Understand Basics of XML

This free book is provided by courtesy of C# Corner and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. Please do not reproduce, republish, edit or copy this book.

Mahesh Chand

Founder C# Corner

Microsoft MVP, Software Architect, Author

Sept 2013, Garnet Valley PA

Sam Hobbs

Editor, C# Corner

TODAY, Mobile and Web applications have taken over the programming world. No matter where you go, you are always consuming data with your smartphone, tablet or laptop. Data is being transferred from one place to another, from one hardware device to another and from one person to another. Do you ever wonder how this data is being transferred and consumed? Do you wonder what technology is being used to do so? You can blame XML for that. Well, partially at least!

The Extensible Markup Language (XML) has become an essential part of Web-based and mobile programming. XML is not only a way to store data but also a data transfer mechanism.

This book is a basic introduction to XML for beginners who have never used XML before. After completing this book, you will understand:

- Basic definitions of HTML and XML
- XML documents
- How to build an XML document
- XML document items

Table of Contents

XML Related Terminology

Before you can work with XML in C# or .NET, it's important you understand the basic building blocks of XML and its related terminology. You'll learn the basic definitions of Standard Generalized Markup Language (SGML) and HTML in the following sections. If you're already familiar with these languages, you can skip to the "XML Overview" section.

Standard Generalized Markup Language (SGML)

In 1986, the Standard Generalized Markup Language (SGML) became the international standard for representing word processing electronic documents in a unified way. SGML provides a standard format for designing your own markup schemes. **Markup** originally referred to symbols and other annotations editors use to indicate changes to a document and the term now also refers to a way to represent information about data. Standard Generalized Markup Language was originally based on IBM's Generalized Markup Language (GML), which is/was a set of macros for their Script word processing software. The "ML" in GML originally stood for "Mosher Lorie" since GML was originally developed by Charles Goldfarb, Edward Mosher and Raymond Lorie.

Later Hypertext Markup Language (HTML) became the international standard for representing documents on the Web in a unified way.

XML and versions of HTML prior to HTML Version 5 are "application profiles" of SGML. Therefore all XML documents and HTML documents prior to HTML Version 5 are SGML documents defined by a SGML standard application profile (one for each of them). SGML documents, including XML and HTML, are composed of many pieces of text called entities. SGML documents consist of markup tags that normally begin with a "<" and end with a ">" and that norm applies to XML and HTML documents.

Hyper Text Markup Language (HTML)

The HTML file format is a text format that contains, rather heavily, Markup tags. A tag is a section of a program that starts with < and ends with > such as <name>. (An **element** consists of a pair of tags, starting with <name> and ending with </name>). The language defines all of the markup tags. All browsers support HTML tags, which tell a browser how to display the text of an HTML document. You can create an HTML file using a simple text editor such as Notepad. After typing text in a text editor, you save the file with a .htm or .html extension.

NOTE: An HTML document is also called HTML pages or HTML file.

Listing 6-1 shows an example of an HTML file. Try typing the following into a text editor, and saving it as myfile.htm.

Listing 6-1. A simple HTML file

```
<html>

<head>
<title> A Test HTML Page </title>
</head>

<body>
Here is the body part.
</body>

</html>
```

If you view this file in a browser then you'll see the text "Here is the body part.". In Listing 6-1, your HTML file starts with the <html> tag and ends with the </html> tag. The <html> tag tells a browser that this is the starting point of an HTML document. The </html> tag tells a browser that this is the ending point of an HTML document. These tags are required in all proper HTML documents. The <head> tag is header information of a document and is not displayed in the browser. The <body> and</body> tags, which are required, identiy the main content of a document. As you can see, all tags ends with a <\> tag.

NOTE: HTML tags are not case sensitive. However, the World Wide Web Consortium (W3C) recommends using lowercase tags in HTML4. The next generation of HTML, XHTML, doesn't support uppercase tags. (The W3C promotes the web worldwide and makes it more useful. You can find more information on the W3C at http://www.w3c.org.)

