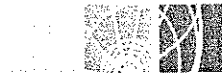


Web Design & Development Using XHTML



Jeff Griffin
Purdue University

Carlos Morales
Purdue University

John Finnegan
Purdue University

Franklin, Beedle & Associates, Incorporated
8536 SW St. Helens Drive, Suite D
Wilsonville, OR 97070
(503) 682-7668
www.fbeedle.com

President and Publisher	Jim Leisy (jimleisy@fbeedle.com)
Manuscript Editor	Jeni Lee
Production	Tom Sumner Bill DeRouchey Stephanie Welch
Cover	Ian Shadburne
Proofreader	Stephanie Welch
Developmental Editor	Sue Page
Marketing	Chris Collier
Order Processing	Krista Brown

Printed in the U.S.A.

Names of all products herein are used for identification purposes only and are trademarks and/or registered trademarks of their respective owners. Franklin, Beedle & Associates, Inc., makes no claim of ownership or corporate association with the products or companies that own them.

©2003 Franklin, Beedle & Associates Incorporated. No part of this book may be reproduced, stored in a retrieval system, transmitted, or transcribed, in any form or by any means—electronic, mechanical, telepathic, photocopying, recording, or otherwise—without prior written permission of the publisher. Requests for permission should be addressed as follows:

Rights and Permissions
Franklin, Beedle & Associates, Incorporated
8536 SW St. Helens Drive, Suite D
Wilsonville, Oregon 97070

Library of Congress Cataloging-in-Publication Data is available from the publisher.



XHTML

We know many of you are wondering why you should use XHTML (*Extensible Hypertext Markup Language*) or what is wrong with HTML. HTML has been the de facto standard for many years, but it is important to keep in mind that HTML is a simple markup language for formatting data. HTML has a fixed set of tags to define the structure and style of text and provides no provisions for expansion. It is this inflexibility that has allowed browser vendors to add many proprietary tags that may be supported only by their browser. It also does not allow individuals to add custom tags because none of the browsers would know how to parse the custom tag. The Web is full of poorly written HTML that violates the basic HTML rules. This has led to browsers having to include code to handle this poorly written HTML.

As the use of the Web continues to grow, it is increasingly being accessed by a variety of devices such as cell phones, personal digital assistants (PDAs), and pagers. These devices, with their displays that are not visually oriented, are driving the change in how we mark up documents for the Web. The *Extensible Markup Language (XML)* should solve many problems because content creators will have to focus on the structure of a document, not just how it looks. XHTML is a reformulation of HTML 4 into an XML application and will provide a migration path while maintaining backwards compatibility. XHTML 1.0 is the current recommended standard of the World Wide Web Consortium (W3C).

RELATIONSHIPS BETWEEN MARKUP LANGUAGES

Before we move on to XHTML, let's take a look at the relationship between the various markup languages.

In the middle 1980s, an international standard called *Standard Generalized Markup Language (SGML)* was published. SGML is written in such a way as to define what a piece of information is. It allows for very clearly defined and marked text, so searching for a specific piece of information was simple. However, because of SGML's complexity, it is a difficult markup language to learn. To become an "expert" in SGML may take years of study. In addition, because of this complexity, SGML was impractical for uncomplicated documents.

Along came HTML, the first "offspring" of SGML developed to respond to the demand for a simpler markup language. HTML was developed to use the

grammar of SGML, but it is a much less complex markup language to learn. It uses a limited number of tags that can be written in either upper or lower case or in a combination of both. Rather than defining what a piece of information is, HTML defines how the information should appear.

But it's this simplicity and lack of structured rules that has resulted in "sloppy" markup. The unstructured format requires a lot of processing power to interpret. With the advent of the cellular telephone and the growing popularity of tools such as the handheld computer, HTML has become too cumbersome to process.

Another of SGML's offspring, XML (Extensible Markup Language), had many promising features, including its compatibility with SGML. XML was an easier language to learn and to use than its predecessor, SGML. Like SGML, information written in XML is defined as what type of information it is rather than how it should look. This allows Web users to search for specific information much more quickly. Although XML is a very structured language, it is extensible, meaning new tags could be developed by anyone. But there's one major problem, and that is that many of the older browsers still in use today can't read XML.

What was needed was a language that was easy to use and easy to learn that would also be compatible with old browsers. XHTML was developed to bridge the gap between the current HTML and the exciting possibilities of XML. XHTML is the result of combining the advantages of the "sibling languages," HTML and XML. XHTML uses the exact tags and elements of HTML and syntax of SGML, but has the structure of XML. XHTML is considered to be an application of XML.

A BRIEF LOOK AT HTML

Hypertext Markup Language (HTML) is the popular language used to write Web pages. But why talk about HTML in an XHTML book? A little background on HTML is important since XHTML is a reformulation of HTML 4 into an XML application. So let's take a quick look. The HTML commands used to generate a Web page are called tags and are contained in a plain ASCII text file. The browser interprets the markup tags, and the results are then displayed on your monitor. The following is an example of a simple HTML document:

```
<HTML>
  <HEAD>
    <TITLE>
      Simple HTML Document
    </TITLE>
  </HEAD>
  <BODY>
    <H1>
      Hello HTML World!
    </H1>
    <P>
      This is the first paragraph of text.
    <P>
      This is the second paragraph of text.
  </BODY>
</HTML>
```

The results of the above HTML are in Figure 2.1.

