# Chapter 1

# How to Write XHTML and CSS



XHTML and CSS are two different animals, or specifications used to create web pages. Each has a distinct look and purpose. When used together, the combination can produce a useful, information-rich, and highly attractive web page.

If you see an example here or in any later chapter typed in a particular way, you should copy that exactly as you type along with the exercises in the book. The rules defining how a language is put together are its *syntax*. In this chapter, you will learn the syntax of XHTML and CSS. You will learn what each of these specifications does, how each looks, and how to write both. Basic rules for typing both XHTML and CSS, such as when to use the spacebar, when to type a semicolon, or when to type a bracket, will also be explained in this chapter.

## Anatomy of a Website

For those of you who have never built a website before, a summary of what goes into a site may help you understand what HTML/XHTML and CSS do and how they can work together to implement your vision. (If you've already worked with one of the "visual tools" like Dreamweaver or FrontPage, you're a little further along the learning curve, but this recap will still help to put what you're about to learn into perspective.)

If you have explored the World Wide Web, you know that a web page may contain text, images, links, sounds, and movies or moving images. You may also be aware that some pages use scripts written in various languages such as JavaScript, PHP, or ASP to create interactivity, to connect the page to a database, or to collect information submitted in a form.

The glue that holds all those pieces and parts together and displays it in a browser such as Internet Explorer or Netscape Navigator in a readable or useable manner is HTML or XHTML. The browser is your window on the World Wide Web; XHTML is the language used to tell the browser how to format the pieces and parts of a web page.

CSS enters the scene by adding style to the formatted elements on a web page. The style might involve color, placement, images, fonts, or spacing, but it would not change the underlying pieces and parts formatted by the XHTML.

**NOTE** The Internet is a vast collection of interconnected computer networks from all over the world. The World Wide Web (WWW) is a part of the Internet but is not the Internet itself. The Internet has many parts besides the WWW, such as e-mail.

### What Are XHTML and HTML?

Hypertext Markup Language (HTML) is the programming specification for how web pages can be written so they will be understood and properly displayed by computers. XHTML is an acronym for

Extensible Hypertext Markup Language, a specification that grew out of HTML. You'll see what "extensible" means in a moment, but to understand the role of HTML and XHTML, you need to understand the three terms in HTML.

*Hypertext* is simply text as it exists in what is called "hyperspace"—the Internet. It is plain text that carries the content of your web page and the programming information needed to display that page and link it to other pages. Hypertext is formatted via a *markup language*—a standardized set of symbols and codes that all browsers can interpret.

**NOTE** The organization devoted to creating and publishing the standardized rules for various web technologies, including XHTML, is the World Wide Web Consortium, or W3C, at `www.w3.org`. See also the Web Standards Project, a grassroots coalition fighting for the adoption of web standards, at `www.webstandards.org`.

*Markup* is used to convey two kinds of information about text or other content on a web page: first, it identifies what kind of *structure* the content requires. If you think of a web page as simply a whole lot of words, the HTML is the markup, or framework, that specifies that certain words are headings or lists or paragraphs. The way you mark up the text on the page structures the page into chunks of meaningful information such as headings, subheads, and quotes.

Markup may also define the *presentation* of those elements; for example, the different fonts to be used for headings and subheadings. When it was first developed, HTML was the only tool for defining visual presentation on screen. When the World Wide Web began, the only information transmitted using the *Hypertext Transfer Protocol (HTTP)* was text. As the capability to transfer images, sounds, and other information was added, presentational markup was added to HTML to help format the new information. After a few years of amazing growth, the HTML that was being used to mark up individual elements reached burdensome proportions. It became apparent that markup for presentation was an inefficient way to define what every item of text or graphics on a website should look like, and the web community developed Cascading Style Sheets (CSS) as a better way to handle presentation.

## What's the Difference between XHTML and HTML?

XHTML actually *is* HTML—it is the most recent standard for HTML recommended by the W3C. XHTML was chosen for the basis of the codes used in this book because it is the current recommendation. You will be learning HTML when you learn XHTML. It is a two-for-the-price-of-one bargain. There are a few basic differences in writing XHTML versus writing HTML, and these will be pointed out to you at appropriate times in the book.