Tags can have *attributes* that provide additional information about the tags. Attributes are part of the starting tag. For example:

```
<table border ="0">
```

In this example the <table> tag has an attribute, border, and its value is 0. This value applies to the entire <table> tag, ending with the </table> tag. Table 6-1 describes some common HTML tags.

Table 6-1 Common HTML Tags

TAG	DESCRIPTION
<html>	Indicates start and end of an HTML document
<title>	Contains the title of the page
<body>	Contains the main content, or body, of the page
<h1…h6>	Creates headings (from level 1 to 6)
<p>	Starts a new paragraph
 	Insert a single line break
<hr>	Defines a horizontal rule
<!-->	Defines a comment tag in a document
<b>	Defines bold text
<I>	Defines italic text
<Strong>	Defines strong text
<table>	Defines a table
<tr>	Defines a row of a table
<td>	Defines a cell of a table row
<font>	Defines a font name and size

There are other tags beyond those described in Table 6-1. In fact the W3C's HTML 4 specification is quite extensive. However, discussing all of the HTML tags is beyond the scope of this article. Before moving to the next topic, you'll take a look at one more HTML example using the tags discussed in the table. Listing 6-2 shows you another HTML document example.

Listing 6-2. HTML tags and their usage

```
<html>
<head>
<title> A Test HTML Page</title>
</head>
<!- – This is a comment - ->
<body>
<h1> Heading 1</h1>
<h2 Heading 2</h2>
<p><b><i><font size = "4">Bold and Italic Text. </font></i></b></p>
<table border = "1" width ="43%">
<tr>
<td width = "50%">Row1, Column1</td>
```

<td width = "50%">Row1, column2</td>
</tr>
<tr>
<td width = "50%"> Row2, Column1</td>
<td width = "50%"> Row2, Column2</td>
</tr>
</table>
</body>
</html>

NOTE: In Listing 6-2, the <font> and <td> tags contain size and width attributes, respectively. The size attribute tells the browser to display the size of the font, which is 4 in this example, and the width attribute tells the browser to display the table cell as 50 percent of the browser window.

XML Overview

I'll now cover some XML-related terminology. So what exactly is XML? XML is the acronym for Extensible Markup Language. If you've ever gotten your hands dirty with HTML, then XML will be a piece of cake.

Unlike HTML, XML not only stores data but also used in data exchanges mechanism between two applications. You can create separate XML files to store data, which can be used as a data source for HTML and other applications.

You'll now see an XML example. Listing 6-3 shows a simple XML file: books.Xml. By default, this file comes with Visual Studio (VS).NET if you have VS .NET or the .NET Framework installed on your machine; you probably have this file in your samples folder.

You'll create this XML file called books.xml, which will store data about books in a bookstore. You'll create a tag for each of these properties, such as a <title> tag that will store the title of the book and so on.

You can write an XML file in any XML editor or text editor. Type the code shown in listing 6-3 and save the file as books.xml.

This file stores information about a bookstore. The root node of the document is <bookstore>. Other tags follow the <bookstore> tag, and the document ends with the </bookstore> tag. Other tags defined inside the <bookstore> tag are <book>, <title>,

<author>, and <price>. The tags store information on the store name, book publication date, book ISBN number, book title, author's name and price.

Listing 6-3. Your first XML file sample

```
<?xml version ='1.0'?>
<bookstore>
<book>
<title>The Autobiography of Benjamin Franklin</title>
<author>
<first-name>Benjamin</first-name>
<last-name>Franklin</last-name>
</author>
<price>8.99</price>
</book>
<book>
<title> The Confidence Man</title>
<author>
<first-name>Herman</first–name>
<last-name>Melville</last-name>
</author>
<price>11.99</price>
</book>
<book>
<title>The Gorgias</title>
<author>
<name>Plato</name>
</author>
<price>9.99</price>
</book>
</bookstore>
```

The first line of an XML file looks like this: <? Xml version ="1.0"? >. This line defines the XML version of the document. This tag tells the browser to start executing the file. You may have noticed that <?> doesn't have an ending </?> tag. Like HTML, other tags in an XML document start with < and are followed by a /> tag. For example, the <title> tag stores the book's title like this: <title>The Gorgias</title>.