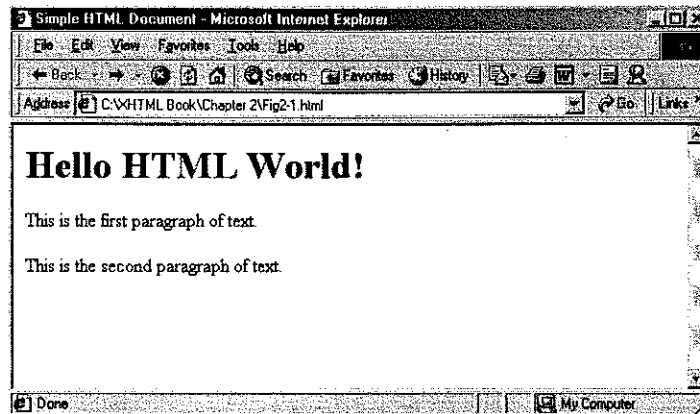


FIGURE 2.1: SIMPLE HTML DOCUMENT

Now that we have seen what a simple HTML document looks like and how the browser interprets the file to render a page on the screen, it's time to discuss some of the terms and syntax associated with HTML. The HTML file contains tags, which can be identified by the fact that they are enclosed in angle brackets, < and >. In the above example, we used the <HTML> tag to indicate that this is an HTML document and will enclose the entire document. The <HEAD> and <BODY> tags are used to create two necessary sections of an HTML document. The information enclosed between the <HEAD> </HEAD> tags will be information about the page that will not be displayed on the page itself.

The `<BODY>` `</BODY>` tags will enclose the material to be displayed on the Web page.

Tags can be used to identify structural elements of a document as well as to control the appearance of the Web page. There are two types of tags in HTML, container and stand-alone. A container tag is used to enclose some text, which is called content. Container elements have the following syntax:

```
<opening tag> text content </closing tag>
```

The opening tag indicates where the browser is to begin to mark up the text. The closing tag indicates where the formatting no longer applies. The closing tag will be the same as the opening tag except for the inclusion of a forward slash. The following is an example from the HTML document created above:

```
<H1>  
    Hello HTML World!  
</H1>
```

The `<H1>` tag is the opening tag followed by the content, while the `</H1>` is the closing tag. The opening tag, content, and closing tag are collectively referred to as an HTML element. In HTML, the `<H1>` tag is used to display a heading, which is large and bold. This tag also needs to have a closing tag `</H1>`, to indicate where the heading style is to end. Another tag used in our example is the paragraph tag `<P>`. This tag is used to start a new paragraph of text and to add some space between the previous paragraphs. In HTML, the closing tag for the `<P>` tag is optional. Our simple HTML contained the following two paragraphs:

```
<P>  
    This is the first paragraph of text.  
<P>  
    This is the second paragraph of text.
```

How does the browser know where to begin the second paragraph without a closing tag on the first paragraph? When the browser encounters the second paragraph tag without encountering a closing paragraph `</P>` tag for the first paragraph, it assumes that the first paragraph has ended and a new paragraph is to begin.

Using optional or required attributes can change the behavior of tags. Tag attributes are used to specify how the tag is to be processed by the browser. Attributes are added to the opening tag by using keywords that give the browser additional information on how to render the tag on the page. For example,

in our simple HTML document, if we wanted to center the heading on the page, we would include the align attribute and give it the value of "center." The opening heading tag would then look like the following:

```
<H1 align = CENTER>
  Hello HTML World!
</H1>
```

The result of adding the attribute to the <H1> tag is shown in Figure 2.2.

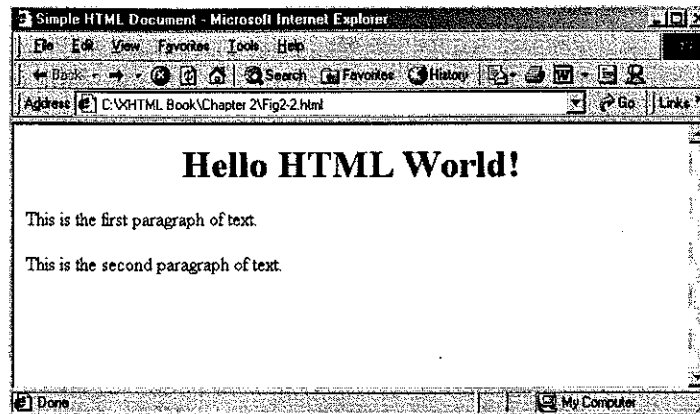


FIGURE 2.2: SIMPLE HTML DOCUMENT USING ATTRIBUTE

Tags are not limited to using only a single attribute; they can contain multiple attributes and values.

Over the years, HTML has gone through many versions, with each one adding features and depreciating others. For more information about the evolution of HTML, visit the *World Wide Web Consortium (W3C)* at <http://www.w3c.org>. The W3C organization is responsible for setting and maintaining the standards for HTML.



XHTML ADVANTAGES

Since XHTML follows the rules of XML, it has the advantages of *extensibility* and *portability*.

Extensibility

HTML has a fixed set of elements and has evolved very little over the years. If additional elements were required, the entire *document type definition (DTD)*

would need to be changed. Since XHTML is an application of XML, its functionality can be extended beyond its predefined elements. By using XML namespaces, new elements can be added without changing the DTD on which the document is based.

Portability

Currently, most Internet access is through the PC. This is changing rapidly as more devices are becoming Web-enabled. These devices have less power than PCs and have a problem rendering current HTML documents. XHTML makes Web documents accessible across platforms.

DIFFERENCES BETWEEN HTML AND XHTML

XHTML simply redefines HTML using XML rules. Although XHTML is just a redefinition of HTML 4, it provides a stricter implementation of those standards and was developed to bridge the gap between XML and HTML. By adopting the new standards, developers can create documents that work with current browsers as well as be processed by XML applications. There are several ways that XHTML documents differ from HTML documents:

- They must be well-formed
- Element and attribute names must be in lower case
- Non-empty elements require end tags
- Attribute values must always be quoted
- They have no attribute minimization
- They end empty elements
- They use elements with id and name attributes
- They use Document Type Definitions
- They use XML namespaces

WELL-FORMED DOCUMENTS

This concept was introduced in XML, and a document is considered well-formed when it conforms to the rules defined in the XML specification. For an XHTML document, this means all elements:

- Must have an opening and closing tag
- Must be properly nested
- If empty, must be written in a special form

Opening and Closing Tags

In XHTML, all tags are required to be closed, so a closing tag must accompany every open tag. This is a change from HTML, where some closing tags were optional. Some examples of such tags are `<p>`, `<td>`, and ``. For example, in HTML, we could use the `<p>` tag like:

```
<p>This was a legal use of this tag in HTML
<p>Second paragraph
```

In XHTML, we would close the `<p>` tag like

```
<p>This is a properly closed XHTML tag</p>
<p>With a second paragraph</p>
```



Properly Nested

Elements must nest properly, which means there can be no overlapping. Overlapping occurs when one element is not contained within the other. Although overlapping was technically illegal, HTML browsers would tolerate code such as:

```
<p>This is<b> bolded text</p></b>
```

Here is this text in XHTML, without overlapping elements:

```
<p>This is<b> bolded text</b></p>
```

tip: Remember to close every XHTML tag.