XHTML is more than HTML, because it is extensible. XHTML uses the syntax rules of the *Extensible Markup Language (XML)*. An extensible markup language can be extended with modules that do things such as make math calculations or draw graphical images. Web pages written in XHTML can interact with XML easily.

## What Is CSS?

CSS is an acronym for Cascading Style Sheets, another programming specification. CSS uses rules called *styles* to determine visual presentation. The style rules are integrated with the content of the web page in several ways. In this book, we will deal with style rules that are embedded in the web page itself, as well as with style rules that are linked to or imported into a web page. You will learn to write the style rules and how to import, link, or embed them in the web pages you make.

In HTML, styles can be written into the flow of the HTML, or *inline*, as well.

CSS can also be integrated into web pages in other ways. Sometimes you have no control over these rules. Browsers allow users to set up certain CSS style rules, or user styles, according to their own preferences. The user preferences can override style rules you write. Further, all browsers come with built-in style rules. Generally the built-in styles can be overridden in your CSS style rules. Built-in browser display rules are referred to as *default* presentation rules. Part of what you will learn is what to expect from a browser by default, in order to develop any new CSS rules to override those default display values.

## Getting Started with XHTML Syntax

The building blocks of XHTML syntax are *tags*, which are used to mark up *elements.* A tag is a code that gives an element its name. For example, the tag used to format a paragraph is a `p` tag, which is called either a "paragraph tag" or "a p tag." When text is marked up with a `p` tag, it is an instruction to the browser to display the element as a paragraph.

Elements in XHTML, such as paragraphs, can also have attributes and values assigned to them. But before you find out about attributes and values, let's dig into tags just a bit more.

### Opening and Closing Tags

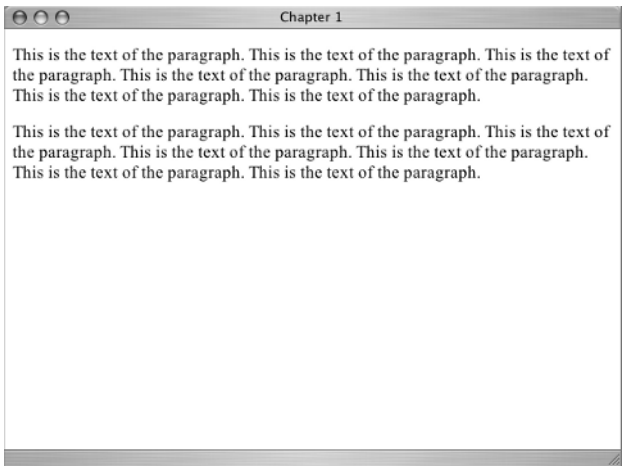
Opening and closing tags are used to specify elements. Here is a marked up paragraph element:

```
<p>This is the text of the paragraph.</p>
```

The paragraph is opened with a `p` tag. Tags are enclosed by angle brackets (< and >). So the markup `<p>` instructs the browser that a paragraph starts now.

A closing tag `</p>` indicates the end of the paragraph. Notice that the closing tag is the same as the opening tag, with the addition of a forward slash (/) before the tag. Tags are rather like on and off switches: turn on a paragraph here and turn it off there. With a few more sentences added to make the paragraph show up, and a second identical paragraph added, this element would appear something like Figure 1.1 in a browser.

**FIGURE 1.1**
Two paragraph elements



Notice that the browser left a blank line between the two paragraphs and that there is no indenting. This is an example of a *default* paragraph. A default display is the browser's built-in interpretation of what the element should be. One way to change the browser's default interpretation of an element is to include additional instructions in the form of *attributes* and *values* that further define the element.

An attribute is information about the element. An example of an attribute that might define a paragraph is alignment. Text in paragraphs can be left-aligned, right-aligned, centered, or justified. As you can see in Figure 1.1, the browser default for text alignment is left-aligned. In XHTML, the type of alignment you choose is the value. The exact attribute is `align`. The value of this attribute could be `left`, `right`, `center`, or `justify`.

An attribute is written as part of the opening tag. The attribute name is followed by an equal sign (=) and the value in quotation marks (").

