Google Maps YouTube Wikipedia News (182) Popular

With Toothpaste

FIGURE 2.3

This page is automatically loaded when the user clicks on the second link on the page.

SEPARATING PRESENTATION FROM CONTENT

The basic purpose of (X)HTML is to provide structure to web documents. It provides a collection of elements that allow web developers to organize web documents in meaningful ways, identifying things like headings, paragraphs, and so on.

In the old days, before the widespread availability of CSS, web developers had limited control over the presentation of text within their web pages. To cope with this problem, a number of attributes were added to HTML that gave developers a bit of control over the appearance of a web document's content. However, this intermixing of content and presentation led to markup code that was not always easy to understand and support because of all of the extra presentation attributes that had to be added to document elements in order to try to manage their appearance.

Finally, along came Cascading Style Sheets or CSS in the late 1990s. CSS is its own language separate and distinct from (X)HTML. CSS allows web developers to define presentation rules in a stylesheet, which can then be applied to (X)HTML elements. Style sheets can be defined separately from markup inside web documents or externally in CSS files, separating content from presentation even further. Once created, external style sheets can be applied to any number of (X)HTML documents, allowing for the centralized administration of presentation for any number of web documents using a single style sheet.

Thanks to the widespread use of CSS, web developers can now separate content from presentation by using (X)HTML to outline a web document's design and organization and CSS to specify the appearance of that content. The result is web documents that are significantly easier to understand and update. The focus of this chapter is on the development of properly structured or *well-formed* (X)HTML. Concerns over presentation will be saved for later chapters.

The Six Flavors of (X)HTML

In total there are currently six different versions of HTML and XHTML, referred to collectively throughout this book as (X)HTML. There are three current versions of HTML named Transitional, Frameset, and Strict and three roughly parallel versions of XHTML also named Transitional, Frameset, and Strict. You specify which version of HTML or XHTML you are working with in a special element located at the beginning of every (X)HTML page known as the *Document Type Declaration* or *DOCTYPE* element. The `DOCTYPE` element tells web browsers what version of (X)HTML is being used so that the browser knows what set of rules to follow when rendering and displaying the document's content.

Technically, the `DOCTYPE` declaration is not an HTML element. Its sole purpose is to indicate a document's type. It must be placed at the top of all (X)HTML documents and must be defined exactly as outlined in the sections that follow, with no variation in syntax or capitalization. Other than white space, no other statements may precede the DOCTYPE declaration.

TRAP

Technically speaking, to be compliant with XML standards, all XML documents should include an XML declaration statement before the `DOCTYPE` declaration. The XML declaration statement is used to identify the document as an XML document and to specify the document character encoding method. An example of an XML declaration statement is shown here:

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

Unfortunately, not all web browsers, most notably Internet Explorer, are able to properly interpret the XML declaration statement. As a result, these browsers may render web pages inconsistent with your expectations. Because of limited browser support, it is best to omit the XML declaration statement.

HTML Standards

As already stated, there are three different current versions of HTML, each of which defines a similar but slightly different standard that specifies the rules that must be followed in order to generate well-formed documents.

HTML 4.01 Transitional

HTML 4.01 Transitional supports all HTML elements, including a number of presentation elements. Its purpose is to help web developers make the transition from earlier versions of HTML to HTML 4.01. To work with this version of HTML, you must add the following `DOCTYPE` element to be beginning of your HTML pages, typed exactly as shown here:

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

HTML 4.01 Frameset

HTML 4.01 Frameset is identical to HTML 4.01 Transition but includes added support for dealing with frames. Frames are an older web development methodology in which web pages were organized into different sections or frames, into which separate HTML pages are loaded. To work with this version of HTML, you must add the following `DOCTYPE` element to the beginning of your HTML pages, exactly as shown here:

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

HTML 4.01 Strict

HTML 4.01 Strict excludes support for older presentation-based HTML elements, deferring to CSS to provide for web page presentation. Well-formed HTML pages that use this version will display more consistent results when rendered by different web browsers. To work with this version of HTML, you must add the following `DOCTYPE` element to the beginning of your HTML pages, typed exactly as shown here:

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

XHTML Standards

As is the case with HTML, there are also three different current versions of XHTML. All three are similar but vary slightly in regards to the rules that must be followed in order to generate well-formed documents.

XHTML 1.0 Transitional

XHTML 1.0 Transitional, as its name implies, is a version of XHTML designed to support web developers who are in the process of converting from HTML to XHTML. As such, it retains support for a number of deprecated features, which, if present do not prevent a document from being well formed. To work with this version of XHTML, you must add the following `DOCTYPE` element to the beginning of your HTML pages, exactly as shown here:

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

XHTML 1.0 Frameset

XHTML 1.0 Frameset is designed to support web pages that still rely on the use of framesets. A frameset is a mechanism for laying out web pages into separate frames or panes, each of which displays its own web page. Frames are a deprecated feature in both Strict and Transitional XHTML. To work with this version of XHTML, you must add the following `DOCTYPE` element to the beginning of your HTML pages, exactly as shown here:

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

XHTML 1.0 Strict

XHTML 1.0 Strict, as its name implies, is the most stringent of the three versions of XHTML. As such, presentation and other deprecated features are not allowed and syntax rules must be rigidly adhered to in order for an XHTML document to be regarded as being well formed. To work with this version of XHTML, you must add the following `DOCTYPE` element to the beginning of your HTML pages, exactly as shown here:

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

Unless otherwise specifically noted, all of the examples that are presented in this book will be done using XHTML 1.0 Strict.

TRAP

Back in the early days of CSS, web browsers did not provide uniform levels of support for the language. Browsers of the day often interpreted CSS in accordance with their own rules in place of the official standard. To overcome this challenge, browser developers introduced DOCTYPE switching. With DOCTYPE switching, browsers assume that any document with a properly defined `DOCTYPE` is well formed and exactly follows the standard applications to its definition and will render the page in compliance mode.