Empty Elements

In HTML, some elements are allowed to omit the closing tag because they do not enclose any content. Two common elements are the `<hr>` and `
`. For example, the following would be acceptable:

```
<hr>
<br>
```

In XHTML, all empty elements must have a closing tag. This can be accomplished through the use of a shortcut. This is an example of the correct way:

tip: Leave a space before the / character.

```
<hr />
<br />
```

tip: Remember to close every XHTML tag in the proper order.

ELEMENT AND ATTRIBUTE NAMES MUST BE LOWER CASE

XML is case-sensitive, which means that it would treat the tags <P> and <p> differently. In XHTML, all element and attribute names must use lowercase.

In HTML, the following could be used:

```
<A HREF="first.html" TARGET=MAIN>Link to the first page</A>
```

In XHTML, the same line would appear as follows:

```
<a href="first.html" target="main">Link to the first page</a>
```

ATTRIBUTE VALUES MUST ALWAYS BE QUOTED

In XHTML, all values assigned to attributes must be quoted, even numeric values. In HTML, the following could be used:

```
<table align = center width = 85% border = 2 height = 200>
```

In XHTML, the same line would appear as follows:

```
<table align= "center" width= "85%" border= "2" height= "200">
```

NO ATTRIBUTE MINIMIZATION

Attribute minimization (stand-alone attribute) is not supported by XML. HTML has several attributes that do not require a value.

In HTML, the following could be used:

```
<FRAME SRC="frame1.html" NORESIZE>
```

In XHTML, the same line would appear as follows:

```
<frame src="frame1.html" noresize="noresize"/>
```

USING ELEMENTS WITH ID AND NAME ATTRIBUTES

Another change under XHTML is the use of the name attribute, which is defined in HTML 4 for use with the elements <a>, <frame>, <iframe>, , and <map>. The name attribute has been deprecated, but some browsers do not support the id attribute. For compatibility, use both id and name attributes.

tip: Since future versions of XHTML will drop the name attribute, use both id and name attributes to ensure future compatibility.



In HTML, the following could be used:

```
<FRAME SRC="toc.html" NAME="toc_menu">
```

In XHTML, the same line would appear as follows:

```
<frame src="toc.html" id="toc_menu" name="toc_menu"/>
```

USE OF DOCUMENT TYPE DEFINITIONS

The W3C defines a DTD as "a collection of XML declarations that, as a collection, defines the legal structure, elements, and attributes that are available for use in a document that complies to the DTD." Simply put, it is going to define how the contents of the XHTML document are going to be displayed by the browser.

In an XHTML document, the DTD is referenced in the `<!DOCTYPE>` element, which is used to declare and define the type of the document. The DTD declaration must be at the top of the document. XHTML documents should reference one of the DTDs, which correspond to the three HTML 4 DTDs. The XHTML DTDs are strict, transitional, and frameset.

Strict DTD

Strict DTD is used for a document that will conform exactly to the XHTML 1.0 standard. This means that the document will not contain any deprecated or frame elements. It is used when presentational markup can be separated from document structure. This is typically used when formatting is being done with cascading style sheets (CSS). Here's an example of the strict DTD:

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

Transitional DTD

Transitional DTD is used when the document needs frames. For compatibility, it includes deprecated elements. Here's an example of the transitional DTD:

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

Frameset DTD

This DTD is used when presentational markup needs to be embedded in the document. This includes all of the elements, transitional in addition to the frame elements. Here's an example of the *frameset DTD*:

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

XML NAMESPACES

The W3C defines an *XML namespace* as “a collection of names, identified by a URI (uniform resource identifier) reference, which is used in XML documents as element types and attribute names.” Earlier in the chapter, we talked about the advantages of extensibility and portability. To make use of these advantages, we must declare an XML namespace that will indicate that it is an XHTML document and how it should be interpreted. The XHTML namespace declaration is an attribute in the <html> element. The XHTML namespace is:

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

XHTML DOCUMENTS

An XHTML document must meet the following criteria:

1. It must validate against one of the three DTDs.
2. The root element of the document must be <html> and designate the XHTML namespace by using the xmlns attribute.
3. A DOCTYPE declaration must be in the document and placed before the root element.