Here is a marked up paragraph element with an attribute name and value.

```
<p align="right">This is the text of the paragraph.</p>
```

By adding `align="right"` to the first paragraph element in Figure 1.1, you can see the alignment change to an appearance similar to Figure 1.2.

There are two important things to take note of in this example. First, there is a space between the tag and the attribute name, `align`. The attribute is followed by an equal sign and the attribute value `"right"` is enclosed in quotation marks with no surrounding spaces. Attribute/value pairs in XHTML always follow this syntax. Also notice the closing tag. It does the job of ending the paragraph and the effect of the paragraph's attributes merely by using the forward slash (/) and the `p` again. When the paragraph ends, all of its attributes and values terminate with it.

One of the distinctions between XHTML and HTML is that closing tags are *required* by XHTML. In HTML, closing tags are not always required.

## Empty Elements

Before I describe empty elements, let me quickly define nonempty elements. An element with text in it, such as the previous paragraph examples, is considered nonempty.

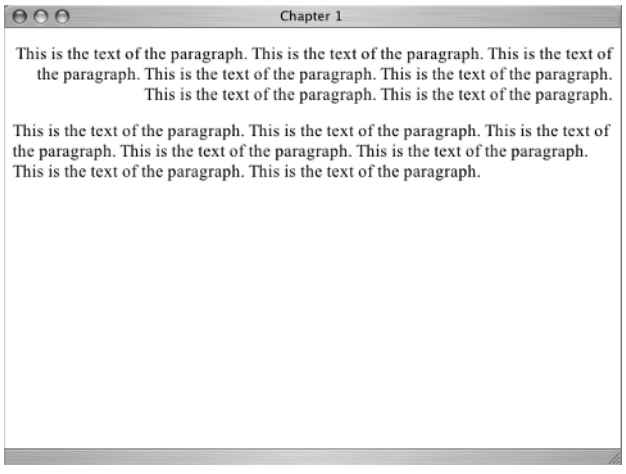



**FIGURE 1.2**
The first paragraph with `align="right"`

Sometimes you put something on a web page that does not contain text. Such elements don't need closing tags and are referred to as *empty elements*. An example would be an image or a line break. But before I get into the requirements for empty elements in XHTML, here's an example in HTML:

```
<p>Jingle bells, jingle bells,<br>
Jingle all the way...</p>
```

The HTML tag `<br>` (for break) is used for a line break in formatting this paragraph. The line break doesn't have to open and close, it merely has to *be.* A line break moves down to the next line, without any intervening white space, which you would get by default if you put the second line in a new paragraph element.

Formatted with a break in the first line, this paragraph would display like Figure 1.3 in a browser.

However, XHTML uses syntax based on the rules of Extensible Markup Language (XML) to write HTML. One of the requirements of XML is that every element must be terminated whether it is empty or not. You may be asking how an element can be "terminated" if there is no closing tag. The solution for XHTML is to add the closing forward slash to the empty tag itself. Empty tags in XHTML look like this example.

```
<p>Jingle bells, jingle bells,<br />
Jingle all the way...</p>
```

Notice the space between the `br` and the forward slash (/). Even empty tags with attributes and values can be closed in this way.

**NOTE** The space before the closing forward slash is not required by XHTML. In other words, `<br/>` would be considered correct. However, inserting a space enables older browsers to correctly display the document, so I will use the space here. (Older browsers are those versions earlier than 5, such as Netscape 4.7.)

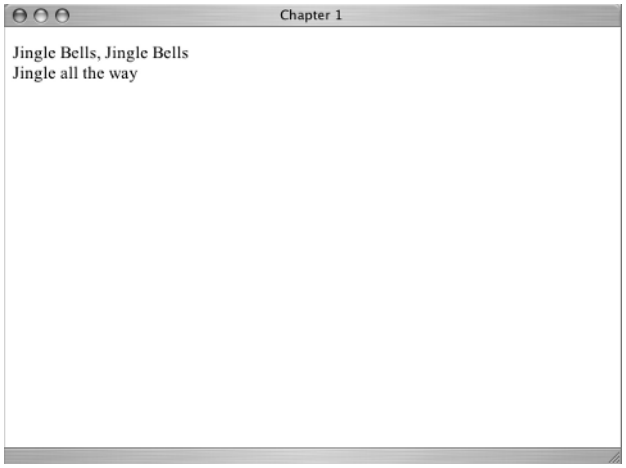



**FIGURE 1.3**
The line break

The `img` (for image) tag is another empty element. Look at this example:

```
<img src="photo.gif" />
```

This empty element places an image on the page, and the source of the image is given as an attribute of the `img` tag. The space and forward slash at the end give the empty element the required XHTML closing.