If, however, the `DOCTYPE` is not properly defined or present, the browser will render the document in quirks mode. *Quirks mode* is more lenient than compliance mode and may result in less than desirable presentation of the document. In contrast, documents rendered in compliance mode are rendered in a far more predictable manner.

The html Element

The `html` element marks the beginning of a document's markup and is referred to as the document's root element. In (X)HTML documents the `html` element is defined by an opening `<html>` tag and a corresponding closing `</html>` tag. In HTML documents the `html` element is used without any attributes, as demonstrated here:

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

  <body>
  </body>
 </html>
```

In XHTML documents, you must include a required `xmlns` attribute in the `head` element and assign it a value of http://www.w3.org/1999/xhtml. This attribute specifies the location where the XHTML namespace resides. This namespace defines all of the elements and attributes supported by XHTML.

In addition, you should specify the optional `lang` and `xml:lang` attributes. These attributes specify the language in which the web document has been written. The following example demonstrates how the `html` element will appear in all of the XHTML examples that you will see in this book.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
  <head>
  </head>

  <body>
  </body>
 </html>
```

HINT

XHTML is based on XML. XML is an extensible markup language. It allows for the creation of custom elements. However, XHTML 1.0 only supports a predefined collection of elements as specified in its namespace. XHTML 1.0 does not support the definition of custom elements. XHTML 1.1 and XHTML 2 will allow web developers to introduce custom elements through the development of a customized namespace. However, neither of these versions of XHTML has been published as an official standard yet.

Dissecting (X)HTML Markup

(X)HTML consists of numerous elements that you must learn how to use as a web developer. (X)HTML elements are made up of tags. Tags are used to mark the beginning and end of document content. Tag names are descriptive. They instruct web browsers as to the type of content they contain, so that the browsers will know how to render the document's content.

(X)HTML tags are enclosed within `<` and `>` brackets. The opening bracket (`<`) identifies the beginning of the tag. It is followed by the tag name. Tag names end with a closing bracket (`>`). (X)HTML consists of many different tags, each of which serves a different and distinct purpose. For example, the `<p>` tag specifies the beginning of a paragraph and the `<h1>` tag specifies the beginning of a heading.

Tag Pairs

Most (X)HTML tags work in pairs, including a start and an end tag. Tag pairs have the following syntax.

```
<tag>content</tag>
```

Here, ***content*** represents the content that is embedded within the two tags. The start tag identifies the beginning of an element and the end tag identifies where the element ends. For example, the following elements are all made up of tag pairs.

```
<h1>Little Red Riding Hood</h1>
<p>There once was a little girl named Little Red Riding Hood.</p>
```

Here, the first pair of tags defines a level one heading. It begins with the `<h1>` start tag and ends with the `</h1>` tag. The second pair defines a paragraph. It begins with the `<p>` tag and ends with the `</p>` tag. As you can see, all end tags include a / character, which sets them apart from start tags. These two tags and everything in between them is an element. Elements form the building blocks with which (X)HTML documents are created.

As already stated, (X)HTML supports two primary types of tags: tag pairs and single tags. A pair tag is a set of two tags that identify the beginning and the ending of an element. Figure 2.4 provides a high-level breakdown on the components of an (X)HTML element tag pair.

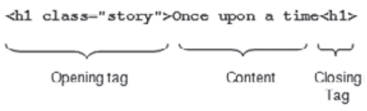


FIGURE 2.4

Elements are composed of an opening tag, content, and a closing tag.

As depicted in Figure 2.5, elements also consist of attributes.

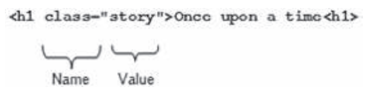


FIGURE 2.5

An attribute is made up of a name and an assigned value.

Single Tags

Not all elements require closing tags. Elements that do not have an end tag are referred to as single or empty tags. Single tags do not contain any contents. While HTML will let you get away without supplying an end tag in many situations, XTHML mandates that all elements must be closed. To comply with this rule, all single tags are self-closed, which is accomplished by placing a / before the closing >, as demonstrated here:

```
<tag/>
```

Unfortunately, since XHTML is based on XML and older browsers do not support XHTML, problems arise when these browsers attempt to process XHTML pages within single tags. To prevent these browsers from running into trouble when they try to render your web pages, you need to include a blank space in front of the closing / character, as demonstrated here:

```
<tag />
```

This little trick allows older browsers to ignore the closing slash, since it's not supported. Newer browsers, on the other hand, are smart enough to ignore the extra space and correctly interpret the tag. Three examples of how you should formulate single tags are provided here:

```
<br />
<hr />
<img src= "logo.gif" />
```