SIMPLE XHTML DOCUMENT

We start our simple XHTML document with the <?xml> declaration to indicate that the document is based on XML and what version. This is not required, but it is highly recommended. The following is our simple XHTML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD/XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Simple XHTML Document</title>
</head>
<body>
<h1>Hello XHTML World</h1>
</body>
</html>
```

The results of the above XHTML are in Figure 2.3.

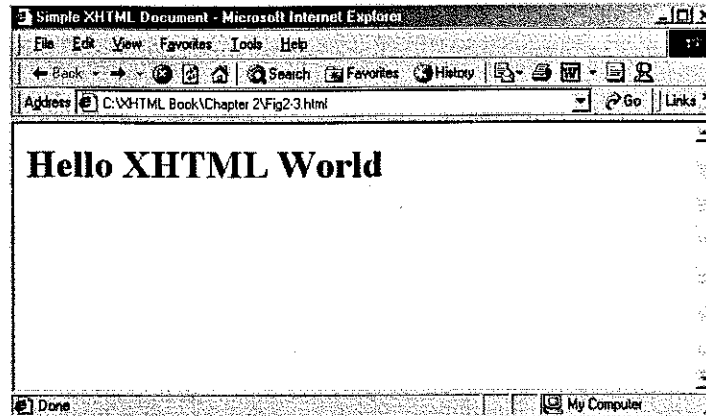


FIGURE 2.3: SIMPLE XHTML DOCUMENT

Before we can start learning about XHTML elements and tags that are used to create Web pages, we must discuss the tools that are available to aid in this development process also known as coding. Currently, there are many HTML development tools but very few XHTML tools.

APPROACHES TO CREATING XHTML DOCUMENTS

There are several ways that can be used to code an XHTML document. HTML tools can be used to create XHTML and then modified to XHTML standards, or we could just code our XHTML in a simple text editor. The following are four types of tools that could be used to create an XHTML file:

What You See Is What You Get (WYSIWYG) applications

- Tag-based editors
- Conversion applications
- Text editors

You are probably asking yourself, “Why would I use a text editor when I can use an application that generates the code for me?” The answer is simple: If you know XHTML, you can easily learn any development system that comes along, but the converse is not true. In the long run, people who know only how to use a Web development editor are limiting their job marketability. So it is important to learn the basics of XHTML in an ASCII editor and then move on to development systems for their added productivity and functionality when that is appropriate.

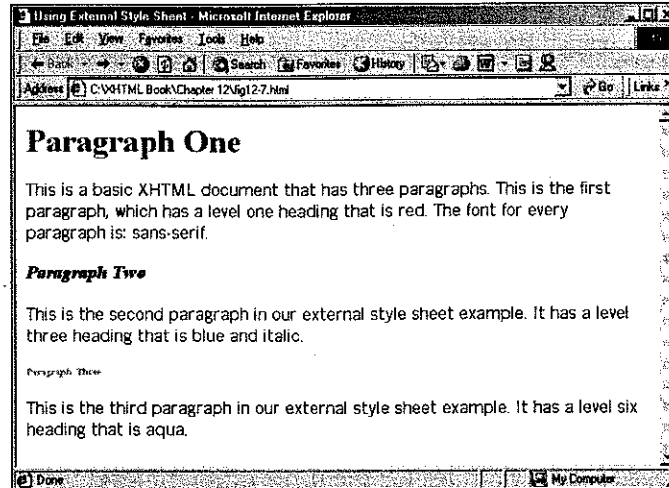


FIGURE 12.7: BASIC XHTML DOCUMENT USING EXTERNAL STYLE SHEET

CREATING A STYLE CLASS

A *style class* allows you to define different styles for the same element or for a group. This allows for different styles to be applied to parts of an XHTML document that are structurally the same. For example, you could define classes to enable the `<h1>` element to have different appearances instead of just the one defined. Without the use of classes, this would have to be done with inline styles. Style classes are defined within embedded or external style sheets using the following syntax:

```
.class name{
    property:value;
    property:value;
}
```

The class name is used to identify the class and is preceded by a period. When naming the class, it is a good practice to give it a name that represents function and not style. The following is an example of defining classes in the style element:

```
<style type="text/css">
<!--
  hl.go{
    color: green;
```

```

    }
    h1.caution{
        color: yellow;
    }
    .stop{
        color: red;
    }
    h1{
        font-size:16pt
    }
    p{
        font-style: italic;
        color: black;
    }
-->
</style>

```

In this example, there are two classes targeted specifically at a level-one heading and can be used only with them. These two classes are `go` and `caution`. Notice that the class `stop` does not have an XHTML element associated with it. This allows the class to be used with any XHTML element that can display color. Let's apply the classes that we created to the following XHTML document using the class attribute:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD/XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title> Using Classes With Style Sheets </title>
<style type="text/css">
    <!--
        h1.go{
            color: green;
        }
        h1.caution{
            color: yellow;
        }
        .stop{
            color: red;
        }
        h1{
            font-size:16pt
        }
        p{
            font-style: italic;
            color: black;
    -->

```

```

-->
</style>
</head>
<body>
  <h1 class="go">This heading is green because it uses the go
  class.</h1>
  <h1 class="caution">This heading is yellow because it uses the
  caution class.</h1>
  <h1 class="stop">This heading is red because it uses the stop
  class.</h1>
  <p>
    Since the stop class was defined with no associated XHTML
    element, I can use it with the span element to change
    the color of some text to red. <span class="stop">This text
    is red and italic.</span>
  </p>
</body>
</html>

```

The results of the above XHTML are shown in Figure 12.8.

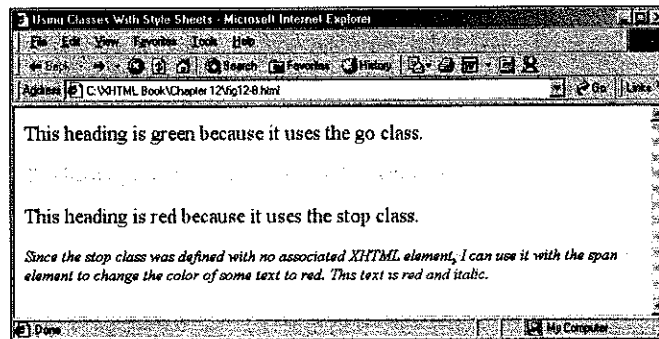


FIGURE 12.8 BASIC XHTML DOCUMENT USING CLASSES

USING STYLE SHEET PROPERTIES

So far in this chapter, we have applied some simple style rules to XHTML documents using style properties that affected color, font size, and font type. There are a great many more properties that could be used, so many that it just would not be practical to cover them in a single chapter. The next part of this chapter will look at the most common style sheet properties and how they can be used.

The properties that are used with style sheets can be divided into two groups: text-level and block-level. Text-level properties would cover color, fonts, space, size, and positioning. Block-level properties deal more with page layout and include borders, margins, text alignment, and indents.