In Listing 6-3, <bookstore> is the root node. Every XML document must start with a root node with the starting tag and end with the root node ending tag; otherwise the XML passer gives an error. (I'll discuss XML parsers shortly.)

Now, if you view this document in a browser, the output looks like Listing 6-4.

Listing 6-4. Output of books.xml in the browser

```
<?xml version="1.0" ?>
  -<bookstore>
  -<book>
    <title>The Autobiography of Benjamin Franklin</title>
  -<author>
    <first-name>Benjamin</first-name>
    <last-name>Franklin</last-name>
    </author>
    <price>8.99</price>
    </book>
  -<book>
    <title>The Confidence Man</title>
  - <author>
    <first-name>Herman</first-name>
    <last-name>Melville</last-name>
    </author>
    <price>11.99</price>
    </book>
  -<book>
    <title>The Gorgias</title>
  - <author>
    <name>Plato</name>
    </author>
    <price>9.99</price>
    </book>
    </bookstore>
```

Your browser recognizes the XML and colors it appropriately.

Important Characteristics of XML

There are a few things you need to know about XML. Unlike HTML, XML is case sensitive. In XML, <Books> and <books> are two different tags. All tags in XML must be well formed and must have a closing tag. A language is **well formed** only if it follows exact language syntaxes the way they are defined.

Improper nesting of tags in XML won't the document property. For example:

Bold and Italic Text.

is not well-formed. The well- formed version of the same code is this:

Bold and Italic Text.

Another difference between HTML and XML is that attributes must use double quotes in XML. XML attributes function like HTML attributes and are extra information you can add to a tag. Providing data as an attribute can be an alternative to providing the data as a sub-element and sometimes the descision of which to use is unclear. (I'll discuss attributes in the "An XML Document and its Items" section later in this article.) Having attributes without double quotes is improper in XML. For example, Listing 6-5 is a correct example of using the attributes ISBN, genre, and Publication date inside the <book> tag.

Listing 6-5 Attributes in XML files

```
?xml version ='1.0'?>
<!-- This file represents a fragment of a book store inventory database -->
<bookstore>
<book genre = "autobiography" publicationdate = "1981" ISBN ="1-861003-11- 0">
<title>The Autobiography of Benjamin Franklin</title>
<author>
<first-name>Benjamin</first-name>
<last-name>Franklin</last-name>
</author>
<price>8.99</price>
</book>
</bookstore>
```

The genre, publicationdate, and ISBN attributes store information about the category, publication date, and ISBN number of the book, respectively. Browsers won't have a

problem parsing the code in listing 6-5, but if you remove the double quotes the attributes like this:

<book genre = autobiography publicationdate = 1981 ISBN =1-861003-11-0>

then the browser will give the error message shown in Figure 6-1.