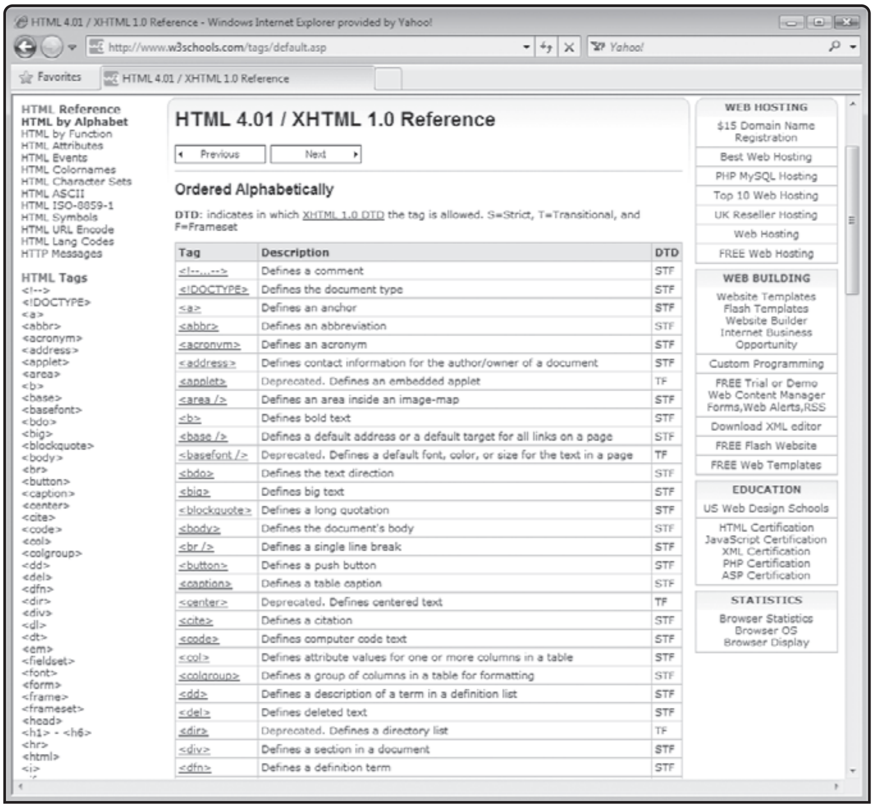
The first tag is used to insert a line break. The second tag inserts a horizontal rule (a line) across a web page, and the third example uses an image element (`<img />`) to insert an external image file into the web page at a specified location. Don't worry right now what each of these tags do. You learn about them in greater detail later in this book.

HINT This book teaches you how to work with both (X)HTML and CSS. One easy way to determine which type of content you are looking at is to look and see if code has been embedded within the < and > characters (e.g., (X)HTML) or within { and } characters (e.g., CSS).

Learning More about Tags

Throughout this book you will be introduced to different types of tags, all of which support a host of different attributes. While you will be introduced to many of the elements that support these tags, there is not enough room in this book to define and present every possible attribute belonging to every available (X)HTML tag. Instead, you'll be introduced to the most commonly used tags and the most commonly used attributes.

If you find that you need to know more about any of the tags or tag attributes covered in this book, you can visit http://www.w3schools.com/tags/default.asp, as shown in Figure 2.6.


HTML 4.01 / XHTML 1.0 Reference - Windows Internet Explorer provided by Yahoo!
http://www.w3schools.com/tags/default.asp
Yahoo!
Favorites
HTML 4.01 / XHTML 1.0 Reference
Previous
Next

HTML Reference
HTML by Alphabet
HTML by Function
HTML Attributes
HTML Events
HTML Colornames
HTML Character Sets
HTML ASCII
HTML ISO-8859-1
HTML Symbols
HTML URL Encode
HTML Lang Codes
HTTP Messages

HTML Tags
<!-->
<!DOCTYPE>
<a>
<abbr>
<acronym>
<address>
<applet>
<area>
<b>
<base>
<basefont>
<bdo>
<big>
<blockquote>
<body>

<button>
<caption>
<center>
<cite>
<code>
<col>
<colgroup>
<dd>
<del>
<dfn>
<dir>
<div>
<dl>
<dt>
<em>
<fieldset>
<font>
<form>
<frame>
<frameset>
<head>
<h1> - <h6>
<hr>
<html>
<i>

HTML 4.01 / XHTML 1.0 Reference

Ordered Alphabetically

DTD: indicates in which XHTML 1.0 DTD the tag is allowed. S=Strict, T=Transitional, and F=Frameset

Tag	Description	DTD
<!--...-->	Defines a comment	STF
<!DOCTYPE>	Defines the document type	STF
<a>	Defines an anchor	STF
<abbr>	Defines an abbreviation	STF
<acronym>	Defines an acronym	STF
<address>	Defines contact information for the author/owner of a document	STF
<applet>	Deprecated. Defines an embedded applet	TF
<area />	Defines an area inside an image-map	STF
<b>	Defines bold text	STF
<base />	Defines a default address or a default target for all links on a page	STF
<basefont />	Deprecated. Defines a default font, color, or size for the text in a page	TF
<bdo>	Defines the text direction	STF
<big>	Defines big text	STF
<blockquote>	Defines a long quotation	STF
<body>	Defines the document's body	STF
<br />	Defines a single line break	STF
<button>	Defines a push button	STF
<caption>	Defines a table caption	STF
<center>	Deprecated. Defines centered text	TF
<cite>	Defines a citation	STF
<code>	Defines computer code text	STF
<col>	Defines attribute values for one or more columns in a table	STF
<colgroup>	Defines a group of columns in a table for formatting	STF
<dd>	Defines a description of a term in a definition list	STF
<del>	Defines deleted text	STF
<dir>	Deprecated. Defines a directory list	TF
<div>	Defines a section in a document	STF
<dfn>	Defines a definition term	STF

WEB HOSTING
$15 Domain Name Registration
Best Web Hosting
PHP MySQL Hosting
Top 10 Web Hosting
UK Reseller Hosting
Web Hosting
FREE Web Hosting

WEB BUILDING
Website Templates
Flash Templates
Website Builder
Internet Business Opportunity
Custom Programming
FREE Trial or Demo
Web Content Manager
Forms,Web Alerts,RSS
Download XML editor
FREE Flash Website
FREE Web Templates

