PART 1

Getting Started
Why XML?

XML, which stands for Extensible Markup Language, was defined by the XML Working Group of the World Wide Web Consortium (W3C). This group described the language as follows:

The Extensible Markup Language (XML) is a subset of SGML...Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

This is a quotation from version 1.0 of the official XML specification. You can read the entire document at http://www.w3.org/TR/REC-xml on the W3C Web site.

note

As this book goes to press, the current version of the XML specification is still 1.0. The first edition of this specification was published in February 1998. The second edition, which merely incorporates error corrections and clarifications and does not represent a new XML version, was published in October 2000. You’ll find the text of the second edition at the previous URL (http://www.w3.org/TR/REC-xml). The XML specification has the W3C status of Recommendation. Although this status might sound a bit tentative, it actually refers to the final, approved specification. (The role of the W3C is to recommend standards, not to enforce them.)

As you can see, XML is a markup language designed specifically for delivering information over the World Wide Web, just like HTML (Hypertext Markup Language), which has been the standard language used to create Web pages since the inception of the Web. Since we already have HTML, which continues to evolve to meet additional needs, you might wonder why we require a completely new language for the Web. What is new and different about XML? What
are its unique advantages and strengths? What is its relationship to HTML? Is it intended to replace HTML or to enhance it? And finally, what is this SGML that XML is a subset of, and why can’t we just use SGML for Web pages? In this chapter, I’ll attempt to answer all of these questions.

The Need for XML

HTML provides a fixed set of predefined elements that you can use to mark the components of a typical, general-purpose Web page. Examples of elements are headings, paragraphs, lists, tables, images, and hyperlinks. For instance, HTML works fine for creating a personal home page, as in the following example HTML page:

```
<HTML>
<HEAD>
<TITLE>Home Page</TITLE>
</HEAD>

<BODY>
<H1><IMG SRC="MainLogo.gif"> Michael J. Young's Home Page</H1>
<P><EM>Welcome to my Web site!</EM></P>

<H2>Web Site Contents</H2>
<P>Please choose one of the following topics:</P>
<UL>
  <LI><A HREF="writing.htm">Writing</A></LI>
  <LI><A HREF="family.htm">Family</A></LI>
  <LI><A HREF="photos.htm">Photo Gallery</A></LI>
</UL>

<H2>Other Interesting Web Sites</H2>
<P>Click one of the following to explore another Web site:</P>
<UL>
  <LI><A HREF="http://www.yahoo.com/">Yahoo Search Engine</A></LI>
  <LI><A HREF="http://www.amazon.com/">Amazon Bookstore</A></LI>
  <LI><A HREF="http://www.microsoft.com/mspress//">Microsoft Press</A>
```


Chapter 1 Why XML?

Microsoft Internet Explorer displays this page as shown in the following figure:

Each element begins with a start-tag: a block of text preceded with a left angle bracket (<) and followed with a right angle bracket (>) that contains the element name and possibly other information. Most elements end with an end-tag, which is like its corresponding start-tag except that it includes only a slash (/) character followed by the element name. The element’s content is the text—if any—between the start-tag and end-tag. Notice that many of the elements in the preceding example page contain nested elements (that is, elements within other elements).
The example HTML page contains the following elements:

<table>
<thead>
<tr>
<th>HTML element</th>
<th>Page component marked</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>The entire page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Heading information, such as the page title</td>
</tr>
<tr>
<td>TITLE</td>
<td>The page title, which appears in the browser’s title bar</td>
</tr>
<tr>
<td>BODY</td>
<td>The main body of text that the browser displays</td>
</tr>
<tr>
<td>H1</td>
<td>A top-level heading</td>
</tr>
<tr>
<td>H2</td>
<td>A second-level heading</td>
</tr>
<tr>
<td>P</td>
<td>A paragraph of text</td>
</tr>
<tr>
<td>UL</td>
<td>A bulleted list (Unordered List)</td>
</tr>
<tr>
<td>LI</td>
<td>An individual item within a list (List Item)</td>
</tr>
<tr>
<td>IMG</td>
<td>An image</td>
</tr>
<tr>
<td>A</td>
<td>A hyperlink to another location or page (an Anchor element)</td>
</tr>
<tr>
<td>EM</td>
<td>A block of italicized (EMphasized) text</td>
</tr>
<tr>
<td>B</td>
<td>A block of bold text</td>
</tr>
</tbody>
</table>

The browser that displays the HTML page recognizes each of these standard elements and knows how to format and display them. For example, the browser typically displays an H1 heading in a large font, an H2 heading in a smaller font, and a P element in an even smaller font. It displays an LI element within an unordered list as a bulleted, indented paragraph. And it converts an A element into an underlined hyperlink that the user can click to go to a different location or page.

Although the set of predefined HTML elements has expanded considerably since the first HTML version, HTML is still unsuitable for defining many types of documents. The following are examples of documents that can’t adequately be described using HTML:

- **A document that doesn’t consist of typical components (headings, paragraphs, lists, tables, and so on).** For instance, HTML lacks the elements necessary to mark a musical score or a set of mathematical equations.

- **A database, such as an inventory of books.** You could use an HTML page to store and display static database information (such as a list of book descriptions). However, if you wanted to sort, filter, find, and work with the information in other ways, each individual piece of information would need to be labeled (as it is in a database program such as Microsoft Access). HTML lacks the elements necessary to do this.
A document that you want to organize in a treelike hierarchical structure. Say, for example, that you’re writing a book and you want to mark it up into parts, chapters, A sections, B sections, C sections, and so on. A program could then use this structured document to generate a table of contents, to produce outlines with various levels of detail, to extract specific sections, and to work with the information in other ways. An HTML heading element, however, marks only the text of the heading itself to indicate how the text should be formatted. For example:

```html
<H2>Web Site Contents</H2>
```

Because you don’t nest the actual text and elements that belong to a document section within a heading element, these elements can’t be used to clearly indicate the hierarchical structure of a document.