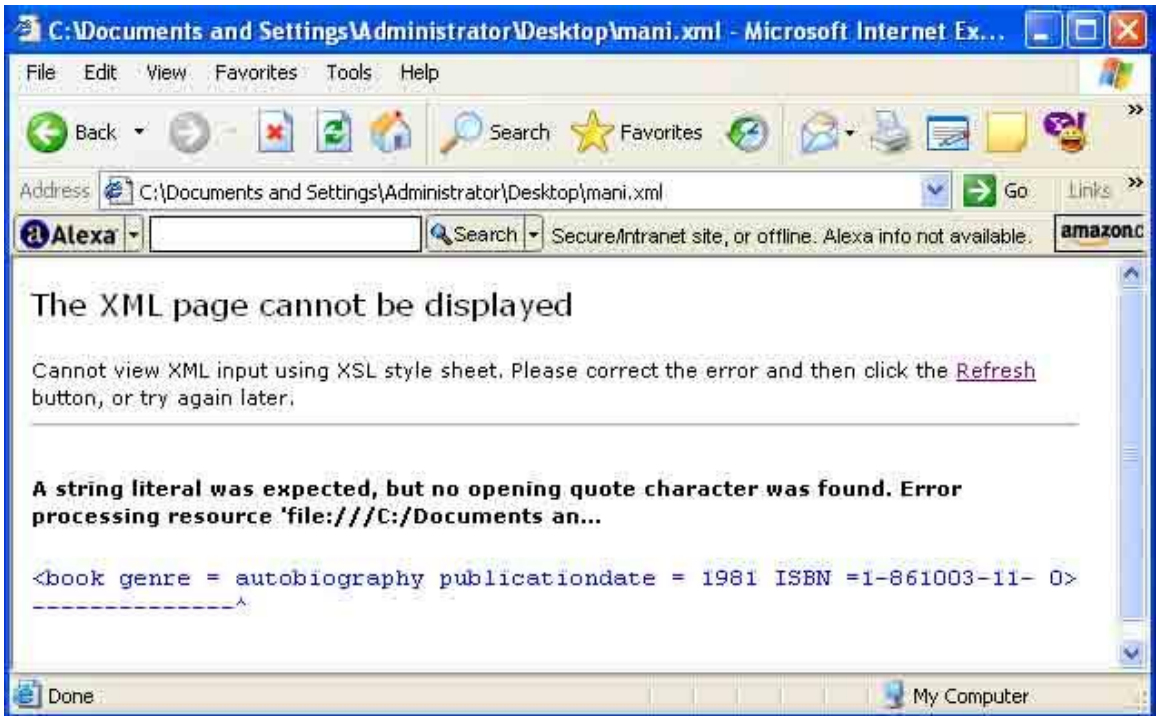

Figure 6-1. XML attribute definition error message

Another character you might notice in Listing 6-5 is the ! - -, which represents a comment in XML documents. (I'll cover comments in a moment. See the "Comments" section.)

Unlike HTML, XML preserves spaces, which means you'll see the white space in your document displayed in the browser.

XML Parser

An XML parser is a program that sits between XML documents and the application using the document. The job of a parser is to make sure the document meets the defined structures, validation, and constraints. You can define validation rules and constraints in a Document type Definition (DTD) or schema.

An XML parser comes with Internet Explorer (IE) 4 or later and can read XML data to process it, generate a structured tree, and expose all data elements as DOM objects. The parser then makes the data available for further manipulation through scripting. After that, another application can handle this data.

The MSXML parser comes with IE 5 or later and resides in the MSXML.DLL library. The MSXML parser supports the W3C XML 1.0 and XML DOM recommendations, DTDs, schemas, and validations. You can use MSXML programmatically from languages such as JavaScript, VBScript, Visual Basic, Perl, and C++.

Universal Resource Identifier (URI)

A Universal Resource Identifier (URI) is a resource name available on the Internet. A URI contains three parts: the naming schema (a protocol used to access the resource), the name of the machine (in the form of an Internet Protocol) upon which the resource resides, and the name of the resource (the file name). For Example, http://www.csharpcorner.com/Images/cshea1.gif is a URI name where http:// is a protocol, www.csharpcorner.com is the address of the machine (which is actually a conceptual name for the address), and Images/afile.gift is the filename location on that machine.

XML Namespaces

Because users define an XML document's element names, it's possible that many developers will use the same names. XML **namespaces** allow developers to write a unique name and avoid conflicts among element names with other developers. With the help of a URI, a namespace ensures the uniqueness of XML elements, tags, and attributes. For the purpose of XML namespaces, URIs can be anything and do not need to represent a true internet address; URIs are simply a conventional way to provide a unique namespace name.