There are not many empty elements in XHTML. Others include horizontal rules, the link element, and meta elements. Most of the time you will mark up text with both opening and closing tags. Even if you are writing HTML, where closing tags are not always required, it is considered good practice to include closing tags whenever possible.

### XHTML: Specific Requirements

As mentioned previously, XHTML use XML syntax rules. In addition to the requirement that every element be terminated, there are several other specifics about writing XHTML that are different from HTML requirements:

- Specific DOCTYPE declarations are required, which you will learn about in Chapter 3.
- All elements, attributes, and values must be in lowercase.
- All values must be enclosed in quotation marks. Values can be quoted with single or double quotation marks, but you must be consistent about using the same type each time. The examples in this book will consistently be in double quote marks (").
- Every attribute must be given an explicit value.

Although these rules are not required when writing HTML, they all work just fine in HTML. The only XHTML syntax rule that does not produce valid HTML is using the forward slash to terminate an empty element. Should you decide to use HTML instead of XHTML, you will need to make only this minor adjustment to your coding habits to write an HTML page.

**TIP** Throughout this book you'll learn the most important XHTML tags and attributes, but as you begin working on your own you'll find it valuable to have a complete reference to the language. You can find that reference in *HTML Complete* (Sybex, 2003), a compilation of useful information that also contains a command reference for CSS.

## Getting Started with CSS Syntax

*Cascading Style Sheets (CSS)* are used to add presentational features to elements within your markup. CSS can set colors, fonts, backgrounds, borders, margins, and even the placement of elements on the page.

A stylesheet can be either placed directly in an XHTML document or linked to it as a completely separate file. In Chapter 2 you'll explore both these approaches, but most of the time CSS is linked to the XHTML page. In one document, you'll have your XHTML page, which you will learn to plan in a clean, logical structure of the headings, paragraphs, links, and images needed to present your ideas. In another file, you'll have your stylesheet, which gives color, emphasis and pizzazz to your display. This way, you can change the way your web page looks simply by changing the stylesheet and without changing the content at all.

The power to change a site's complete appearance by changing the stylesheet gives you great flexibility in its appearance. It also saves enormous amounts of time on maintenance and upkeep, since style rules are in a file that is apart from the content. Any number of web pages can be linked to a single stylesheet, so it becomes merely a matter of minutes to make sweeping changes to the appearance of all those pages. Once a stylesheet has been downloaded by a browser, it is saved in a special folder called *cache*. The next time the browser downloads a page using that stylesheet, there is no wait for the user while it downloads because the browser already has it in cache. So every page that uses that stylesheet will download very quickly, saving waiting time and bandwidth charges.

**NOTE** Visit `www.csszengarden.com` to see inspirational examples of the same content styled in many different ways using CSS.

Styles and stylesheets look very different from XHTML, and a different set of syntax rules is used for writing styles.

## Selectors and Declarations

Style rules are written with *selectors* and *declarations*. Selectors, well, *select.* That is, they select which elements of an XHTML page the style will apply to. The most basic selector is the element selector. For example, the selector `p` selects all the paragraph elements on a page.