Setting Color

The *color* property has been used in previous examples to change the color of text. The value of this property can be specified using one of 16 color keywords. The 16 standard color names include

aqua	gray	navy	teal
black	green	olive	silver
blue	lime	purple	white
fuchsia	maroon	red	yellow

You are not limited to the 16 standard colors; the hexadecimal color value can also be used to specify colors. When using the hexadecimal value, it is preceded by the # sign. Below are the same 16 standard color names with their hexadecimal values:

aqua	#00FFFF	gray	#808080	navy	#000080	teal	#008080
black	#000000	green	#008000	olive	#808000	silver	#C0C0C0
blue	#0000FF	lime	#00FF00	purple	#800080	white	FFFFFF
fuchsia	#FF00FF	maroon	#800000	red	#FF0000	yellow	FFFF00

The following are examples of style rules using the color property to change the color of XHTML elements:

```
h1{color: red;}
h3{color: blue;}
h6{color: aqua;}
p{color: #0000FF;}
```

Setting Background Color

The *background-color* property is used to set the background color of an element. Although you can define a background color for any element, it is normally used to define a background color for the body element. The color values used with the color property also apply to this property. The following are examples of the style rules using the background-color property with XHTML elements:

```
body{background-color: white;}
h1{background-color:#FF0000;}
```

Figure 12.9 is the result of applying the following style rules to a simple XHTML document:

```
body{background-color:gray;}
h1{color:blue; background-color:white;}
p{color:white; font-family: sans-serif;}
```

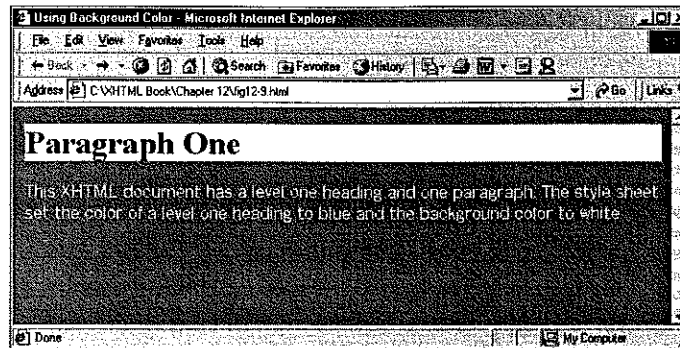


FIGURE 12.9: SIMPLE XHTML DOCUMENT USING BACKGROUND-COLOR PROPERTY

Controlling Font Appearance

Cascading style sheets have a wide range of properties to specify the appearance of a font in an XHTML document. The commonly used cascading style sheet properties used to control fonts are

- font-family
- font-size
- font-style
 - line-height
- font-weight

font-family

Cascading style sheets allow for the naming of a specific font to be used for text. The *font-family* property can be assigned a specific font name or generic font family as its value. The only catch is that the specified font must be available on the computer viewing the page. If the specified font is not available, the default font will be used. The following is an example of a CSS rule using the *font-family* property:

```
p{
  font-family: sans-serif;
}
```

The font-family property can also have multiple family assignments made, which allows for a list of alternatives in case the first choice of font is not available. The list of alternatives is separated by commas and is in order of preference. The browser will start with the first value of the font-family property and work its way to the last. The last value in the list should be a generic font-family. The following is an example of a CSS rule using the font-family property with multiple values:

```
p{
  font-family: helvetica, arial, sans-serif;
}
```

When using a font name that contains multiple words, enclose the name in quotes. For example:

```
p{
  font-family: "century schoolbook", times, serif;
}
```

The following are the five font families recognized by cascading style sheets:

- serif
- sans-serif
- monospace
- script
- fantasy

Figure 12.10 illustrates the different font families.

This is the Sans Serif Font Family
This is the Serif Font Family
This is the Script Font Family
This is the Monospace Font Family
THIS IS THE FANTASY FONT FAMILY

FIGURE 12.10 BASIC FONT FAMILIES

font-size

The *font-size* property can be used to control the size of the font displayed by the browser. The assigned value can be either absolute or relative. The absolute values are based on standard units of measurement such as pixels (px), points (pt), inches (in), centimeters (cm), or picas (pc). A relative value is expressed as a value relative to the size of the parent element. The most common way to express a relative value is by using a percentage. The following are examples of CSS rules using the *font-size* property:

```
h1{
  font-size:16pt;
}
p{
  font-size:120%;
}
```

Figure 12.11 illustrates the effect of the *font-size* property.

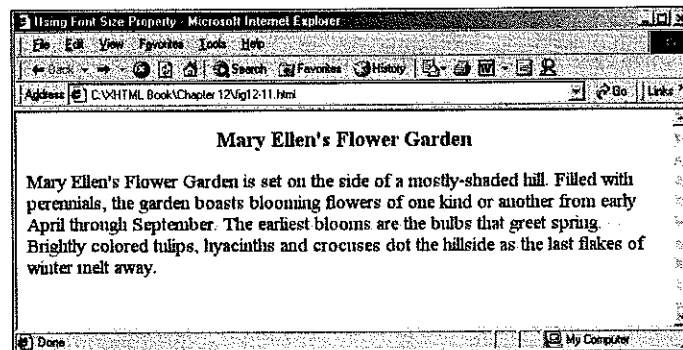


FIGURE 12.11: FONT-SIZE PROPERTY APPLIED

font-style

The *font-style* property defines the appearance of a font in one of three ways: normal, italic, or oblique. The italic and oblique (slanted) are very similar in appearance. The italic style is available in most fonts and should be used over the oblique option. The following is an example of a style sheet declaration using the *font-style* property:

```
p{
  font-style: italic;
}
```

font-weight

The *font-weight* property is used to control the line thickness (lighter or bolder) of the text. The weight of a font can be specified by using a number representing how dark it should be or by using a keyword. When using a number to specify the weight, the value is a number ranging from 100, being the lightest, to 900, being the darkest. The number increases in intervals of 100. The weight of 400 for most fonts is normal, while a weight of 700 is considered bold. The keywords bold, bolder, lighter, and normal can also be used to specify the weight of a font. The following are examples of style sheet declarations using the font-weight property:

```
h1{
  font-weight:200;
}
h3{
  font-weight:bold;
}
h6{
  font-weight:900;
}
```

Figure 12.12 illustrates the effect of the font-weight property.