To declare namespaces, you can use default or explicit names. When you define your own namespace, the W3C recommends you control the URI and point to the same location consistently.

The scope of a document's elements depends on the URI. Listing 6-6 shows an example of a XML document with a namespace. In this example, <book> and its attributes and tags belong to the http://www.c-sharpcorner.com/Images URI.

Listing 6-6. XML namespace declaration example

```
<?xml version ='1.0'?>
<book xmlns = "http://www.c-sharpcorner.com/Images" >
<title> the autobiography of Benjamin Franklin</title>
<author>
<first-name>Benjamin</first-name>
<last-name>Franklin</last-name>
</author>
<price>8.99</price>
</book>
```

Document type Definition (DTD) and schemas

A Documents Type Definition (DTD) defines a document structure with a list of legal elements. You can declare DTDs inline or as a link to an external file. You can also use DTDs to validate XML documents. This is an example of a DTD:

```
<!ELEMENT Two (#PCDATA)>
<!ELEMENT one (B)>
<!ATTLIST one c CDATA # REQUIRED>
```

This DTD defines a format of some data. The following XML is valid because the tag <Two> is inside the tag <one>:

```
<One c="Attrib">
<Two> Something here</Two>
</One>
```

An XML *schema* describes the relationship between a document's elements and attributes. XML schemas describe the rules that can be applied to any XML document, for elements and attributes. If an XML document references a schema and it doesn't meet the criteria then the XML parser will give an error during parsing.

You need a language to write schemas. These languages describe the syntaxes for each schema (XML document) you write. There are many schema languages, including DTD, XML Data Reduced (XDR), and simple object XML (SOX).

Similar to an XML document, an XML schema always starts with statement <?xml version ="1.0" ?>, which specifies the XML version.

The next statement of a schema contains an xsd:schema statement, xmlns, and target namespace. The xsd: schema indicates that the file is a schema.

A schema starts with a <xsd:schema> tag and ends with a </xsd:schema> tag. All schema items have the prefix xsd. The xmlns =”http://www.w3.org/2001/XMLschema” is a http://www.W3c.org URI, which indicates the schema should be interpreted according to the default, namespace of the W3C. The next piece of this line is the target namespace, which indicates the location of a machine (a URI). Listing 6-7 is a schema representation of the document in Listing 6-5.

Listing 6-7. XML schema example

```
<xsd:schema xmlns:xsd ="http://www.w3.org/2001/XML Schema">

<xsd:element name = "bookstore" type = "bookstoreType"/>

<xsd: ComplexType name ="bookstoreType">
<xsd: squence maxOccurs = "unbounded">
<sdx: element name = "book" type = "bookType"/>
</xsd: sequence>
</xsd: complexType>

<xsd: ComplexType name = "bookType">
<xsd: sequence>
<xsd: element name = "title" type = "xsd:string:"/>
<xsd: element name = "author" type = "authorName"/>
<xsd: element name = "price"  type = "xsd:decimal"/>
</xsd: sequence>
</xsd: attribute name = "genre" type = "xsd:string"/>
</xsd: complexType>

<xsd: complexType name = "authorName">
<xsd: sequence>
<xsd: element name ="first-name" type ="xsd:string"/>
<xsd: element name = "last–name" type= "xsd:string"/>
</xsd: sequence>
</xsd: complexType>
</xsd:schema>
```

In this listing, <ComplexType>, <sequence> and <element> are schema elements. An element is a simple item with a single element. The ComplexType element is a set of

attributes that denotes that the element has children. Some other schema items are <all>, <annotation>, <any>, <anyAttribute>, <attribute>, <choice>, <documentation>, <field>, <group>, <include>, <key>, <length>, <maxLength>, <minLegth>, <selection>, <pattern>, <simpleType>, <unique>, and so on.