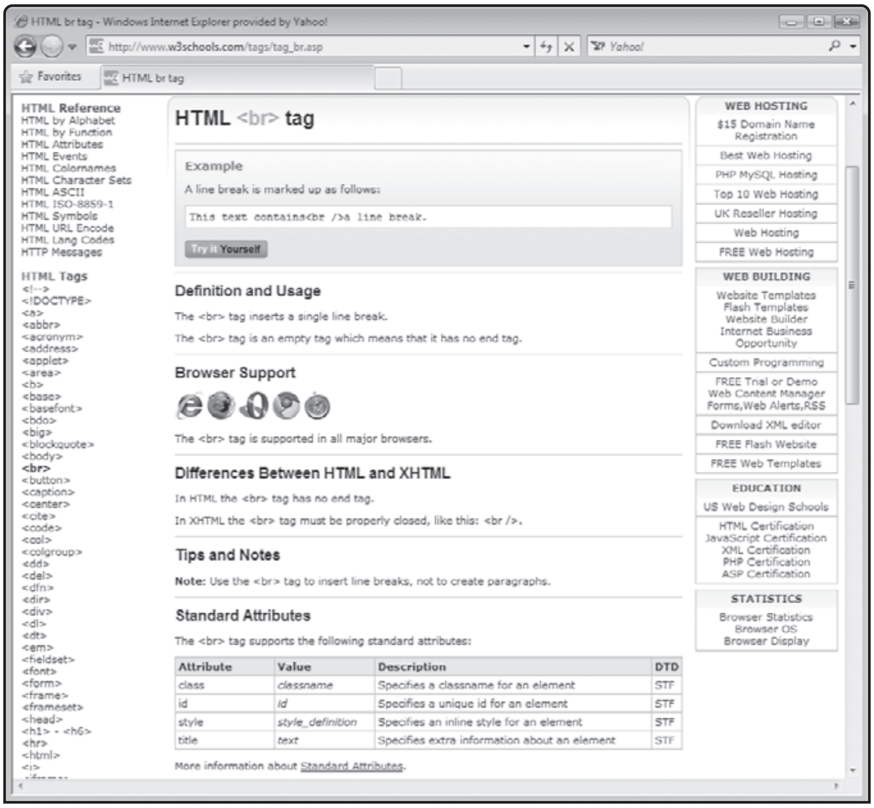
EDUCATION
US Web Design Schools
HTML Certification
JavaScript Certification
XML Certification
PHP Certification
ASP Certification

STATISTICS
Browser Statistics
Browser OS
Browser Display

FIGURE 2.6

Detailed information about every HTML and XTHML tag is available online.

As shown in Figure 2.6, the HTML 4.01 / XHTML 1.0 Reference at www.w3schools.com provides information about every available tag. To review a document for a given tag, all you have to do is click on its tag name and you will be presented with everything there is to know about the tag, as demonstrated in Figure 2.7.



HTML <br> tag

Example

A line break is marked up as follows:

```
This text contains<br />a line break.
```

Definition and Usage

The <br> tag inserts a single line break.

The <br> tag is an empty tag which means that it has no end tag.

Browser Support

The <br> tag is supported in all major browsers.

Differences Between HTML and XHTML

In HTML the <br> tag has no end tag.

In XHTML the <br> tag must be properly closed, like this: <br />.

Tips and Notes

Note: Use the <br> tag to insert line breaks, not to create paragraphs.

Standard Attributes

The <br> tag supports the following standard attributes:

Attribute	Value	Description	DTD
class	*classname*	Specifies a classname for an element	STF
id	*id*	Specifies a unique id for an element	STF
style	*style_definition*	Specifies an inline style for an element	STF
title	*text*	Specifies extra information about an element	STF

More information about Standard Attributes.

FIGURE 2.7

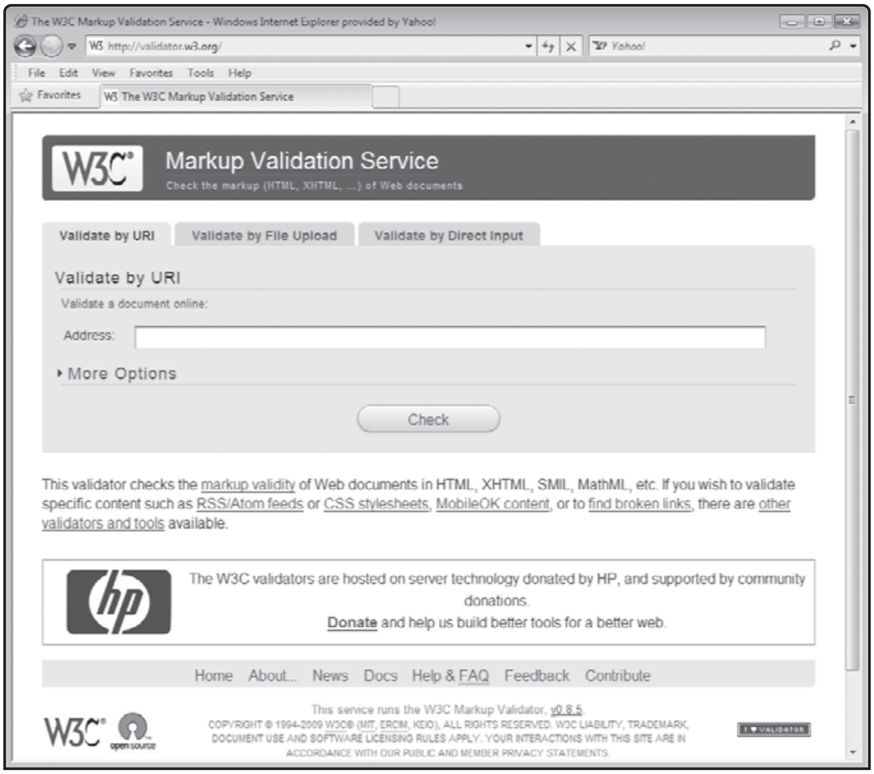
An example of the information that is returned when you look up the `<br>` tag.

As shown in Figure 2.7, the information provided about each tag includes an example of its usage, a list of browsers that support it, an explanation of differences in the way the tag is supported between HTML and XHTML, and a detailed listing of all the tag's attributes.

Markup Validation

In order to render (X)HTML documents in a consistent and predictable manner, you must ensure that they are well formed, meaning that your document should strictly adhere to whatever HTML or XHTML standard you have decided to work with. In this book, that's XHTML Strict. Failure to create well-formed documents will result in unpredictable results.

Beyond taking care when developing your (X)HTML documents, you can ensure that your documents are well formed by taking advantage of a free markup validation service provided by W3C, located at `validator.w3.org` as shown in Figure 2.8.


The W3C Markup Validation Service - Windows Internet Explorer provided by Yahoo!
http://validator.w3.org/
Yahoo!
File Edit View Favorites Tools Help
Favorites
The W3C Markup Validation Service
Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents
Validate by URI
Validate by File Upload
Validate by Direct Input
Validate by URI
Validate a document online:
Address:
More Options
Check
Home About... News Docs Help & FAQ Feedback Contribute
I ♥ VALIDATOR

This validator checks the markup validity of Web documents in HTML, XHTML, SMIL, MathML, etc. If you wish to validate specific content such as RSS/Atom feeds or CSS stylesheets, MobileOK content, or to find broken links, there are other validators and tools available.

The W3C validators are hosted on server technology donated by HP, and supported by community donations.
Donate and help us build better tools for a better web.

This service runs the W3C Markup Validator, v0.8.5.
COPYRIGHT © 1994-2009 W3C® (MIT, ERCIM, KEIO), ALL RIGHTS RESERVED. W3C LIABILITY, TRADEMARK, DOCUMENT USE AND SOFTWARE LICENSING RULES APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR PUBLIC AND MEMBER PRIVACY STATEMENTS.

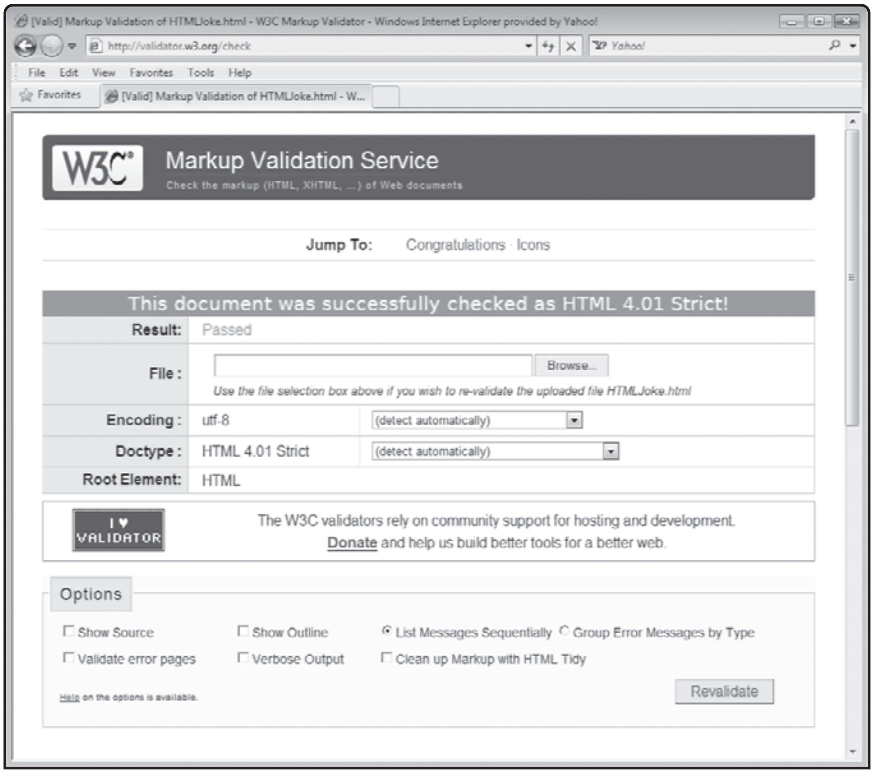
FIGURE 2.8

Use the W3C markup validation service to ensure that all your web pages are valid and well formed.

Using this free service, you can ensure that all of your web documents are valid and therefore should render in most web browsers in a consistent and predictable manner. You can use this service in any of three ways. First, if you have uploaded your document to the Internet, you can enter its URL. Second, if your web page still resides on your computer, you can upload it as a file. Finally, you can copy and paste the contents of your page into a form provided by the service. Regardless of which option you select, the service will analyze your markup and display its results, allowing you to ensure that your (X)HTML documents are well formed and to fix them if they are not.

Most web browsers are able to deal with documents that are not well formed. However, the results generated from (X)HTML documents that are not well formed may cause less than desirable results that differ from browser to browser.

Figure 2.9 shows an example of the results you will see when an HTML 4.01 document is well formed.


[Valid] Markup Validation of HTMLJoke.html - W3C Markup Validator - Windows Internet Explorer provided by Yahoo!
http://validator.w3.org/check
Yahoo!
File Edit View Favorites Tools Help
Favorites
[Valid] Markup Validation of HTMLJoke.html - W...
W3C
Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents
Jump To: Congratulations · Icons
Browse...
I ♥ VALIDATOR
Options
Revalidate

This document was successfully checked as HTML 4.01 Strict!

Result:	Passed	
File :	Use the file selection box above if you wish to re-validate the uploaded file HTMLJoke.html	
Encoding :	utf-8	(detect automatically)
Doctype :	HTML 4.01 Strict	(detect automatically)
Root Element:	HTML	

The W3C validators rely on community support for hosting and development.
Donate and help us build better tools for a better web.