For each selector, you write a set of declarations that govern how the selected element will be displayed. Together the selector and declarations make up a style rule or, more simply, a style. Here is a set of style declarations for the selector `p`:

```
p {
font-family: Arial, Helvetica, sans-serif;
font-size: small;
color: blue;
}
```

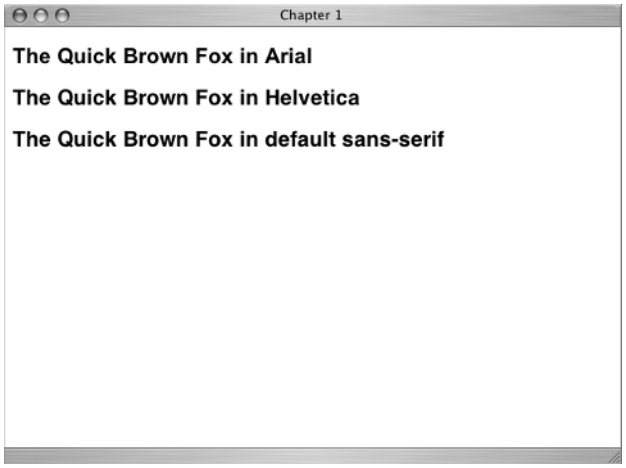
Let's examine that bit by bit. You already know that the `p` is the selector. Everything that comes between the two curly braces ({ }) is the declaration block, which contains three different declarations.

A declaration consists of a *property* followed by a colon, a space, and then the *value*. A semicolon follows the value. As you can see in this example, a property in CSS is similar to an attribute in XHTML. They both identify a characteristic of the element you are formatting. The first property I declared in this example is the font family to be used for text in the paragraphs. I specified Arial as my first choice, if the user's computer has it. If not, Helvetica will do, and if neither is available, the system's default sans-serif font will have to do. See Figure 1.4 for examples of these font families.

**TIP** *Font family* is the slightly fussy typographical term for what we usually call a *typeface* or just a *font*. Strictly speaking, every variation in size and weight within a typeface is considered a separate font, and the whole set of these variations is considered the font family.

It is considered good practice to include more than one font family in a declaration because not all computer systems come equipped with the same set of fonts. As in this example, the fonts are normally listed in the order of preference.

**FIGURE 1.4**
Sans-serif font families




Generally, if no font family is specified, a browser will use Times as the default. See Figure 1.5 for examples of serif fonts.

**TIP** You'll learn more about fonts and font families in Chapter 4. Typography and fonts are the topic of many books, including *The Non-Designer's Design Book* by Robin Williams. CounterSpace at `http://counterspace.motivo.com/` provides a good introduction to the topic.

The second declaration in the preceding rule is `font-size: small`. You will learn about the various options in font sizes in Chapters 4 and 5, but I'm sure you can guess that this declaration sets the font for all the `p` elements to a small size. The final declaration sets the color to blue.

**NOTE** Unless a user has changed the browser default settings, the default font-size setting in most browsers is medium.

This style rule has the effect of making every paragraph on that page appear in a font that is slightly smaller than normal, blue, and Arial.

In the examples in this book, each style declaration is written on a separate line, and the closing curly brace is on its own line as well. This makes the style easier to read. However, style rules don't have to be typed in exactly that form. For example, you could write it like this:

```
p {font-family: Arial, Helvetica, sans-serif; font-size: small; color: blue;}
```

If you do put more than one declaration on a line, be sure to leave a space after the semicolon.

Some styles can be written in shorthand form. For example, `font` can be used as shorthand for all the font properties including `font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, and `font-family`. That allows you to combine the two declarations about fonts into one shorthand declaration like this:

```
p {
font: small Helvetica, Arial, sans-serif;
color: blue;
}
```

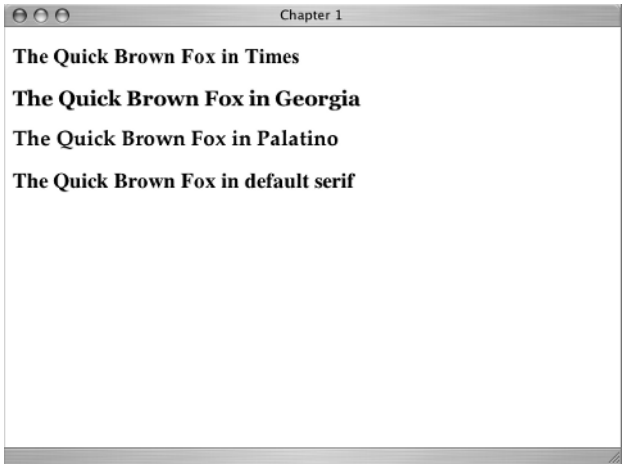



**FIGURE 1.5**
Common serif font families

Color is not considered a property of font. The `color` property expresses a foreground color, meaning that the text in the foreground of the page will be in the color named. Color is distinguished from `background-color`, which, as you can probably guess, sets a background color for the entire element.