Elements and attributes are basic building block of a schema. An *element* is a tag with data. An element can have nested elements and attributes. Elements with one or more elements or attributes are ComplexType elements. An element contains a name and a data type. For example, the element price is of type decimal in the following line:

<xsd:element name ="price" type = "xsd:decimal"/>

This definition of the element price makes sure that it can only store a decimal type of a value. Other types of values are invalid values. For example, this is valid:

<price>19.95</price>

But this example is invalid:

<price>something</price>

Schema attributes are similar to XML attributes, but you can also define them using an xsd:attribute item. For example:

<xsd: ComplexType name= "bookstoreType">

Or:

<xsd: attribute name = " bookstoreType" type ="xsd:string"/>

A full discussion of these items is beyond the scope of this article; however, I'll describe any items I use in any of the samples.

Extensible Hypertext markup language (XHTML)

Extensible Hypertext Markup Language (XHTML) is a next-generation language of HTML. In January 2000, XHTML 1.0 became a W3C recommendation. XHTML is a better and improved version of HTML; however, it does have some restrictions.

XHTML is a combination of XML and HTML. XHTML uses elements of HTML 4.01 and rules of XML to provide a more consistent, well-formed and organized language.

An XML Document and its Items

An XML document is a set of elements in a well-formed and valid standard format. A document is valid if it has a DTD associated with it and if it complies with the DTD. As mentioned earlier, a document is well formed if it contains one or more elements and if it follows the exact syntaxes of the language. An XML parser will only parse a document that is a well formed, but the document doesn't necessarily have to be valid. This means that a document must have at least one element (a root element) in it, but it doesn't matter whether it uses DTDs.

An XML document has the following parts, each described in the sections that follow:

- Prolog
- DOCTYPE declaration
- Start and end tags
- Comments
- Character and entity references
- Empty elements
- Processing instructions
- CDATA section
- Attributes
- White spaces

Prolog

The prolog part of a document appears before the root tag. The prolog information applies to the entire document. It can have character encoding, stylesheets, comments, and processing instructions. This is an example of a prolog:

```
<?xml version ="1.0"  ?>
```

```
<?xml-stylesheet type="text/xsl"  href ="books.xsl" ?>
<!DOCTYPE StudentRecord SYSTEM "mydtd.dtd">
<!=my comments - - - ->
```

DOCTYPE Declaration

With the help of a DOCTYPE declaration, you can read the structure of your root element and DTD from external files. A DOCTYPE declaration can contain a root element or a DTD (used for document validation). In a validating environment, a DOCTYPE declaration is required. In a DOCTYPE reference, you can even use a URI reference. For example:

```
<!DOCTYPE rootElement>
```

or

```
<!DOCTYPE rootElement SYSTEM "URIreference">
```

or

```
<!DOCTYPE StudentRecord SYSTEM "mydtd.dtd">
```

Start and End tags

Start and end tags are the heart of XML language. As mentioned earlier in the article, XML is nothing but a text file with start and end tags. Each tag starts with <TAG> and ends with </TAG>. If you want to add a tag called <book> to your XML file, it must start with <book> and end the </book>, as shown in this example:

```
<?xml version ="1.0"  ?>
<book xmlns = "http://www.c-sharpcorner.com/xmlNet">
<title> The Autobiography of Benjamin Franklin</title>
<author>
<first-name>Benjamin</ First-name>
<last-name>Franklin</ last- name>
</author>
<price>8.99</ price>
</book>
```

NOTE: Empty elements don't have to heed this < >....</ > criteria. I'll discuss empty tags later in the "Empty Elements" section.

NOTE: An element is another name for a starting and ending tag pair.

Comments

Using comments in your code is a good programming practice. They help you understand your code, as well as help others to understand your code, by explaining certain code lines. You use the <! - - and - - > pair to write comments in an XML document:

<!-- My comments here -->
<!-- This is a comment -->

XML parsers ignore comments.