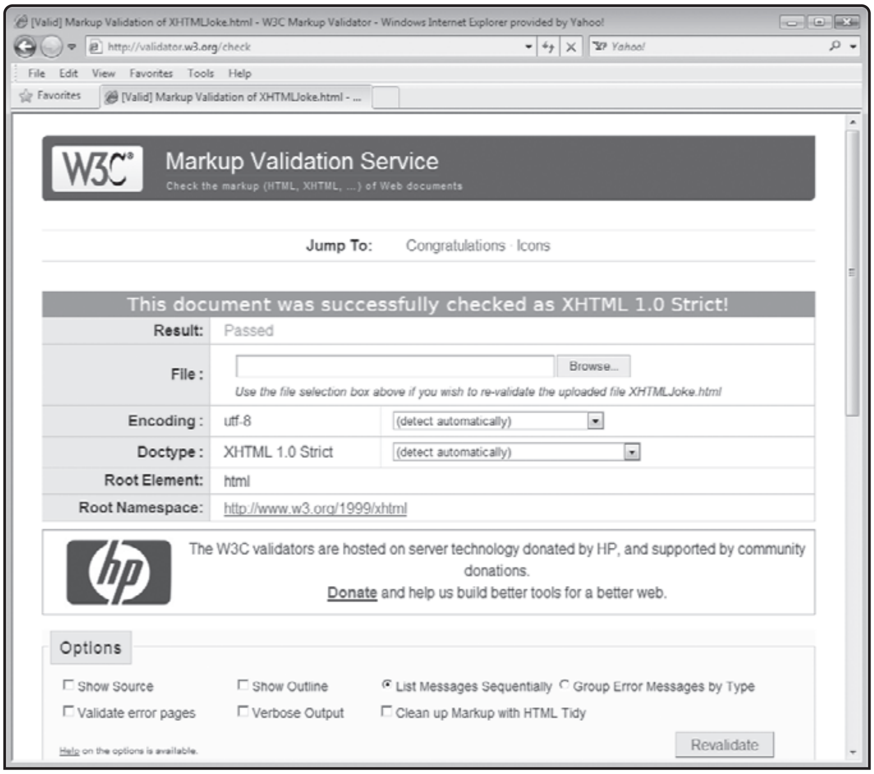
☐ Show Source ☐ Show Outline (● List Messages Sequentially) ○ Group Error Messages by Type
☐ Validate error pages ☐ Verbose Output ☐ Clean up Markup with HTML Tidy

Help on the options is available.

FIGURE 2.9

Error and Warning messages would have been displayed if the document was not well formed.

Similarly, Figure 2.10 shows the results that are displayed when an XHTML 1.0 Strict document is well formed.


[Valid] Markup Validation of XHTMLJoke.html - W3C Markup Validator - Windows Internet Explorer provided by Yahoo!
http://validator.w3.org/check
Yahoo!
File Edit View Favorites Tools Help
Favorites
[Valid] Markup Validation of XHTMLJoke.html - ...

Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents

Jump To: Congratulations · Icons

This document was successfully checked as XHTML 1.0 Strict!

Result:	Passed	
File :	Browse... Use the file selection box above if you wish to re-validate the uploaded file XHTMLJoke.html	
Encoding :	utf-8	(detect automatically)
Doctype :	XHTML 1.0 Strict	(detect automatically)
Root Element:	html	
Root Namespace:	http://www.w3.org/1999/xhtml	

The W3C validators are hosted on server technology donated by HP, and supported by community donations.
Donate and help us build better tools for a better web.

Options

☐ Show Source ☐ Show Outline [✓ List Messages Sequentially] ○ Group Error Messages by Type
☐ Validate error pages ☐ Verbose Output ☐ Clean up Markup with HTML Tidy

Help on the options is available.

Revalidate

Figure 2.10

Well-formed markup should display consistently in different web browsers.

Configuring Element Attributes

Most (X)HTML elements support a range of attributes, which you can modify to configure the appearance and behavior, or the element based on its use in current circumstances. Like all XHTML syntax, attributes must be spelled in all lowercase. HTML allows for both upper- and lowercase. For example, the anchor element `<a>` uses the `href` attribute to specify the URL to which the link points, as demonstrated here:

```
<a href="http://www.microsoft.com">Microsoft's website</a>
```

Attributes are always specified after the element name and are only allowed in opening elements or in single elements before the self-closing tag. Most elements support a number of different attributes, allowing you to specify as many as you want in any order that makes sense to you, provided you separate each with a blank space. Note that according to XML syntax rules, you must specify attribute values within matching double-quotation marks using the syntax outlined here:

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

(X)HTML elements support many different types of attributes. Many attributes are limited to specific values while others can take any text value you assign them. To determine which types of attributes are supported by each type of (X)HTML element, visit http://www.w3schools.com/tags/default.asp.

Element attributes are made up of an attribute name followed by the equals sign and then an assigned value, as demonstrated here:

```
<p id="Intro">A long time ago in a far away land…</p>
```

Here, a paragraph element has been assigned an `id` of `Intro`. `Id` is a universal attribute that can be assigned to any (X)HTML element. The `id` attribute will allow the paragraph to be referenced from elsewhere within the document, perhaps by a CSS style rule that set the font type, style, or size of the text that makes up the paragraph. Note that the quotation marks around the `id`, though optional in HTML, are required in XHTML strict.

Standard Element Attributes

As you learn different (X)HTML elements throughout this book, you will be introduced to many of the attributes that the elements support. However, there are a number of universal attributes that are common to just about every element. Use of these elements is entirely optional but often helpful. These elements are outlined below and demonstrated throughout the rest of the book.

`title`

`title` is an optional attribute that is used to assign a title to an element. Most browsers display the contents specified by the title in a tooltip when the mouse pointer is moved over the rendered element.

`id`

`id` is an optional attribute that specifies a unique name or identifier for an element, allowing the element to be referenced elsewhere, typically by CSS or JavaScript. When assigning a name to an `id`, the following rules apply:

- Each `ID` must be unique throughout the document
- Class names are made up of letters and numbers
- Class names are also limited to the following special characters: underscore (_) and hyphen (-)

`class`

`class` is an optional attribute used to define an element as being part of a class, allowing it along with all elements of that class to be referenced as a group. Any number of elements can be assigned to the same class. In addition, an element can be assigned to two or more classes. Classes are often used in conjunction allowing all elements within the same class to be assigned the same presentation rules. Classes are also used in conjunction with scripting. When working with classes, the following rules apply:

- When an element is assigned to more than one class, class assignments must be separated by spaces
- You may use any letter or number
- The first character must be a letter
- Except for the underscore (_) and hyphen (-) no special character can be used

`style`

The optional `style` attribute allows you to embed inline styling inside elements. Inline styles are seldom used by web developers, who generally defer to external style sheets and their inherent benefits. External style sheets are covered in Chapters 7 and 8.

Understanding Element Levels

There are two ways of working with (X)HTML elements, block level and inline. Block-level elements are used to display content on its own line, separate from other contents. Inline elements are designed to enclose small amounts of text embedded within block-level elements.

Working with Block-Level Elements

Block-level elements are elements that display their content on their own line, apart from other element content. An example of such an element is the `p` (paragraph) element, which is used to organize text into its own separate block, as demonstrated here:

```
<p>Perhaps today is a good day to die!</p>
```

Other examples of block-level elements include the `div` element and each of the heading elements (`h1`, `h2`, `h3`, `h4`, `h5`, and `h6`).

Embedding Inline Elements

Inline elements enclose smaller amounts of text for the purpose of highlighting them in some fashion. Block-level elements can stand on their own within an (X)HTML document. They may also be embedded within other block statements. Inline elements cannot stand on their own. To use them, they must be embedded within a block-level element as demonstrated here:

```
<p>The first letter of people's names <strong>should</strong> always be
capitalized.</p>
```

In this example, the inline `strong` element, which places strong emphasis on a word or words, has been embedded within a `p` element. An inline element can, however, contain another inline element provided the outer inline element resides within a block-level element, as demonstrated here:

```
<p>The first letter of people's names <em><strong>should</strong></em> always
be capitalized.</p>
```

Here, the `em` element has been added around the `strong` element to further add emphasis to a word within a `p` element.

Nesting Elements

Most (X)HTML elements allow you to embed them within other elements. You must properly nest embedded elements to provide valid and well-formed (X)HTML documents. Otherwise, errors will occur, as demonstrated in the following example.

```
<p>The first letter of people's names <em><strong>should</em></strong> always
be capitalized.</p>
```

Here, the `em` and `strong` elements are not properly embedded.

When embedded elements are within one another, it is essential that you remember to complete the inner element before you close the outer tags, as demonstrated here:

```
<div><p>Hello World!</p></div>
```

Failure to follow this simple rule will result in errors. An example of improperly nested elements is provided here:

```
<div><p>Hello World!</div></p>
```

Some (X)HTML elements are specifically designed to be used in a nested manner. Examples of these types of elements include the `<ol>`, which defines an ordered list, and `<li>` element, which defines an item within a list. The following example demonstrates these two types.

```
<ol>
  <li>Apples</li>
  <li>Oranges</li>
  <li>Pears</li>
  <li>Grapes</li>
</ol>
```

Commenting Your Markup

(X)HTML consists of a collection of English-like tags, which is the basis for defining elements. Despite this, (X)HTML documents can be quite complex. One way to make your document easier to understand and support is to embed comments inside your documents that explain what is going on and why you have laid out your documents in the manner you have.

(X)HTML comments are created by embedding text within an opening `<!--` tag and a closing `-->` tag. For example,

```
<!-- The following paragraph introduces the story's main character. -->
<p>Once upon a time there was a wizard named Gandor.</p>
```

If you need to, you can spread comments over multiple lines, as demonstrated here:

```
<!-- The following paragraph introduces
the story's main character. -->
<p>Once upon a time there was a wizard named Gandor.</p>
```

TRICK

One common use of comments is to temporarily comment out one or more elements when developing and testing documents. For example, when troubleshooting a problem with a document you might temporarily comment out one or more elements that you suspect to be the source of the problem in order to see how the rest of the document is rendered when those elements are not processed.

Web browsers will not display the contents stored in a document's embedded comments. You should use them liberally throughout your documents to document every major part of the document.

Improving Document Organization with White Space

One nice feature of (X)HTML is that you are permitted to use white space at will for improved formatting of your documents without affecting the document in any meaningful way (other than increasing its size). You add white space to your documents by inserting extra spaces and line breaks into your documents.

When web browsers load (X)HTML documents, they automatically ignore all of the extra white space by collapsing all extra space down to a single space. For example, in the following example, extra white space has been added before and after different elements in order to make the resulting statements easier to view and maintain.

```
<div>

  <p>

      Once there was a hero named <em>Mighty Molly</em>!

  </p>

</div>
```

When rendered by the browser, the output generated by this example is identical to that generated by the following example. However, as you can see, it is clearly easier to view and understand what is going on in the first example thanks to the extra white space.

```
<div><p>Once there was a hero named <em>Mighty Molly</em>!</p></div>
```

Likewise, this third example will be rendered in an identical manner.

```
<div>

      <p>

   Once    there    was     a     hero     named
      <em>Mighty Molly</em>!

      </p>

  </div>
```

As this example shows, the overuse of white space in this third example has become anti-productive. Obviously too much of a good thing is not always good. When rendered by the browser, this example's output, shown next, is the same as the other examples.

```
Once there was a hero named Mighty Molly
```

HINT If you need to preserve white space, you can enclose your content within the `pre` element, covered in Chapter 4.

Finding a Web Host for Your Web Pages

In Chapter 1, you learned how to create a web page and test it by using the browser to open a copy of the web page stored on your own computer. However, in order to be able to share your website with the rest of the world, you need to upload your web documents to a web server. A web server is simply a specially configured server that is connected to the Internet and whose purpose is to accept requests from web browsers and return specified web pages and other types of content.

If you do not already have a web host, there are a number of different ways of finding a good web host for your web pages. You could begin your search by checking with your Internet service provider. Sometimes Internet service providers offer their customers a little web server space as part of their service. Another option for finding a host for your web pages is one of any number of free website hosts like Google (http://googlewebhosting.net/). The price that you pay for free services like this is the display of advertisements on your web pages, typically in the form of banners. A third option that you can pursue is to find a web host provider. You have to pay a little for this service but it is often the best choice. For as little at $7.95 per month, a good web host provider will provide you with space to store your best pages, multiple e-mail accounts, support for advanced features like PHP, MySQL, Ruby on Rails, Perl, Python, website statistics, and many other options. An example of one such web host is site5 (www.site5.com).

Once selected, your web host will provide the URL of your website, which might be something like www.*hostname*.com/*sitename*. Here, *hostname* is the name of your web host and *sitename* is the name of your website.

Once you have found a web host you are comfortable with, your provider will give you instructions on how to access your website and how to upload and manage your web pages. Most web hosts allow you to upload web pages one of two ways. First, you are usually given a graphical user interface through which you can upload and manage your web documents. Second, you can usually use *FTP*, which stands for *File Transfer Protocol*. Using an FTP client and the FTP address provided by your web host, you can upload and download files from your website. You'll also be able to create a folder structure within which to store your files.

HINT There are a number of very good FTP clients available for download on the Internet. Examples include FileZilla (filezilla-project.org/).

Once you have uploaded your web pages, you can then access and view them using your web browser.

Back to the Linked Jokes Project

Now it is time to return your attention to the Linked Jokes project. In this project, you will create three separate XHTML documents. The first document will display two jokes, each of which is also a link that when clicked instructs the browser which of the other two XHTML documents it should load. As you can see, rather than display the jokes and punch lines all on the same web page, this chapter's project uses links to control navigation from a primary page to two other pages.

Designing the Application

As was the case with preceding chapter projects, you will develop the Linked Jokes project in a series of steps as outlined here:

1. Create the project's HTML documents.
2. Develop the document's markup.
3. Load and test the HTML page.

What makes this project different from the HTML Jokes project that you created in Chapter 1 is the movement of joke punch lines to external web pages, which are loaded and displayed in the web browsers when their corresponding links are clicked.

Step 1: Creating New HTML Document

The first step in the creation of this project's web document is the creation of the empty text files. Begin by opening your preferred code or text editor and creating and saving the following three files, using the names outlined here:

- **LinkedJokes.html.** The main landing page whose URL visitors will use to view this web application.
- **PunchLine1.html.** A web page that is loaded when visitors click on the link for the first joke on the LinkedJokes.html document.
- **PunchLine2.html.** A web page that is loaded when visitors click on the link for the second joke on the LinkedJokes.html document.

Step 2: Developing the Document's XHTML

The next step in creating the Linked Jokes project is the development of the markup for all three of the project's web documents. Begin by adding the following elements to the LinkedJokes.html document.

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

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

  <head>

  </head>

  <body>

  </body>

</html>
```

These statements include the `DOCTYPE` element and the base set of elements required to build an XHTML page. The next step in the development of the Linked Jokes project is to finish the LinkedJokes.html document and to create the project's other two documents.

Completing the LinkedJokes.html Document

To complete the LinkedJokes.html document, you need to modify it by embedding the `meta` and `title` elements as shown next. The `meta` element specifies the content type and character set used by the document and the `title` element specifies a text string that will be displayed in the web browser's titlebar.

```
<head>
  <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
  <title>Chapter 2 - LinkedJokes</title>
</head>
```

You also need to update the document's `body` section by modifying it, as shown here:

```
<body>
  <h1>Linked Jokes</h1>
  <a href="punchline1.html">What does Mickey Mouse's wife drive?</a><br />
  <a href="punchline2.html">How do you fix a tooth?</a>
</body>
```

This markup consists of a level 1 heading and two links that display the document's jokes. When clicked, these links instruct the browser to load and display the documents specified by each link's `href` attribute.

Creating the PunchLine1.html Document

Next, open the PunchLine1.html documents, add the following statements to it, and then save the file.

```
<!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>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 2 - PunchLine1</title>
  </head>

  <body>
    <p>A Minnie-van</p>
 </body>

</html>
```

As you can see, this web document is very similar to the LinkedJokes.html pages, except that instead of two links, it displays a paragraph showing the punch line for one of the game's jokes.

Creating the PunchLine2.html Document

Lastly, open the PunchLine2.html documents, add the following statements to it, and then save the file.

```
<!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>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 2 - PunchLine2</title>
  </head>

  <body>
    <p>With Toothpaste</p>
```

```
 </body>

</html>
```

As you can see, the only difference between this and the PunchLine1.html document is the content stored in the `title` element and the content located in the `body` section's paragraph element.

Step 3: Loading and Testing Your New Web Documents

At this point, all three documents that make up the Linked Jokes project have been completed and you are ready to load and view the results of your work. To do so, open your favorite web browser and then execute the following procedure.

1. Click on the browser's File menu and select the Open File command. This displays a standard file open dialog.
2. Using the dialog window, navigate to the folder where you stored the web page and select it.
3. Click on the Open button. The browser will then load and display the specified document as was demonstrated back in Figure 2.1.

If anything looks out of place on your version of this project, go back and recheck your work. If, after clicking one of the links located on the main page you see an error indicating that the specified web page cannot be found, double-check the URL that you entered into the browser and make sure that the case used within the links matches the case you used when you saved both the PunchLine1.html and PunchLine2.html files.

Summary

This chapter provided an overview of (X)HTML syntax rules and outlined the requirements of how to work with tags and tag elements. You learned about all six (X)HTML standards, how they compare to one another, and how to specify the one you want to work with in your web document. You learned about the different ways in which elements are formulated and how to construct tags. You were introduced to (X)HTML standard tags and learned how to improve document organization and presentation using white space and comments. You also received guidance on what to look for in a web host.

Challenges

1. As currently written, the Link Jokes web application only displays two jokes. Consider modifying it to suit your own personal style by replacing or adding to its collection of jokes.
2. Consider adding more text to the document using the `p` (paragraph) element that explicitly instructs the user to click on a joke in order to see its punch line.
3. After making changes to your web documents, consider visiting www.validator.w3.org and ensuring that they are still well formed.