#### SELECTORS GET SPECIFIC

Often you'll need to be more explicit in styling elements in the XHTML than the first example shows. CSS does allow for more specific selectors than the general element selector just described. Since the selector distinguishes what element in the document will be affected by the style rule, an element selector such as the `p` selector in the example above will affect all the `p` elements. There are times when you want particular (or what CSS terms *specific*) instances of paragraphs to follow different rules from those assigned to all `p` elements in general. Two of those types of selectors are the *ID selector* and the *class selector.* These two selectors allow you to write style rules for elements in a particular context. For example, instead of styling all the paragraphs on a page, you can style only the paragraphs of a certain class or ID.

**NOTE** The types of selectors included in this book are element selectors, class selectors, ID selectors, contextual selectors, pseudo class selectors, and group selectors.

#### *ID Selectors*

IDs can only be used once per XHTML page. They are usually used to identify content that you style as a structural unit, such as a header, footer, content block, or menu. We will be working with this concept in almost every chapter of this book, but for now, you will simply see how ID selectors look in the stylesheet.

ID selectors are preceded by this symbol: #. The correct term for this symbol is "octothorpe," but most people in America call it a pound sign or hash sign. In this book, we will use the term hash sign for this symbol. An `id` rule in a stylesheet looks like this:

```
#footer {
font-size: x-small;
}
```

**NOTE** In XHTML `id` is in lowercase.

This rule would make everything in the section (or division) of the page identified as `footer` extra small. Notice that there is no space between the hash sign and the ID name.

But suppose you don't want everything in the footer to be extra small—you want only the paragraph in the footer to be extra small? You could accomplish this using a *contextual selector* that applies to only a paragraph in the division of the page identified as `footer`:

```
#footer p {
font-size: x-small;
}
```

Notice the space between the `#footer` and the `p`. This font-size value won't apply to other paragraphs on the page, only to paragraphs placed within the context of the footer section. This use of the ID selector followed by the element selector is very specific to only particular paragraphs in particular parts of the page.

The selector `#footer p` is a contextual (or *descendant*) selector. You can build contextual selectors into your XHTML with named IDs that allow for finely drawn CSS selectors. In the upcoming chapters, you will use descendant selectors to create many CSS styles.

You create the names for the `id` selectors yourself. They don't have to relate to any XHTML tag. It is good practice to create a simple and meaningful name that reflects the structural purpose of the content of the section identified with an `id`. The id selector `#footer` is a good example of a name that reflects some meaningful purpose on the page. If you worked on a stylesheet and went back to it after several months, the id selector `#footer` would still make sense to you as you reviewed and changed the stylesheet.

#### *Class Selectors*

Class selectors can be used as many times as you want per XHTML page. Class selectors are preceded by a period (`.`). As with `id` selectors, you create the `class` name yourself. If you want a style that will highlight certain terms on your page, you can create a `class` and name it `term`.

```
.term {
background-color: silver;
}
```

This style rule would put a silver background behind any words or phrases that were identified as being in the class `term`. Notice that there is no space between the preceding period and the name of the class.

**NOTE** One of the reasons CSS is popular is because a page's whole look can change almost instantly. It is good practice to choose class names that express purpose rather than some momentary choice such as a color, which might be changed later. So a class named `.term` is a better choice here than a class named `.silver` because the name `term` will continue to make sense no matter what color is used.

You can use complex combinations of selectors, IDs, and classes to style specific sections of your pages. The following selector would apply only to paragraphs of a class called `term` in a division of the page called `footer`.

```
#footer p.term {
background-color: gray;
}
```

Notice that when writing a style declaration for an element found in XHTML such as the `p` element, which is assigned to a particular class, there is no space between the element and the period and class name: `p.term`.

**NOTE** SelectORacle is a free tool at `http://gallery.theopalgroup.com/selectoracle/` that will translate complex CSS selectors into plain English to help you understand exactly what CSS rules are selecting.

#### Grouping Selectors

You may want to use the same style for several elements on your page. Perhaps you want all the paragraphs, lists, and block quotes on the page to have the same font size. To achieve this effect, list the selectors, separated by commas, and give the font-size declaration:

```
p, li, blockquote {
font-size: medium;
}
```

This rule makes the text in any paragraph, list, or block quote have the font size `medium`. Notice that there is a space between each item in the comma-separated list of selectors.

In this example, the comma sets up a rule for every element in the list. Here is a similar rule with no comma:

```
p blockquote {
font-size: medium;
}
```

Without the comma, it looks a whole lot like the preceding comma free `#footer p` rule, doesn't it? The selector `p blockquote`, with no comma, styles only a `blockquote` that's *part of* a paragraph, not every `blockquote` element.

The comma (or the lack of a comma) is an important distinction between grouped selectors and descendant selectors. Grouped selectors use commas. Descendant selectors do not.

**TIP** If you pay attention to the details such as whether you see a comma, a semicolon, a space, or a bracket and type carefully, you will have greater success with both the XHTML and the CSS examples in the book. If you do an exercise and it doesn't seem to work as I say it should, check carefully for typos. The syntax is exacting.

### Quotation Marks

The last difference between the syntax rules for writing XHTML and writing CSS that you'll look at before moving on to Chapter 2 involves quotation marks.

You recall that in XHTML, all attribute values in a tag must be enclosed in quote marks. In CSS style declarations, however, property values do not appear in quotes. Most of the time, you don't see quotation marks in stylesheets, although you do use quotation marks in stylesheets when listing a font with two or three words and spaces in the name of the font.

For example, you looked at this style rule earlier, in which no quotes were used:

```
p {
font-family: Helvetica, Arial, sans-serif;
font-size: small;
color: blue;
}
```

The fonts listed here are one-word font names. Sans-serif is hyphenated and doesn't have a space, so it is considered one word. However, if you want to list a font name such as Times New Roman, which is more than one word and includes spaces, you wrap the name in quotation marks, like this:

```
p {
font-family: "Times New Roman", Times, Georgia, serif;
}
```

Notice that the comma separating Times New Roman from Times is *after* the ending quotation mark. Also note that although specific font-family names such as Times are capitalized, generic font names such as serif are not.

## Real World Example

A strong thread emphasized throughout this book is that the use of the standard specifications recommended by the W3C by both web professionals and browser manufacturers makes writing XHTML and CSS easier, faster, and more universal.

There is a grassroots group working hard for the implementation of web standards called The Web Standards Project (WaSP) at `www.webstandards.org` (Figure 1.6).

The Web Standards Project site provides opportunities to take action in favor of web standards and offers information to help you learn to use those standards.

Be aware that any site given as a "Real World Example"—in fact, any site referred to in this book—is protected by copyright law from being copied. Copyright law protects both the images and the text in a website. I encourage you to look and learn, but not to take. There may be a few exceptions—for example, some CSS layout sites say in very clear terms that you have permission to take the material for your own use—but in general, it is best to assume that you do not have permission to "borrow" material from any website.


**FIGURE 1.6**
The Web Standards Project home page contains news in a section called Recent Buzz—listen to that WaSP buzz!

The Web Standards Project

## Summary

XHTML helps you create web pages. You'll use it to format the headings, paragraphs, tables, images, and lists that logically organize your information so that you can convey your message with words, images, and information. XHTML pages, even without any CSS attached to them, are displayed by browsers. Without any CSS attached, your page might look plain and simple, but all your information and content is still displayed in a nicely organized way.

CSS is about presentation and is used by most web designers to style for the screen. CSS determines whether something is blue or green, on the left or on the right, large or small, visible or hidden, has bullets or doesn't have bullets. However, stylesheets have to be applied to something else, such as XHTML pages, to have any effect.

In Chapter 2, you will build your first XHTML document and your first stylesheet. While you're doing that, you'll learn where to write styles and how to link to styles. Chapter 2 will explain the meaning of the *Cascade* in Cascading Style Sheets.