CDATA Sections

What if you want to use < and > characters in your XML file but not as part of a tag? Well, you can't use them because the XML parser will interpret them as start and end tags. CDATA provides the following solution. So you can use XML markup characters in your documents and have the XML parser ignore them. If you use the following line:

<! [CDATA [I want to use < and >, characters]]>

Then the parser will treat those characters as data.

Another good example of CDATA is the following example:

<! [CDATA [< Title>This is the title of a page</ Title>

In this case, the parser will treat the second title as data, not as a markup tag.

Character and entity reference

In some cases, you can't use a character directly in a document because of some limitations, such as a character being treated as a markup character or a device or processor limitation.

By using character and entity references, you can include information in a document by reference rather than the character.

A character reference is a hexadecimal code for a character. You use the hash symbol (#) before the hexadecimal value. The XML parser takes care of the rest. For example, the character reference for the Return Key is #x000d.

The reference starts with an ampersand (&) and a #, and it ends with a semicolon (;). The syntax for decimal and hexadecimal references is & # value; and &#xvalue; respectively. XML has some built-in entities. Use the It, gt, and amp entities for less than, greater than, and ampersand, respectively. Table 6-2 shows five XML built-in entities and their references. For example, if you want to write a > b or Jack & Jill, you can do that using entities as in the following:

A>b and Jack& Jill

Table 6-2. XML Build- in Entities

ENTITY	REFERENCE	DESCRIPTION
Lt	&lt	Less than: <
Gt	&gt	Greater than: >
Amp	&amp	Ampersand: &
Apos	&apos	Single quote: ‘
Auot	&quot	Double quote: “

Empty elements

Empty elements start and end with the same tag. They start with < and end with >. The text between these two symbols is the text data. For example:

<Name> </Name>
<IMG SRC= "img.jpg" />
<tagname/>

Are all examples of empty elements. The <IMG> specifies an inline image, and the SRC attribute specifies the image’s location. The image can be any format, though browsers generally support only GIF, JPEG, and PNG images.

Processing Instructions

Processing instructions (PIs) play a vital role in XML parsing. A PI holds the parsing instructions, which are read by the parser and other programs. If you noticed, the first line of any of the XML samples discussed earlier, a PI starts like this:

```
<?xml version ="1.0" ?>
```

All PIs start with <? and end with ?>. This is another example of a PI:

```
<?xml-stylesheet type ="text/ xsl" href = "myxsl.xsl"?>
```

This PI tells a parser to apply a stylesheet on the document.

Attributes

Attributes let you add extra information to an element without creating another element. An attribute is a name and value pair. Both the name and value must be present in an attribute. The attribute value must be in double quotes; otherwise the parser will give an error. Listing 6-8 is an example of an attribute in a <table> tag. In the example, the <table> tag has border and width attributes, and the <td> tag a width attribute.

Listing 6-8. Attributes in the < table> tag

```
<table border ="1" width = "43%">
<tr>
<td width ="50%">Row1, Column1</td>
<td width ="50%">Row1, Column2</td>
</tr>
<tr>
<td width = "50%">Row2, Column1</td>
<td width = "50%">Row2, Column2</td>
</tr>
</table>
```

White spaces

XML preserves white spaces except in attribute values. That means white space in your document will be displayed in the browser. However, white spaces are not allowed before the XML declaration. The XML parser reports all white spaces available in the document. If white spaces appear before a declaration, the parser treats them as PI.

In elements, the XML 1.0 standard defines the xml: space attribute to insert spaces in a document. The XML:space attribute accepts only two values: default and preserve. The default value is the same as not specifying an xml:space attribute. It allows the parser to treat spaces as in a normal document. The Preserve value tells the parser to preserve

space in the document. The parser preserves space in attributes, but it converts line breaks into single spaces.

Summary

XML has become one of the most popular mechanisms of data transfer among applications. In this book, we learned basics of XML and its related elements.