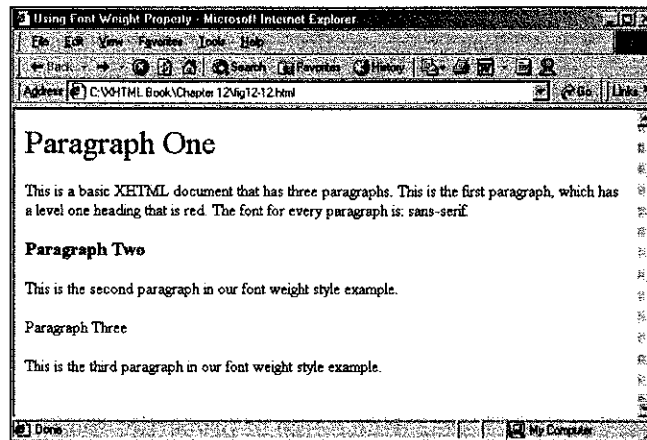


FIGURE 12.12: FONT-WEIGHT PROPERTY APPLIED

line-height

The *line-height* property refers to how much vertical space is displayed between lines of text in a paragraph or other element. The normal vertical space is directly related to the size of the font, but it can be modified by using the *line-height* property. The value assigned to this property can be expressed in

- a percentage of font size
- a specific numeric value specified by a measurement in points, inches, pixels, or centimeters

The following are examples of style sheet declarations using the *line-height* property:

```
h1{
    line-height:14pt;
}
p{
    line-height:150%;
}
```

Figure 12.13 illustrates the effect of the *line-height* property.

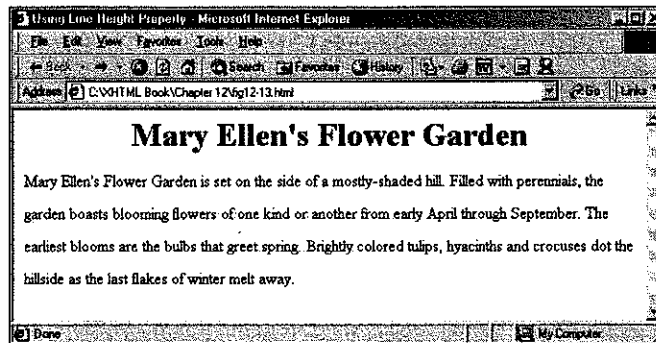


FIGURE 12.13 FONT-HEIGHT PROPERTY APPLIED

Using Block-level Properties to Control Layout

Cascading style sheets have a wide range of properties to improve the layout of an XHTML document. CSS handles block-level properties based on the *box model*, which is shown in Figure 12.14 as a graphical representation. The concept is based on the fact that every block-level element is displayed in a box and could have a box within it. This includes the body element; the browser

will create a box for it inside its containing block, which would be the browser window. At the center of every box created by an element is the content; this is what is going to be displayed. The area that surrounds the content is called the *padding*, and if a background color or image were present, it would extend into this area. The border surrounds the padding and is a line around the contents. The *margin* surrounds the border and is the space between any other boxes that may be around it. The margins of a box are always transparent. This allows the background color of the body to be seen.

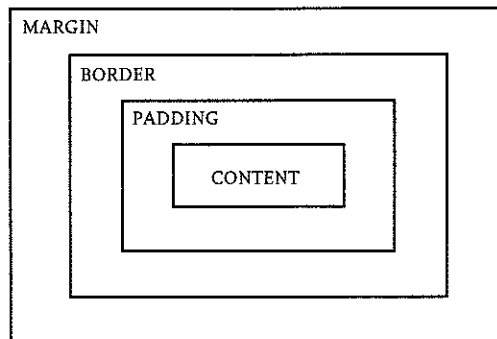


FIGURE 12.14: BOX MODEL

To get a better understanding of the box model, let's look at Figure 12.15, which shows a graphical representation of the following simple XHTML code:

```
<body>
  <p>
    This is a paragraph in an XHTML document. This is the
    content of the paragraph element.
  </p>
</body>
```

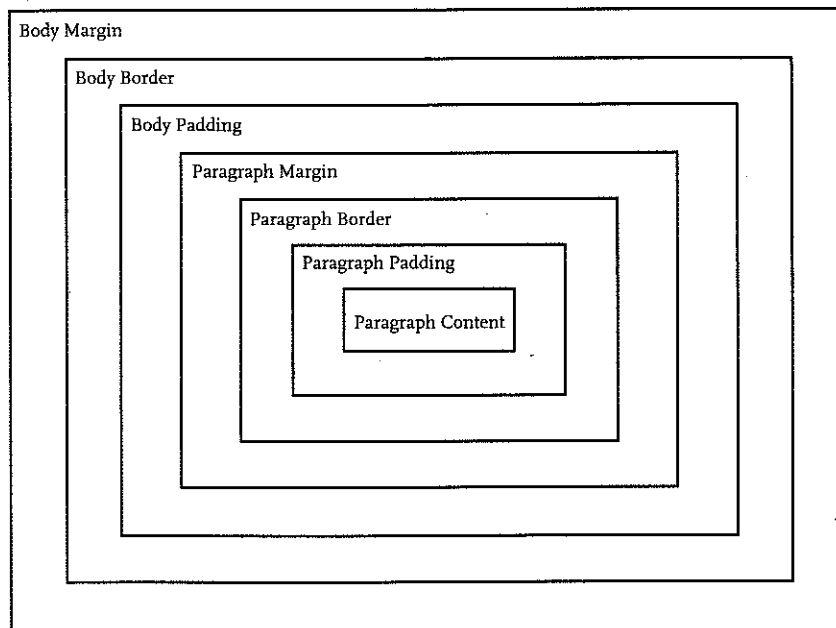


FIGURE 12.15: BOX MODEL USING XHTML EXAMPLE

The commonly used cascading style sheet properties used to improve layout are

- margin-top
- margin-right
- margin-bottom
- margin-left
- padding-top
- padding-right
- padding-bottom
- padding-left
- text-align
- text-indent
- text-transform

Applying Margin Properties

The CSS properties of *margin-top*, *margin-bottom*, *margin-left*, and *margin-right* describe the sides of the margin box. These properties allow for the control of

space between the block-level element and the surrounding element. Margin sizes are set using units of lengths expressed as pixels, points, inches, centimeters, or a percentage value. The following is an example of a style sheet with a declaration using the margin properties:

```
body{  
    margin-left:10%; margin-right:10%;  
}
```

Using the above style sheet declaration, Figures 12.16 and 12.17 illustrate the effect of the margin properties.