The solution to these limitations is XML.

**The XML Solution**

The XML definition consists of only a bare-bones syntax. When you create an XML document, rather than use a limited set of predefined elements, you create your own elements and you assign them any names you like—hence the term *extensible* in Extensible Markup Language. You can therefore use XML to describe virtually any type of document, from a musical score to a database. For example, you could describe a list of books, as in the following XML document:

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

<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Moby-Dick</TITLE>
    <AUTHOR>Herman Melville</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>605</PAGES>
    <PRICE>$4.95</PRICE>
  </BOOK>
</INVENTORY>
```
note
When used to describe a database, XML has two advantages over proprietary formats (such as the Access .mdb or dBase .dbf format): XML is humanly readable, and it is based on a public, open standard.

It’s important to understand that the element names in an XML document (such as INVENTORY, BOOK, and TITLE, in this example) are not part of the XML definition. Rather, you make up the names when you create a particular document. And you can choose any legal names for your elements (such as LIST rather than INVENTORY, or ITEM rather than BOOK).

tip
When you name elements in an XML document, try to choose descriptive names—for example, BOOK or ITEM rather than FOO or BAR. One of the advantages of an XML document is that it can be self-describing—that is, each piece of information can have a descriptive label attached.

As you can see from the previous example, an XML document is structured in a treelike hierarchy, with elements completely nested within other elements and with a single top-level element (INVENTORY in this example)—known as the document element or root element—that contains all other elements. The structure of the example XML document can be drawn like this:
Chapter 1 Why XML?

You can thus readily use XML to define a hierarchically structured document, such as a book with parts, chapters, and various levels of sections, as mentioned previously.

Writing XML Documents

Because XML doesn’t include predefined elements, it might seem to be a relatively casual standard. XML does, however, have a strictly defined syntax. For example, unlike HTML, every XML element must have both a start-tag and an end-tag (or a special empty-element tag, which I’ll describe in later chapters). And any nested element must be completely contained within the element that encloses it.

In fact, the very flexibility of creating your own elements demands a strict syntax. That’s because the custom nature of XML documents demands custom software (for example, Web page scripts or freestanding programs) to handle and display the information these documents contain. The strict XML syntax gives XML documents a predictable form and makes this software easier to write. Recall from the quotation at the beginning of the chapter that “ease of implementation” is one of the chief goals of the language.

Part 2 of this book discusses creating XML documents that conform to the rules of syntax. As you’ll learn, you can write an XML document to conform to either of two different levels of syntactical strictness. A document is known as either well-formed or valid depending on which level of the standard it meets.
Displaying XML Documents

In an HTML page, a browser knows that an H1 element, for example, is a top-level heading and will format and display it accordingly. This is possible because this element is part of the HTML standard. But how can a browser or other program know how to handle and display the elements in an XML document you create (such as BOOK or BINDING in the example document), since you invent those elements yourself?

There are three basic ways to tell a browser (specifically, Microsoft Internet Explorer) how to handle and display each of your XML elements. I’ll cover these techniques in detail in Part 3 of the book.

- **Style sheet linking.** With this technique, you link a style sheet to the XML document. A style sheet is a separate file that contains instructions for formatting the individual XML elements. You can use either a cascading style sheet (CSS)—which is also used for HTML pages—or an Extensible Stylesheet Language Transformations (XSLT) style sheet—which is considerably more powerful than a CSS and is designed specifically for XML documents. I’ll cover these techniques in Chapters 2, 8, 9, and 12.

- **Data binding.** This option requires you to create an HTML page, link the XML document to it, and bind standard HTML elements in the page, such as SPAN or TABLE elements, to the XML elements. The HTML elements then automatically display the information from the XML elements they are bound to. You’ll learn this technique in Chapter 10.

- **XML DOM Scripting.** With this technique, you create an HTML page, link the XML document to it, and access and display individual XML elements by writing script code (JavaScript or Microsoft Visual Basic Scripting Edition [VBScript]). The browser exposes the XML document as an XML Document Object Model (DOM), which provides a large set of objects, properties, and methods that the script code can use to access, manipulate, and display the XML elements. I’ll discuss this technique in Chapter 11.
Chapter 1  Why XML?

SGML, HTML, and XML

SGML, which stands for Structured Generalized Markup Language, is the mother of all markup languages. Both HTML and XML are derived from SGML, although in fundamentally different ways. SGML defines a basic syntax, but allows you to create your own elements (hence the term generalized). To use SGML to describe a particular document, you must invent an appropriate set of elements and a document structure. For example, to describe a book, you might use elements that you name BOOK, PART, CHAPTER, INTRODUCTION, A-SECTION, B-SECTION, C-SECTION, and so on.

A general-purpose set of elements used to describe a particular type of document is known as an SGML application. (An SGML application also includes rules that specify the ways the elements can be arranged—as well as other features—using techniques similar to those I’ll discuss in Chapter 5.) You can define your own SGML application to describe a specific type of document that you work with, or a standards body can define an SGML application to describe a widely used document type. The most famous example of this latter type of application is HTML, which is an SGML application developed in 1991 to describe Web pages.

SGML might seem to be the perfect extensible language for describing information that’s delivered and processed on the Web. However, the W3C members who contemplate these matters deemed SGML too complex to be a universal language for the Web. The flexibility and superfluity of features provided by SGML would make it difficult to write the software needed to process and display the SGML information in Web browsers. What was needed was a streamlined subset of SGML designed specifically for delivering information on the Web. In 1996, the XML Working Group of the W3C began to develop that subset, which they named Extensible Markup Language. As the quotation at the beginning of the chapter states, XML was designed for “ease of implementation,” a feature clearly lacking in SGML.

XML is thus a simplified version of SGML optimized for the Web. As with SGML, XML lets you devise your own set of elements when you describe a particular document. Also like SGML, an individual or a standards body can define an XML application, which is a general-purpose set of elements and attributes and a document structure that can be used to describe documents of a particular type (for example, documents containing mathematical formulas or vector graphics). You’ll learn more about XML applications later in this chapter.

The XML syntax offers fewer features and alternatives than SGML, making it easier for humans to read and write XML documents and for programmers to write browsers, Web page scripts, and other programs that access and display the document information.
Does XML Replace HTML?

Currently, the answer to that question is no. HTML is still the primary language used to tell browsers how to display information on the Web.

With Internet Explorer, the only practical way to dispense entirely with HTML when you display XML is to attach a cascading style sheet to the XML document and then open the document directly in the browser. However, using a cascading style sheet is a relatively restrictive method for displaying and working with XML. All the other methods you’ll learn in this book involve HTML. Data binding and XML DOM scripts both use HTML Web pages as vehicles for displaying XML documents. And with XSLT style sheets, you create templates that transform the XML document into HTML that tells the browser how to format and display the XML data.

Rather than replacing HTML, XML is currently used in conjunction with HTML and vastly extends the capability of Web pages to:

- Deliver virtually any type of document
- Sort, filter, rearrange, find, and manipulate the information in other ways
- Present highly structured information

As the quotation at the beginning of the chapter states, XML was designed for interoperability with HTML.

The Official Goals of XML

The following are the 10 design goals for XML as stated in the official XML specification posted on the W3C Web site (http://www.w3.org/TR/REC-xml).

1. XML shall be straightforwardly usable over the Internet.

XML was designed primarily for storing and delivering information on the Web, as explained earlier in this chapter, and for supporting distributed applications on the Internet.

2. XML shall support a wide variety of applications.

Although its primary use is for exchanging information over the Internet, XML was also designed for use by programs that aren’t on the Internet, such as software tools for creating documents and for filtering, translating, or formatting information.
“3 XML shall be compatible with SGML.”

XML was designed to be a subset of SGML, so that every valid XML document would also be a conformant SGML document, and to have essentially the same expressive capability as SGML. A benefit of achieving this goal is that programmers can easily adapt SGML software tools for working with XML documents.

“4 It shall be easy to write programs which process XML documents.”

If a markup language for the Web is to be practical and gain universal acceptance, it must be easy to write the browsers and other programs that process the documents. In fact, the primary reason for defining the XML subset of SGML was the unwieldiness of writing programs to process SGML documents.

“5 The number of optional features in XML is to be kept to the absolute minimum, ideally zero.”

Having a minimal number of optional features in XML facilitates writing processors that can handle virtually any XML document, making XML documents universally interchangeable. The abundance of optional features in SGML was a primary reason why it was deemed impractical for defining Web documents. Optional SGML features include redefining the delimiting characters in tags (normally the < and > characters) and the omission of the end-tag when the processor can figure out where an element ends. A universal processor for SGML documents would be difficult to write because it would have to account for all optional features, even those that are seldom used.

“6 XML documents should be human-legible and reasonably clear.”

XML was designed to be a lingua franca for exchanging information among users and programs the world over. Human readability supports this goal by allowing people—as well as specialized software programs—to read XML documents and to write them using simple text editors. A benefit of human legibility is that users can easily work around limitations and bugs in their software tools by simply opening an XML document in a text editor and taking a look at it. Its human legibility distinguishes XML from most proprietary formats used for databases and word-processing documents.

Humans can easily read an XML document because it’s written in plain text and has a logical treelike structure. You can enhance XML’s legibility by choosing meaningful names for your document’s elements, attributes, and entities; by carefully arranging and indenting the text to clearly show the logical structure of the document at a glance; and by adding useful comments. (I’ll explain elements, attributes, entities, and comments in later chapters.)
“7 The XML design should be prepared quickly.”

A standard such as XML can, of course, be viable only if the community of programmers and users adopts it. The XML standard therefore needed to be completed before this community began to adopt alternative standards, which software companies tend to produce at a rapid pace.

“8 The design of XML shall be formal and concise.”

The XML specification includes a formal XML grammar, which uses a notation known as Extended Backus-Naur Form (EBNF). This notation, although difficult to read casually, resolves ambiguities and ultimately makes it easier to write XML documents and especially XML software tools, further encouraging XML’s adoption.

“9 XML documents shall be easy to create.”

For XML to be a practical markup language for Web documents, not only must the software for handling XML be easy to write, but also XML documents themselves must be easy to create.

“10 Terseness in XML markup is of minimal importance.”

In keeping with goal 6, intelligibility of XML is more important than brevity. Part of the problem in using SGML as a universal Web markup language is the excessive terseness it fosters by allowing you to omit markup in certain situations.

As you learn XML, you’ll be able to judge how fully these goals have been realized. After reading Chapter 6 on using entities, you might possibly question the success of the standard in achieving clarity and ease of use, but I hope Part 2 of this book will help you over that possible hurdle.

**Standard XML Applications**

As you’ve seen, not only can you use XML to describe an individual document, but also a person, company, or standards committee can define a general-purpose set of XML elements and attributes, together with a document structure, to be used for describing a particular class of documents. The definition of the elements, attributes, and structure is known as an **XML application** or **XML vocabulary**.

For example, an organization could define an XML application for creating documents that depict molecular structures, documents that describe human resources, documents that choreograph multimedia presentations, or documents that store vector graphics.
An XML application is usually defined by creating a document type definition (DTD), which is an optional component of an XML document, or an XML schema, which is contained in a separate file. A DTD or an XML schema defines and names the elements that can be used in the document, the order in which the elements can appear, the element attributes that can be used, and other document features. To use a particular XML application, you usually include its DTD in your XML document or employ its XML schema in processing your document. Using a DTD or XML schema restricts the elements, attributes, and structure that you can use so that your document is forced to conform to the XML application standard. (The example XML document you saw earlier in the chapter doesn’t include a DTD.) You’ll learn how to define DTDs in Chapter 5 and how to create XML schemas in Chapter 7.

An important advantage of using a standard XML application to develop documents is that an application promotes consistency, both within a single document and among separate documents of the same type. Consistency in the document elements, attributes, structure, and other features is critical in a group of documents that are all going to be processed and displayed using a particular software tool (for example, a Web page script) that is designed for that type of document.

In the next two sections, I discuss specific XML applications that have been proposed or that have already been created.

**Real-World Uses for XML**

Although XML might be an interesting concept, you may be wondering what you can actually do with it in the real world. In this section, I’ve listed a sampling of practical uses for XML. I’ve included ways that XML is currently used, as well as uses that various organizations have proposed. For most of these uses, I’ve listed in parentheses one or more standard XML applications that have been defined. For example, I’ve listed MathML (Mathematical Markup Language) as a specific XML application for formatting mathematical formulas and scientific content on the Web.
Storing databases. Like proprietary database formats, XML can be used to label each field of information within each database record. (For example, it could label each name, address, and phone number within the records of an address database.) Labeling each piece of information lets you display the data in a variety of ways and search, sort, filter, and process the data in other ways.

note
Several of the Microsoft Office XP programs now use XML as one of the standard document formats. For example, Microsoft Excel 2002 lets you open or save workbooks in XML format, as an alternative to the standard .xls Excel workbook format. And Microsoft Access 2002 allows you to create a database table by importing an XML document, or to export a database table or other object to an XML document.

Structuring documents. The treelike structure of XML documents makes XML ideal for marking the structure of documents such as novels, nonfiction books, and plays. For example, you could use XML to mark a play into acts, scenes, speakers, lines, stage directions, and so on. The XML marking allows software tools (such as style sheets or Web page scripts) to display or print the document with proper formatting; to find, extract, or manipulate document information; to generate tables of contents, outlines, and synopses; and to handle the information in other ways. (For instance, Jon Bosak, who chaired the XML Working Group, has created XML versions of the complete works of William Shakespeare, as well as the Old Testament, New Testament, Koran, and Book of Mormon. Go to http://www.ibiblio.org/bosak/ to download these documents.)
Chapter 1  Why XML?

- Storing vector graphics.  (VML, or Vector Markup Language)
- Describing multimedia presentations.  (SMIL, or Synchronized Multimedia Integration Language)
- Creating voice interfaces for Internet programs.  (VoxML, or Voice Markup Language)
- Defining channels.  Channels are Web pages that are pushed (sent automatically) to subscribers.  (CDF, or Channel Definition Format)
- Describing software packages and their interdependencies.  These descriptions allow software to be distributed and updated over networks.  (OSD, or Open Software Description Format)
- Communicating among programs over the Web in an open and extensible way, using XML-based messages.  These messages are independent of the operating systems, object models, and computer languages used.  (SOAP, or Simple Object Access Protocol)
- Exchanging financial information.  The information is exchanged in an open, humanly readable format, among financial programs (such as Quicken and Microsoft Money) and financial institutions (such as banks and mutual funds).  (OFX, or Open Financial Exchange)
- Creating, managing, and using complex digital forms for Internet commerce transactions.  These forms can include digital signatures that make the forms legally binding.  (XFDL, or Extensible Forms Description Language)
- Exchanging job descriptions and résumés.  (HRMML, or Human Resource Management Markup Language)
- Exchanging information between printing companies and their customers.  (PrintML, or Printing Industry Markup Language, and XPP, or XML for Publishers and Printers)
- Filing legal documents and exchanging legal information electronically.  (OXCI, or Open XML Court Interface)
- Exchanging insurance-related data.  (iLingo)
- Exchanging real estate transaction information.  (RETML, or Real Estate Transaction Markup Language)
- Exchanging news articles on the Internet. (NML, or News Markup Language)
- Storing tracking information by courier services. Federal Express, for example, currently uses XML for this purpose.
- Formatting mathematical formulas and scientific content on the Web. (MathML, or Mathematical Markup Language)
- Describing molecular structures. (CML, or Chemical Markup Language)
- Encoding and displaying DNA, RNA, and protein sequence information. (BSML, or Bioinformatic Sequence Markup Language)
- Exchanging astronomical data. (AML, or Astronomical Markup Language)
- Encoding weather observation reports. (OMF, or Weather Observation Markup Format)
- Storing and exchanging geographic information. (GML, or Geography Markup Language)
- Encoding genealogical data. (GedML, or Genealogical Data in XML)
- Formatting theses and dissertations for electronic submission. (ETD-ML, or Electronic Thesis and Dissertation Markup Language)
- Storing theological information and marking up liturgical texts. (ThML, or Theological Markup Language, and LitML, or Liturgical Markup Language)
- Representing musical scores. (MusicXML)
- Sending electronic business cards via e-mail. (XML version of vCard)
- Storing and exchanging information on chess. (ChessML, or Chess Markup Language)
- Recording recipes on computers. (DESSERT, or Document Encoding and Structuring Specification for Electronic Recipe Transfer)
Creating HTML Web pages that are valid XML documents. (XHTML. If you write HTML Web pages that conform to the XHTML application of XML, you can validate, display, and work with the pages using standard XML tools, such as XML DOM scripts, which are discussed in Chapter 11. See http://www.w3.org/TR/xhtml1.)

XML Applications for Enhancing XML Documents

In addition to XML applications for describing specific classes of documents, several XML applications have been defined that you can use in conjunction with any type of XML document to facilitate the document’s creation or to enhance it in some way. Several examples are:

- **Extensible Stylesheet Language Transformations (XSLT)** allows you to create powerful document style sheets using XML syntax. XSLT is used in conjunction with XML Path Language (XPath), which lets you select specific parts of an XML document. XPath uses a location path notation—similar to file paths or URLs—to address locations within the hierarchical XML document structure.

- **XML Schema** lets you write detailed schemas for your XML documents using standard XML syntax. It provides a more powerful alternative to writing DTDs.

- **XML Linking Language (XLink)** lets you link your XML documents. It allows multiple link targets and other advanced features, and is considerably more powerful than the HTML hyperlink mechanism. XLink is used in conjunction with XML Pointer Language (XPointer), which lets you define flexible link targets. You can use XPointer to link to any location in a target document, not just to a specially marked link target as in HTML.

I’ll discuss XML Schema in Chapter 7, and XSLT and XPath in Chapter 12. XLink and XPointer are still evolving and aren’t currently supported by Internet Explorer; these applications are beyond the scope of the book.

As you can see, XML is not only an immediately useful tool for defining documents, but is also serving as the framework for building the applications and XML enhancements that will be needed as the Internet evolves.
In this chapter, you’ll gain an overview of the entire process of creating and displaying an XML document in a Web browser. First you’ll create a simple XML document, explore the document’s structure, and learn some of the fundamental rules for creating a well-formed XML document. Then you’ll discover how to display that document in the Microsoft Internet Explorer Web browser by creating and attaching a simple style sheet that tells the browser how to format the elements in the document.

This chapter provides a brief preview of the topics that I’ll address in depth throughout the remainder of the book.

Creating an XML Document

Because an XML document is written in plain text, you can create one using your favorite text editor. For example, you can use the Notepad editor that comes with Microsoft Windows. Or, better yet, you can use a programming editor with features that make it easier to type in XML and related source files. Useful features include automatic tab insertion (the next line is indented automatically when you press the Enter key) and the ability to select and indent, or decrease the indent of, multiple lines of text. The Microsoft Visual Studio programming editor, the text editor that comes with Microsoft Visual Studio or Visual Studio .NET, is one example of an editor with these features.
The Visual Studio .NET editor offers many special features for working with XML. For instance, when you type a start-tag, it automatically inserts a matching end-tag; when you enter an attribute, it automatically inserts quote characters for the attribute value; and it color-codes different XML components. The editor also provides a Data view, which displays the XML data in a table format that allows you to view and quickly edit the contents of the document.

If you use a word-processing program—such as Microsoft Word or WordPad—to create an XML document, a style sheet, or other source file, you must save the file in a plain text format. If you save the file in a word-processor format (such as native Word .doc format or Rich Text Format) the file will contain extraneous characters that make it unsuitable for processing in a browser or other XML program. For information on using various text file encodings, see the sidebar “Characters, Encoding, and Languages” on page 77.

Create the XML Document

1. Open a new, empty text file in your text editor, and type in the XML document shown in Listing 2-1. (You’ll find a copy of this listing on this book’s companion CD under the filename Inventory.xml.) If you want, you can omit some of the BOOK elements. You don’t need to type in all eight of them—three or four will do. (A BOOK element consists of the <BOOK> and </BOOK> tags plus all text between them.)

2. Use your text editor’s Save command to save the document on your hard disk, assigning it the filename Inventory.xml.
Chapter 2 Creating and Displaying Your First XML Document

**tip**
Notepad normally assigns the .txt extension to a file you save. To assign a different extension (such as .xml for an XML document or .css for a cascading style sheet), you might need to put quotation marks around the entire filename and extension. For example, to save a file as Inventory.xml, you might need to type “Inventory.xml” (including the quotation marks) in the File Name text box of Notepad’s Save As dialog box. If you omit the quotation marks, Notepad will save the file as Inventory.xml.txt if the .xml extension isn’t registered on your computer. In general, if you type an extension that isn’t registered, Notepad will append the .txt extension.

To open a file in Notepad that has an extension other than .txt, you need to run the Notepad program and use the Open command on the File menu. Or, once Notepad is running, you can drag a file from Windows Explorer and drop it on the Notepad window. Because the file doesn’t have the .txt extension, you can’t open it by double-clicking it as you can with a .txt file.

Inventory.xml

```xml
<?xml version="1.0"?>
<!-- File Name: Inventory.xml -->

<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
  </BOOK>
</INVENTORY>
```
Listing 2-1.

<BOOK>
  <TITLE>The Marble Faun</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>473</PAGES>
  <PRICE>$10.95</PRICE>
</BOOK>

<BOOK>
  <TITLE>Moby-Dick</TITLE>
  <AUTHOR>Herman Melville</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>724</PAGES>
  <PRICE>$9.95</PRICE>
</BOOK>

<BOOK>
  <TITLE>The Portrait of a Lady</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>256</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>

<BOOK>
  <TITLE>The Scarlet Letter</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>253</PAGES>
  <PRICE>$4.25</PRICE>
</BOOK>

<BOOK>
  <TITLE>The Turn of the Screw</TITLE>
  <AUTHOR>Henry James</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>384</PAGES>
  <PRICE>$3.35</PRICE>
</BOOK>

</INVENTORY>
Chapter 2  Creating and Displaying Your First XML Document

The Anatomy of an XML Document

An XML document, such as the example document you just typed, consists of two main parts: the prolog and the document element. (The document element is also known as the root element.)

The Prolog

The prolog of the example document consists of three lines:

```xml
<?xml version="1.0"?>
<!-- File Name: Inventory.xml -->
</INVENTORY>
```

The first line is the XML declaration, which states that this is an XML document and gives the XML version number. (At the time of this writing, the latest XML version was 1.0.) The XML declaration is optional, although the specification states that it should be included. If you do include an XML declaration, it must appear at the very beginning of the document.

The second line of the prolog consists of white space. To enhance readability, you can insert any amount of white space (spaces, tabs, or line breaks) between the components of the prolog. The XML processor ignores it.

The third line of the prolog is a comment. Adding comments to an XML document is optional, but doing so can increase the document’s readability. A comment begins with the <!-- characters and it ends with the --> characters. You can type any text you want (except --) between these two groups of characters. The XML processor ignores it.

The Elements nested within document element

The Document element (Root element)

The Prolog

The prolog of the example document consists of three lines:

```xml
<?xml version="1.0"?>
<!-- File Name: Inventory.xml -->
</INVENTORY>
```

The first line is the XML declaration, which states that this is an XML document and gives the XML version number. (At the time of this writing, the latest XML version was 1.0.) The XML declaration is optional, although the specification states that it should be included. If you do include an XML declaration, it must appear at the very beginning of the document.

The second line of the prolog consists of white space. To enhance readability, you can insert any amount of white space (spaces, tabs, or line breaks) between the components of the prolog. The XML processor ignores it.

The third line of the prolog is a comment. Adding comments to an XML document is optional, but doing so can increase the document’s readability. A comment begins with the <!-- characters and it ends with the --> characters. You can type any text you want (except --) between these two groups of characters. The XML processor ignores it.

As explained in Chapter 11, the Internet Explorer XML processor
makes comment text available to Web page scripts, and as explained in Chapter 12, it also makes comments available to XSLT style sheets.)

**note**
The XML processor is the software module that reads the XML document and provides access to the document’s contents and structure. It provides this access to another software module called the application, which manipulates and displays the document’s contents. When you display an XML document in Internet Explorer, the browser provides both the XML processor and at least part of the application. (If you write HTML or script code to display an XML document, you supply part of the application yourself.) The distinction is more than academic because the XML specification governs the behavior of the processor but not that of the application. An XML processor that conforms to the specification provides a predictable body of data to the application, which can do whatever it wants with this data. Note that the term application as used here is not the same thing as an XML application (or XML vocabulary), which I defined in Chapter 1 as a general-purpose set of elements and attributes, along with a document structure, that can be used to describe documents of a particular type.

The prolog can also contain the following optional components:

- **A document type declaration**, which defines the type, content, and structure of the document. If used, the document type declaration must come after the XML declaration. (The definition of the document’s content and structure is contained in a subcomponent of the document type declaration known as a document type definition or DTD.)

- One or more **processing instructions**, which provide information that the XML processor passes on to the application. Later in this chapter, you’ll see a processing instruction for linking a style sheet to the XML document.

**note**
All of the prolog components mentioned in this section are described in detail in later chapters.
The Document Element

The second main part of an XML document is a single element known as the document element or root element, which can contain additional nested elements.

In an XML document, the elements indicate the logical structure of the document and contain the document’s information content (which in the example document is the book information, such as the titles, author names, and prices). A typical element consists of a start-tag, the element’s content, and an end-tag. The element’s content can be character data, other (nested) elements, or a combination of both.

note

The text in an XML document consists of intermingled markup and character data. Markup is delimited text that describes the storage layout and logical structure of the document. The following are the different kinds of markup: element start-tags, element end-tags, empty-element tags, comments, document type declarations, processing instructions, XML declarations, text declarations, CDATA section delimiters, entity references, character references, and any white space that is at the top level of the document (that is, outside the document element and outside other markup). You’ll learn about each of these types of markup in later chapters. All other text is character data—the actual information content of the document (in the example document, the titles, author names, prices, and other book information).

In the example document, the document element is INVENTORY. Its start-tag is <INVENTORY>, its end-tag is </INVENTORY>, and its content is eight nested BOOK elements.

note

The document element in an XML document is similar to the BODY element in an HTML page, except that you can assign it any legal name.
Each BOOK element likewise contains a series of nested elements:

![XML Diagram]

**note**
The name that appears at the beginning of the start-tag and in the end-tag identifies the element’s type.

Each of the elements nested in a BOOK element, such as a TITLE element, contains only character data:

![XML Diagram]

In Part 2 of the book, you’ll learn all about adding elements to your XML documents and including attributes in an element’s start-tag.

**Some Basic XML Rules**
The following are a few of the basic rules for creating a well-formed XML document. A well-formed document is one that conforms to the minimal set of rules that allow the document to be processed by a browser or other XML program. The document you typed earlier in the chapter (Listing 2-1) is an example of a well-formed XML document that conforms to these rules.

- The document must have exactly one top-level element (the document element or root element). All other elements must be nested within it.
Elements must be properly nested. That is, if an element starts within another element, it must also end within that same element.

Each element must have both a start-tag and an end-tag. Unlike HTML, XML doesn’t let you omit the end-tag—not even in situations where the browser would be able to figure out where the element ends. (In Chapter 3, however, you’ll learn a shortcut notation you can use for an empty element—that is, an element with no content.)

The element-type name in a start-tag must exactly match the name in the corresponding end-tag.

Element-type names are case-sensitive. In fact, all text within XML markup is case-sensitive. For example, the following element is illegal because the type name in the start-tag doesn’t match the type name in the end-tag:

```xml
<TITLE>Leaves of Grass</Title> <!-- illegal element -->
```

tip
In Part 2 of the book, you’ll find detailed instructions for writing not only well-formed XML documents but also valid XML documents, which meet a more stringent set of requirements.

Displaying the XML Document

You can open an XML document directly within the Internet Explorer browser, just like you’d open an HTML Web page.

If the XML document doesn’t contain a link to a style sheet, Internet Explorer will simply display the text of the complete document, including both the markup (the tags and comments, for example) and the character data. Internet Explorer color-codes the different document components to help you identify them, and it displays the document element as a collapsible/expandable tree to clearly indicate the document’s logical structure and to allow you to view various levels of detail.

If, however, the XML document contains a link to a style sheet, Internet Explorer will display only the character data from the document’s elements, and it will format this data according to the rules you have specified in the style sheet.
You can use either a cascading style sheet (CSS)—the same type of style sheet used for HTML pages—or an Extensible Stylesheet Language Transformations (XSLT) style sheet—a more powerful type of style sheet that employs XML syntax and can be used only for XML documents. (An XSLT style sheet lets you display attribute values and other information contained in an XML document, in addition to character data from elements.)

Display the XML Document Without a Style Sheet

1. In Windows Explorer or in a folder window, double-click the name of the file, Inventory.xml, that you saved in the previous exercise. Internet Explorer will display the document as shown here:

   ![XML Document Without Style Sheet]

2. Experiment with changing the level of detail shown within the document element. Clicking the minus symbol (-) to the left of a start-tag collapses the element, while clicking the plus symbol (+) next to a collapsed element expands it. For instance, if you click the minus symbol next to the INVENTORY element, as shown here:

   ```xml
   <?xml version="1.0" ?>
   <!-- File Name: Inventory.xml -->
   <INVENTORY>
   <BOOK>
     <TITLE>The Adventures of Huckleberry Finn</TITLE>
     <AUTHOR>Mark Twain</AUTHOR>
     <BINDING>mass market paperback</BINDING>
     <PAGES>298</PAGES>
     <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
     <TITLE>Leaves of Grass</TITLE>
     <AUTHOR>Walt Whitman</AUTHOR>
     <BINDING>hardcover</BINDING>
     <PAGES>462</PAGES>
     <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
     <TITLE>The Legend of Sleepy Hollow</TITLE>
     <AUTHOR>Washington Irving</AUTHOR>
     <BINDING>mass market paperback</BINDING>
     <PAGES>96</PAGES>
     <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK>
     <TITLE>The Marble Faun</TITLE>
     <AUTHOR>Nathaniel Hawthorne</AUTHOR>
     <BINDING>trade paperback</BINDING>
   </BOOK>
   ```
the entire document element will be collapsed, as shown here:

![XML Document in Internet Explorer](image.png)

**Catch XML Errors in Internet Explorer**

Before Internet Explorer displays your XML document, its XML parser component analyzes the document contents. If the parser detects an error, Internet Explorer displays a page with an error message rather than attempting to display the document. Internet Explorer will display the error page whether or not the XML document is linked to a style sheet.

**note**

The XML parser is the part of the XML processor that scans the XML document, analyzes its structure, and detects any errors in syntax. See the Note on page 26 for a definition of XML processor.
In the following exercise, you’ll investigate the Internet Explorer error-checking feature by purposely introducing an error into the Inventory.xml document.

1. In your text editor, open the Inventory.xml document that you created in a previous exercise. Change the first TITLE element from
   `<TITLE>The Adventures of Huckleberry Finn</TITLE>`
   to
   `<TITLE>The Adventures of Huckleberry Finn</Title>`
   The element-type name in the end-tag now no longer matches the element-type name in the start-tag. Remember that element-type names are case-sensitive!

2. Save the changed document.

3. In Windows Explorer or in a folder window, double-click the document filename Inventory.xml.
   Rather than displaying the XML document, Internet Explorer will now display the following error-message page:
note
When you open an XML document directly in Internet Explorer, as you do in this chapter, the parser checks only whether the document is well-formed and then displays a message if it finds an error. It doesn’t check whether the document is valid.

Because you’ll work with Inventory.xml again in later chapters, you should now restore the end-tag in the first TITLE element to its original form (<TITLE>) and then resave the document.

note
If an XML document contains more than one well-formedness error, Internet Explorer displays only the first one it encounters. You’ll need to fix the errors one at a time. After you fix each error, you’ll need to save the document and reopen it in Internet Explorer to check for additional errors. (You can quickly reopen the document or page that’s currently displayed in Internet Explorer by clicking the Refresh toolbar button or by pressing F5.)

Even though you didn’t link a style sheet to the XML document, Internet Explorer uses a default style sheet to display the document; hence the error page refers to “using XSL style sheet.” (XSL style sheets are similar to the more recent XSLT style sheets, which are covered in Chapter 12.)

tip
As you work through the chapters in this book, keep in mind that you can quickly check whether an XML document is well-formed by simply opening it directly in Internet Explorer. (If you display an XML document through an HTML page, as described in Part 3, an XML document with an error will fail to display, but you won’t see an error message unless you explicitly write script code to show one.)
Display the XML Document Using a Cascading Style Sheet

1. Open a new, empty text file in your text editor, and type in the cascading style sheet (CSS) shown in Listing 2-2. (You'll find a copy of this listing on the companion CD under the filename Inventory01.css.)

2. Use your text editor’s Save command to save the style sheet on your hard disk, assigning it the filename Inventory01.css. The CSS you just created tells Internet Explorer to format the elements’ character data as follows:
   - Display each BOOK element with 12 points of space above it (margin-top:12pt) and a line break above and below it (display:block), using a 10-point font (font-size:10pt).
   - Display each TITLE element in italic (font-style:italic).
   - Display each AUTHOR element in bold (font-weight:bold).

`Inventory01.css`
/* File Name: Inventory01.css */

BOOK
{display:block;
 margin-top:12pt;
 font-size:10pt}

TITLE
{font-style:italic}

AUTHOR
{font-weight:bold}

Listing 2-2.

3. In your text editor, open the Inventory.xml document that you created in a previous exercise. Add the following processing instruction to the end of the document prolog, directly above the INVENTORY element:

```xml
<?xml-stylesheet type="text/css" href="Inventory01.css"?>
```

This processing instruction links the CSS you just created to the XML document. As a result, when you open the document in Internet Explorer, the browser displays the document content according to the instructions in the style sheet.
4. To reflect the new filename you’re going to assign, change the comment near the beginning of the document from

```xml
<!-- File Name: Inventory.xml -->
```
to

```xml
<!-- File Name: Inventory01.xml -->
```
Listing 2-3 shows the complete XML document. (You’ll find a copy of this listing on the companion CD under the filename Inventory01.xml.)

5. Use your text editor’s Save As command to save a copy of the modified document under the filename Inventory01.xml. Be sure to save it in the same file folder in which you saved Inventory01.css.

**Inventory01.xml**

```xml
<?xml version="1.0"?>
<!-- File Name: Inventory01.xml -->
<!--xml-stylesheet type="text/css" href="Inventory01.css"-->
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
  </BOOK>
</INVENTORY>
```
Listing 2-3.
6 In Windows Explorer or in a folder window, double-click the Inventory01.xml filename to open the document. Internet Explorer will open the Inventory01.xml document and display it according to the rules in the linked Inventory01.css style sheet, as shown here:

To get a feel for how you can change the XML document’s appearance by modifying the linked style sheet, open a new, empty text file in your text editor, and type in the modified CSS shown in Listing 2-4. (You’ll find a copy of this listing on the companion CD under the filename Inventory02.css.)

8 Use your text editor’s Save command to save the new style sheet on your hard disk, assigning it the filename Inventory02.css. The modified style sheet you just typed tells Internet Explorer to format the elements’ character data as follows:

- Display each BOOK element with 12 points of space above it (margin-top:12pt) and a line break above and below it (display:block), using a 10-point font (font-size:10pt).
- Display the TITLE, AUTHOR, BINDING, and PRICE elements each on a separate line (display:block).
- Display the TITLE element in a 12-point (font-size:12pt), bold (font-weight:bold), italic (font-style:italic) font. (Note that the 12-point font-size specification made for the TITLE element overrides the 10-point specification made for the element’s parent, BOOK.)
- Indent the AUTHOR, BINDING, and PRICE elements each by 15 points (margin-left:15pt).
In your text editor, open the Inventory.xml document. Add the following processing instruction to the end of the document prolog, directly above the INVENTORY element:

```xml
<?xml-stylesheet type="text/css" href="Inventory02.css"?>
```
Chapter 2  Creating and Displaying Your First XML Document

<?xml-stylesheet type="text/css" href="Inventory02.css"?>
This processing instruction links the new CSS you just created to the XML document.

10 To reflect the new filename you're going to assign, change the comment near the beginning of the document from

<!-- File Name: Inventory.xml -->
to

<!-- File Name: Inventory02.xml -->
Listing 2-5 shows the complete XML document. (You'll find a copy of this listing on the companion CD under the filename Inventory02.xml.)

11 Use your text editor's Save As command to save a copy of the modified document under the filename Inventory02.xml. Be sure to save it in the same file folder in which you saved Inventory02.css.

Inventory02.xml

<?xml version="1.0"?>

<!-- File Name: Inventory02.xml -->

<?xml-stylesheet type="text/css" href="Inventory02.css"?>

<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Legend of Sleepy Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <BINDING>mass market paperback</BINDING>
   </BOOK>
</INVENTORY>
Listing 2-5.
In Windows Explorer or in a folder window, double-click the Inventory02.xml filename to open it. Internet Explorer will open the Inventory02.xml document and display it according to the rules in the linked Inventory02.css style sheet, as shown here (only the first six books are shown; scrolling down would reveal the last two books):

![Image of the Inventory02.xml document in Internet Explorer]

**tip**

Part 3 of the book provides complete instructions for displaying XML documents on the Web. I’ll cover cascading style sheets, such as the one you created here, in Chapters 8 and 9. I’ll cover XSLT style sheets in Chapter 12. You’ll learn alternative methods for displaying XML documents on the Web in Chapters 10 and 11.