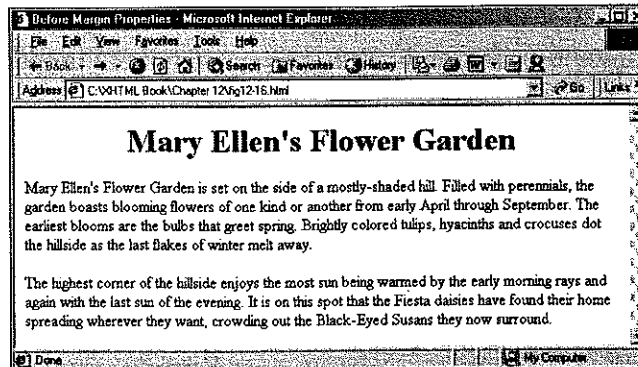


FIGURE 12.16: DOCUMENT BEFORE MARGIN PROPERTIES APPLIED

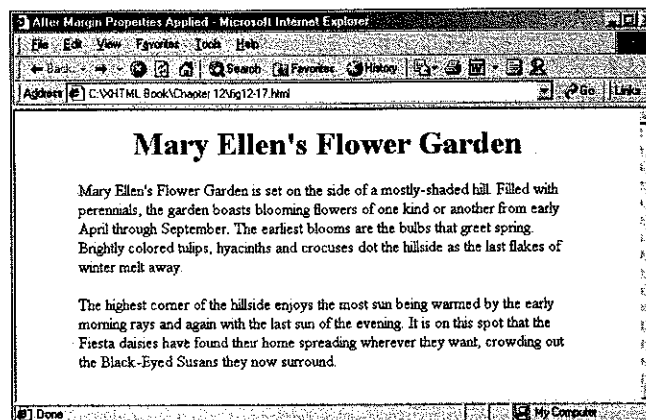


FIGURE 12.17: DOCUMENT AFTER MARGIN PROPERTIES APPLIED

When a percentage value is used, as in the previous example, the percentage is relative to another length value; in this case, since we are working with the body element, it would be the browser window. The value assigned to the margin size can also be expressed as a negative number. The margin-top and margin-bottom properties can also be used to control the space above and below elements such as headings. The following is an example of a style sheet declaration using the top and bottom margin properties on a level-one heading:

```
h1{
  margin-top:10%; margin-bottom:10%
}
```

Using the above style sheet declaration, compare Figures 12.17 and 12.18, which illustrate the effect of the margin top and bottom properties.

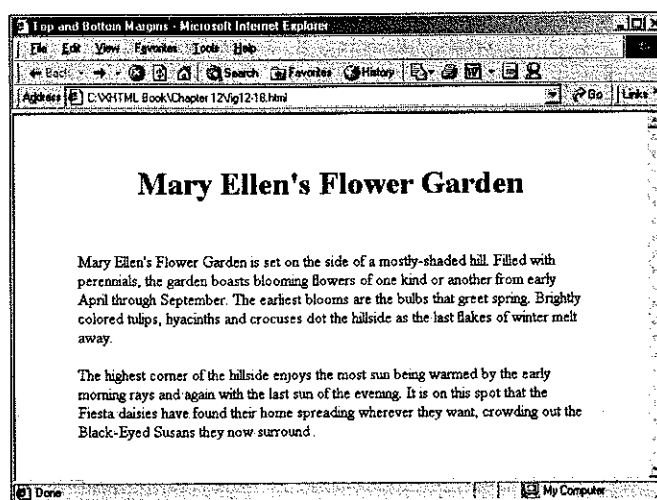


FIGURE 12.18: DOCUMENT AFTER MARGIN TOP AND BOTTOM PROPERTIES APPLIED

Applying Padding Properties

The CSS properties of padding-top, padding-bottom, padding-left, and padding-right describe the sides of the padding box. These properties allow for the control of space

tip: By using a relative value such as a percentage, if the browser window is resized, the margins and padding will adapt to the page's new proportions.

between the border and the content of an element. *Padding* values are set using units of lengths expressed as pixels, points, inches, centimeters, or as a percentage value. The following is an example of a style sheet with a declaration using the padding properties:

```
td{
  padding-left:20pt;
}
```

Padding properties are useful with cells of a table. Figure 12.19 shows an example of a table from an earlier chapter. Notice how each cell has no space between the left border and the contents. Figure 12.20 shows the table after the above style sheet declaration has been added to an embedded style sheet.

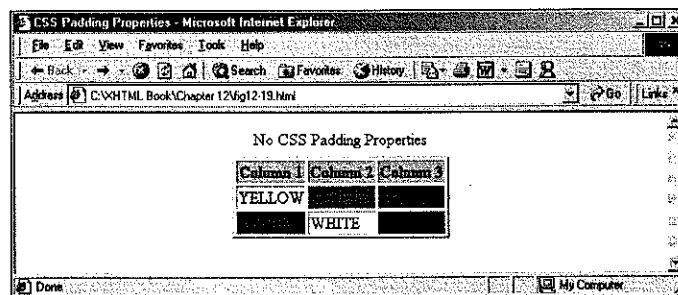


FIGURE 12.19: TABLE BEFORE PADDING PROPERTIES APPLIED

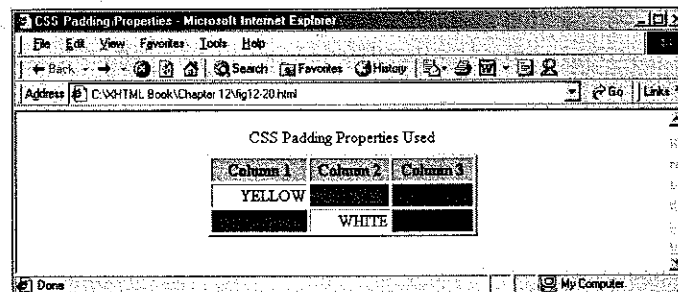


FIGURE 12.20: TABLE AFTER PADDING PROPERTIES APPLIED

Padding can also be added by using the padding property by itself, followed by up to four values, which are used to set padding on each side. For example:

```
td{
  padding:10pt 10pt 10pt 10pt;
}
```

Figure 12.21 shows the table example after the above style sheet declaration has been applied. Notice the space that has been added around each of the cell contents.

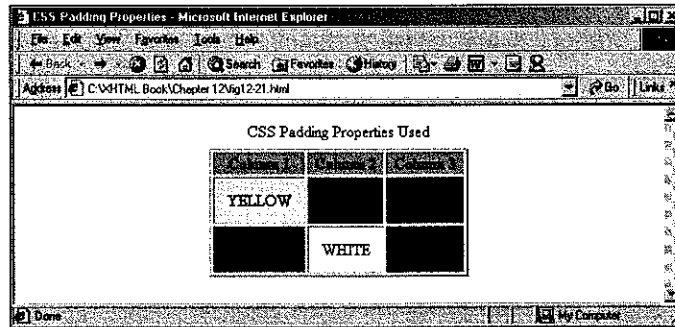


FIGURE 12.21 TABLE AFTER PADDING PROPERTY WITH FOUR VALUES APPLIED

The padding property can take one to four values. The following example is equivalent to the declaration shown above with four values:

```
td{
  padding:10pt;
}
```

The number of padding property values is interpreted in the following way:

- One value—indicates that the value given will be applied to all sides
- Two values—indicates that the first value will be applied to the top and bottom while the second value will be applied to the left and right
- Three values—indicates that the first value will apply to the top, the second value will apply to the left and right, and the third will be applied to the bottom
- Four values—indicates that the values will be applied to the top, right, bottom, and left, respectively

Applying Text-align Property

The CSS *text-align* property can be applied to block-level elements and allows for the justification of text within the element. The values used to align text within

an element include left, center, and right. The following is an example of a style sheet with a declaration using the text-align property:

```
h1{
    text-align:center;
}
```

Applying Text-indent Property

The CSS *text-indent* property can be applied to block-level elements and is used to set the amount of indentation of the first line of text within the element. The value used to define the amount of indentation can be expressed as an absolute or relative value. The following is an example of a style sheet with a declaration using the text-align property:

```
p{
    text-indent:5%;
}
```

Figure 12.22 shows an example of the above style sheet declaration applied to an XHTML document.

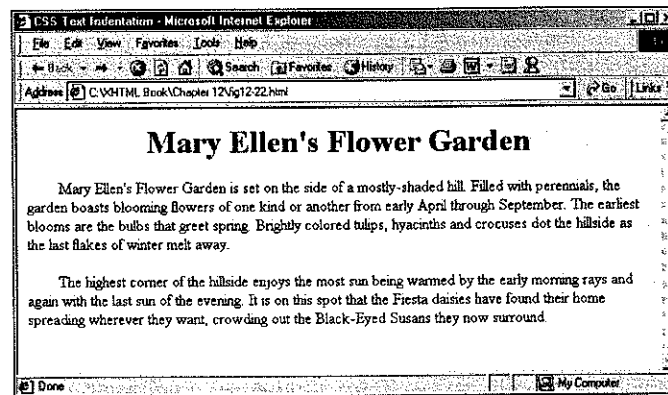


FIGURE 12.22: PARAGRAPHS AFTER TEXT-INDENT PROPERTY APPLIED

Applying Text-transform Property

The CSS *text-transform* property can be used to change the case of text. The text is transformed based on the following values:

- capitalize—first character of each word is capitalized
- lowercase—all characters will become small letters
- uppercase—all characters will be capitalized

The following are examples of style sheet declarations applying the text-transform property to the <h1> and elements:

```
h1{
    text-transform:capitalize;
}
span{
    text-transform:capitalize;
}
```

Figure 12.23 shows an example of the above style sheet declarations applied to an XHTML document.

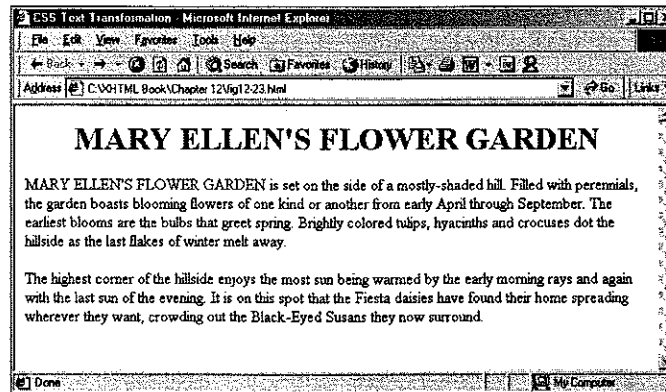


FIGURE 12.23: PARAGRAPHS AFTER TEXT-TRANSFORM PROPERTY APPLIED

CHAPTER SUMMARY

Cascading style sheets allow you to take more control over the presentation of a Web page. This chapter introduced style sheet concepts and how to apply them using inline styles, embedded styles, and external styles. With this language, we can control the text size, color, and font, as well as the background color and white space around a particular element. We learned how to apply CSS properties to block-level elements. Some of the advantages to using style sheets include: