# 1
# Essential XML

WELCOME TO THE WORLD OF EXTENSIBLE MARKUP Language (XML). This book is your guided tour to that world, so have no worries—you've come to the right place. The world of XML is large and is expanding in unpredictable ways every minute, but we'll become familiar with the lay of the land in detail here. We also have a lot of territory to cover because XML is getting into the most amazing places, and in the most amazing ways, these days.

XML is a language defined by the World Wide Web Consortium (W3C, at `www.w3c.org`), the body that sets the standards for the Web. This first chapter is all about getting a solid overview of that language and how you can use it. For example, you probably already know that you can use XML to create your own elements by designing a customized markup language for your own use. In this way, XML supercedes other markup languages such as Hypertext Markup Language (HTML): In HTML, all the HTML elements you can use are predefined—and there are simply not enough of them. In fact, XML is a meta-markup language because it lets you create your own markup language.

## Markup Languages

Markup languages are all about describing the form of the document—that is, the way the content of the document should be interpreted. The markup language that most people are familiar with today, of course, is HTML,

which you use to create standard Web pages. Here's a sample HTML page:

```
<HTML>
    <HEAD>
        <TITLE>Hello From HTML</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>
               Hello From HTML
            </H1>
        </CENTER>
        Welcome to the wild and woolly world of HTML.
    </BODY>
</HTML>
```

You can see the results of this HTML in Figure 1.1 in Netscape Navigator. Note that the HTML markup in this page—that is, *tags* such as `<HEAD>`, `<CENTER>`, `<H1>`, and so on—is there to give directions to the browser. That's what markup does; it specifies directions on the way the content is to be interpreted.

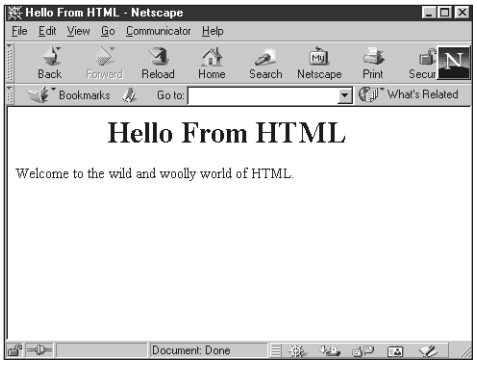


**Figure 1.1** An HTML page in a browser.

There is a real relationship between HTML and XML; both are based on *Standard Generalized Markup Language* (*SGML*). As its name implies, SGML is a very general markup language, with enormous capabilities. Because of the large number of things you can do with SGML, however, it can be very difficult to learn, and it hasn't caught on in general use. XML is actually an easier-to-use *subset* of SGML (and technically speaking, HTML is called an *application* of SGML). You can read more about the relationship between SGML and XML at `www.w3.org/TR/NOTE-sgml-xml`.

When you think of markup in terms of specifying how the content of a document is to be handled, it's easy to see that many kinds of markup languages abound already. For example, if you use a word processor to save a document in rich text format (RTF), you'll find all kinds of markup codes embedded in the document. Here's an example; in this case, I've just created an RTF file with the letters "abc" underlined and in bold using Microsoft Word. Try searching for the actual text (hint: it's near the very end):

```
{\rtf1\ansi\ansicpg1252\uc1 \deff0\deflang1033
deflangfe1033{\fonttbl{\f0\froman\fcharset0\fprq2{\*\panose\
020206030504050203040}Times New Roman;}}{\colortbl;\red0
green0\blue0;\red0\green0\blue255;\red0\green255\blue255;\
red0\green255\blue0;\red255\green0\blue255;\red255\green0\
blue0;\red255\green255\blue0;\red255\green255\blue255;\red0\
green0\blue128;\red0\green128\blue128;\red0\green128\blue0;\
red128\green0\blue128;\red128\green0\blue0;\red128\green128\
blue0;\red128\green128\blue128;\red192\green192\blue192;}
{\stylesheet{\widctlpar\adjustright \fs20\cgrid \snext0 Normal;}
{\*\cs10 \additive Default Paragraph Font;}}{\info{\title  }
{\author Steven Holzner}{\operator Steven Holzner}{\creatim
yr2000\mo\dy\hr\min}{\revtim\yr2000\mo4\dy17\hr13\min55}
{\version1}{\edmins1}{\nofpages1}{\nofwords0}{\nofchars1}
{\*\company SteveCo}{\nofcharsws1}{\vern89}}\widowctrl\ftnbj\
aenddoc\formshade\viewkind4\viewscale100\pgbrdrhead\pgbrdrfoot\
fet0\sectd \psz1\linex0\endnhere\sectdefaultcl {\*\pnseclvl1\
pnucrm\pnstart1\pnindent720\pnhang{\pntxta .}}{\*\pnseclvl2\
pnucltr\pnstart1\pnindent720\pnhang{\pntxta .}}{\*\pnseclvl3\
pndec\pnstart1\pnindent720\pnhang{\pntxta .}}{\*\pnseclvl4\
pnlcltr\pnstart1\pnindent720\pnhang{\pntxta )}}{\*\pnseclvl5\
pndec\pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta )}}
{\*\pnseclvl6\pnlcltr\pnstart1\pnindent720\pnhang{\pntxtb (}
{\pntxta )}}{\*\pnseclvl7\pnlcrm\pnstart1\pnindent720\pnhang
{\pntxtb (}{\pntxta )}}{\*\pnseclvl8\pnlcltr\pnstart1\
pnindent720\pnhang{\pntxtb (}{\pntxta )}}{\*\pnseclvl9\pnlcrm\
pnstart1\pnindent720\pnhang{\pntxtb (}{\pntxta )}}\pard\plain\
sl480\slmult1\widctlpar\adjustright \fs20\cgrid {\b\fs24\ulabc }
{\b\ul \par }}
```

The markup language that most people are familiar with these days is HTML, but it's easy to see how that language doesn't provide enough power for anything beyond creating standard Web pages.

HTML 1.0 consisted of only a dozen or so tags, but the most recent version, HTML 4.01, consists of almost 100—if you include the other tags added by the major browsers, that number is closer to 120. But as handling

data on the Web and other nets intensifies, it's clear that 120 tags isn't enough—in fact, you can never have enough.

For example, what if your hobby was building model ships, and you wanted to exchange specifications with others on the topic? HTML doesn't include tags such as <BEAMWIDTH>, <MIZZENHEIGHT>, <DRAFT>, <SHIPCLASS>, and others that you might want. What if you worked for a major bank and wanted to exchange financial data with other institutions—would you prefer tags such as <B>, <UL>, and <FONT> or tags such as <FISCALYEAR>, <ACCOUNTNUMBER>, and <TRANSFERACCOUNT>? (In fact, such markup languages, including Extensible Business Reporting Language, exist now and are built on XML.)

Likewise, what if you were a Web browser manufacturer who wanted to create your own markup language to let people configure your browser, adding scrollbars, toolbars, and other elements? You might create your own markup language. In fact, Netscape has done just that with the XML-based User Interface Language, which we'll see in this chapter.

The upshot is that there are as many reasons to create markup languages as there are ways of handling data—and, of course, both are unlimited numbers. That's where XML comes in: It's a meta-markup specification that lets you create your own markup languages.

## What Does XML Look Like?

So what does XML look like, and how does it work? Here's an example that mimics the HTML page just introduced:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

We'll see the parts of an XML document in detail in the next chapter, but in overview, here's how this one works: I start with the XML *processing instruction* <?xml version="1.0" encoding="UTF-8"?> (all XML processing instructions

start with <? and end with ?>), which indicates that I'm using XML version 1.0 (the only version currently defined) and the UTF-8 *character encoding*, an 8-bit condensed version of Unicode (more on this later in the chapter). Also, when I add new sections of code, they'll be highlighted with shading to point out the actual lines I'm discussing.

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


<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

Next, I create a new tag named `<DOCUMENT>`. As we'll see in the next chapter, you can use any name for a tag, not just DOCUMENT, as long as the name starts with a letter or an underscore (_), and as long as the following characters consist of letters, digits, underscores, dots (.), or hyphens (-), but no spaces. In XML, tags always start with < and end with >.

XML documents are made up of XML *elements*. Much like in HTML, you create XML elements with an opening tag, such as `<DOCUMENT>`, followed by the element content (if any), such as text or other elements, and ending with the matching closing tag that starts with </, such as `</DOCUMENT>`. (There are additional rules we'll see in the next chapter if the element doesn't contain any content.) It's necessary to enclose the entire document, except for processing instructions, in one element, called the *root element*—that's the `<DOCUMENT>` element here:

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

Now I'll add to this XML document a new element that I made up, `<GREETING>`, which encloses text content (in this case, `Hello From XML`), like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    .
    .
    .
</DOCUMENT>
```

Next, I can add a new element, <MESSAGE>, which also encloses text content, like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

Now the <DOCUMENT> root element contains two elements: <GREETING> and <MESSAGE>. Each of the <GREETING> and <MESSAGE> elements also hold text themselves. In this way, I've created a new XML document.

Note the similarity of this document to the HTML page we saw earlier. Note also, however, that in the HTML document, all the tags were predefined, and a Web browser knew how to handle them. Here, we've just created the tags <DOCUMENT>, <GREETING> and <MESSAGE> from thin air. How can we use an XML document like this one? What would a browser make of these new tags?

## What Does XML Look Like in a Browser?

It turns out that a browser such as Microsoft Internet Explorer version 5 or later lets you display raw XML documents directly. For example, if I saved the XML document we just created in a document named greeting.xml and then opened that document in the Internet Explorer, you'd see something like what appears in Figure 1.2.

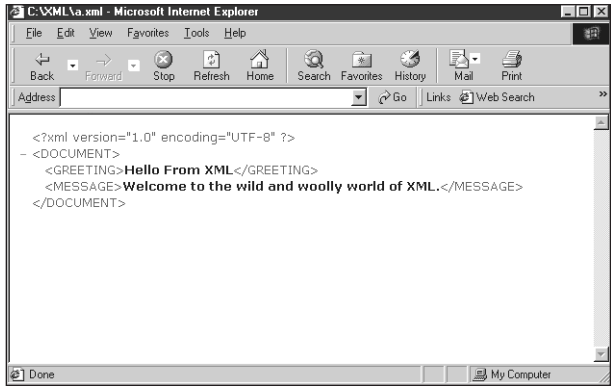


**Figure 1.2** An XML document in Internet Explorer.

You can see our complete XML document in Figure 1.2, but it's nothing like the image you see in Figure 1.1; there's no particular formatting at all. So, now that we've created our own markup elements, how do you tell a browser how to display them?

Many people who are new to XML find the claim that you can use XML to create new markup languages very frustrating—after all, what then? It turns out that it's up to you to assign meaning to the new elements you create, and you can do that in two main ways. First, you can use a *style sheet* to indicate to a browser how you want the content of the elements you've created to be formatted. The second way is to use a programming language, such as Java or JavaScript, to handle the XML document in programming code. We'll see both ways throughout this book, and I'll take a quick look at them in this chapter as well. I'll start by adding a style sheet to the XML document we've already created.

There are two main ways of specifying styles when you format XML documents: with cascading style sheets (CSS) and with the Extensible Style Sheets Language (XSL). We'll see both in this book; here, I'll apply a CSS style sheet by using the XML processing instruction `<?xml-stylesheet type="text/css" href="greeting.css"?>`, which tells the browser that the type of the style sheet I'll be using is CSS and that its name is greeting.css:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="greeting.css"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
```

*continues* ▸

```
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

Here's what the contents of the file greeting.css itself looks like. In this case, I'm customizing the `<GREETING>` element to display its content in red, centered in the browser, and in 36-point font. The `<MESSAGE>` element has been customized to display its text in black 18-point font. The `display: block` part indicates that I want the content of these elements to be displayed in a block, which translates here to being displayed on individual lines:

```
GREETING {display: block; font-size: 36pt; color: #FF0000; text-align: center}
MESSAGE (display: block; font-size: 18pt; color: #000000}
```

You can see the results in two browsers that support XML in Figures 1.3 and 1.4. Figure 1.3 shows greeting.xml in Netscape 6 (available only in a preview version as of this writing), and Figure 1.4 shows the same document in Internet Explorer. As you can see, we've formatted the document as we like it—in fact, this result already represents an advance over HTML because we can format exactly how we want to display the text instead of having to rely on predefined elements such as `<H1>`.


**Figure 1.3** An XML document in Netscape 6 (preview version).

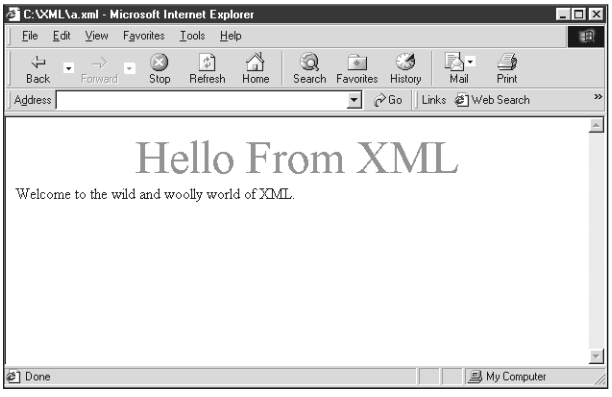


**Figure 1.4** An XML document in Internet Explorer.

That gives us a taste of XML. Now we've seen how to create a first XML document and how to use a style sheet to display it in a browser. We've seen what it looks like, so what's so great about XML? Take a look at the overview, coming up next.

# What's So Great About XML?

XML is popular for many reasons, and I'll examine some of them here as part of our overview of where XML is today. My own personal favorite is that XML allows easy data handling and exchange.

## Easy Data Exchange

I've been involved with computing for a long time, and one of the things I've watched with misgiving is the growth of proprietary data formats. In earlier days, programs could exchange data easily because data was stored as text. Today, however, you need conversion programs or modules to let applications transfer data between themselves. In fact, proprietary data formats have become so complex that frequently one version of a complex application can't even read data from an earlier version of the same application.

In XML, data and markup are stored as text that you can configure. If you like, you can use XML editors to create XML documents, but if something goes wrong, you can examine or modify the document directly because its all just text. The data also is not encoded in some way that has been patented or copyrighted, so it's more accessible.

You might think that binary formats would be more efficient because they can store data more compactly, but that's not the way things have worked out. For example, Microsoft Corporation is notorious for turning out huge applications that store even simple data in huge files (the not-so-affectionate name for this is "bloatware"). If you store only the letters "abc" in a Microsoft Word 97 document, you may be surprised to find that the document is something like 20,000 bytes long. A similar XML file might be 30 or 40 bytes. Even large amounts of data are not necessarily stored efficiently; for instance, Microsoft Excel routinely creates even larger files that are five times as long as the corresponding text. As we'll see, XML provides a very efficient way of storing most data.

In addition, when you standardize markup languages, many different people can use them. I'll take a look at that next.

## Customizing Markup Languages

As we've already seen, you can create customized markup languages using XML, and that represents its extraordinary power. When you and a number of other people agree on a markup language, you can create customized browsers or applications to handle that language. Hundreds of such languages already are being standardized now, including these:

- Banking Industry Technology Secretariat (BITS)
- Financial Exchange (IFX)
- Bank Internet Payment System (BIPS)
- Telecommunications Interchange Markup (TIM)
- Schools Interoperability Framework (SIF)
- Common Business Library (CBL)
- Electronic Business XML Initiative (ebXML)
- Product Data Markup Language (PDML)
- Financial Information eXchange protocol (FIX)
- The Text Encoding Initiative (TEI)

Some customized markup languages, such as Chemical Markup Language (CML), let you represent complex molecules graphically, as we'll see later in this chapter. Likewise, you can imagine how useful a language would be that creates graphical building plans for architects when you open a document in a browser.

Not only can you create custom markup languages, but you also can extend them using XML. If someone creates a markup language based on XML, you can add the extensions you want easily. In fact, that's happening now with Extensible Hypertext Markup Language (XHTML), which I'll take a look at briefly in this chapter and in detail later in the book. Using XHTML, you can add your own elements to what a browser displays as normal HTML.

## Self-Describing Data

The data in XML documents is self-describing. Take a look at this document:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

Based solely on the names we've given to each XML element here, you can figure out what's going on: This document has a greeting and a message to impart. Even if you came back to this document years later, you could figure out what's going on. This means that XML documents are, to a large extent, self-documenting. (We'll also see in the next chapter that you can add explicit comments to XML files.)

## Structured and Integrated Data

Another powerful aspect of XML is that it lets you specify not only data, but also the structure of that data and how various elements are integrated into other elements. This is important when you're dealing with complex and important data. For example, you could represent a long bank statement in

HTML, but in XML, you actually can build in the semantic rules that specify the structure of the document so that the document can be checked to make sure it's set up correctly.

Take a look at this XML document:

```
<?xml version="1.0"?>
<SCHOOL>
   <CLASS type="seminar">
        <CLASS_TITLE>XML In The Real World</CLASS_TITLE>
        <CLASS_NUMBER>6.031</CLASS_NUMBER>
        <SUBJECT>XML</SUBJECT>
        <START_DATE>6/1/2002</START_DATE>
        <STUDENTS>
            <STUDENT status="attending">
                <FIRST_NAME>Edward</FIRST_NAME>
                <LAST_NAME>Samson</LAST_NAME>
            </STUDENT>
            <STUDENT status="withdrawn">
                <FIRST_NAME>Ernestine</FIRST_NAME>
                <LAST_NAME>Johnson</LAST_NAME>
            </STUDENT>
        </STUDENTS>
   </CLASS>
</SCHOOL>
```

Here I've set up an XML seminar and added two students to it. As we'll see in Chapter 2, "Creating Well-Formed XML Documents," and Chapter 3, "Valid XML Documents: Creating Document Type Definitions," with XML you can specify, for example, that each `<STUDENT>` element needs to enclose a `<FIRST_NAME>` and a `<LAST_NAME>` element, that the `<START_DATE>` element can't go in the `<STUDENTS>` element, and more.

In fact, this emphasis on the correctness of documents is strong in XML. In HTML, a Web author could (and frequently did) write sloppy HTML, knowing that the Web browser would take care of any syntax problems (some Web authors even exploited this intentionally to create special effects in some browsers). In fact, some people estimate that 50% or more of the code in modern browsers is there to take care of sloppy HTML in Web pages. For that kind of reason, the story is different in XML. In XML, browsers are supposed to check your document; if there's a problem, they are not supposed to proceed any further. They should let you know about the problem, but that's as far as they're supposed to go.

So, how does an XML browser check your document? XML browsers can make two main checks: a check to see whether your document is *well-formed*, and a check to see whether it's *valid*. We'll see what these terms mean in more detail in the next chapter, and I'll go over them briefly here.

## Well-Formed XML Documents

What does it mean for an XML document to be well-formed? To be well-formed, an XML document must follow the syntax rules set up for XML by W3C in the XML 1.0 specification (which you can find at `www.w3.org/TR/REC-xml`, and which we'll discuss in more detail in the next chapter). Informally, well-formedness often means that the document must contain one or more elements, and one element, the root element, must contain all the other elements. Each element also must nest inside any enclosing elements properly. For example, this document is not well-formed because the `</GREETING>` closing tag comes after the opening `<MESSAGE>` tag for the next element:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    <MESSAGE>
    </GREETING>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

## Valid XML Documents

Most XML browsers will check your document to see whether it is well-formed. Some of them also can check whether it's valid. An XML document is valid if there is a document type definition (DTD) associated with it, and if the document complies with that DTD.

A document's DTD specifies the correct syntax of the document, as we'll see in Chapter 3. DTDs can be stored in a separate file or in the document itself, using a `<!DOCTYPE>` element. Here's an example in which I add a `<!DOCTYPE>` element to the greeting XML document we developed earlier:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="first.css"?>
<!DOCTYPE DOCUMENT [
    <!ELEMENT DOCUMENT (GREETING, MESSAGE)>
    <!ELEMENT GREETING (#PCDATA)>
    <!ELEMENT MESSAGE (#PCDATA)>
]>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

We'll see more about DTDs in Chapter 3, but this DTD indicates that you can have `<GREETING>` and `<MESSAGE>` elements inside a `<DOCUMENT>` element, that the `<DOCUMENT>` element is the root element, and that the `<GREETING>` and `<MESSAGE>` elements can hold text.

Most XML browsers will check XML documents for well-formedness, but only a few will check for validity. I'll talk more about where to find XML validators in the later section "XML Validators."

We've gotten an overview of XML documents now, including how to display them using style sheets, and what constitutes a well-formed and valid document. However, this is only part of the story. Many XML documents are not designed to be displayed in browsers at all, for example, or even if they are, they're are not designed to be used with modern style sheets (such as browsers that convert XML into industry-specific graphics such as molecular structure, physics equations, or even musical scales). The more powerful use of XML involves *parsing* an XML document to break it down into its component parts and then handling the resulting data yourself. Next I'll take a look at a few ways of parsing XML data that are available to us.

## Parsing XML Yourself

Let's say that you have this XML document, greeting.xml, which we developed earlier in this chapter:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

Now say that you want to extract the greeting `Hello From XML` from this XML document. One way of doing that is by using XML data islands in Internet Explorer and then using a scripting language, such as JavaScript, to extract and display the text content of the `<GREETING>` element. Here's how that looks in a Web page:

```
<HTML>
    <HEAD>
        <TITLE>
            Finding Element Values in an XML Document
        </TITLE>

        <XML ID="firstXML" SRC="greeting.xml"></XML>

        <SCRIPT LANGUAGE="JavaScript">
            function getData()
            {
                xmldoc= document.all("firstXML").XMLDocument;

                nodeDoc = xmldoc.documentElement;
                nodeGreeting = nodeDoc.firstChild;

                outputMessage = "Greeting: " +
                    nodeGreeting.firstChild.nodeValue;
                message.innerHTML=outputMessage;
            }
        </SCRIPT>
    </HEAD>

    <BODY>
        <CENTER>
            <H1>
                Finding Element Values in an XML Document
            </H1>
```

*continues* ▸

```
            <DIV ID="message"></DIV>
            <P>
            <INPUT TYPE="BUTTON" VALUE="Get The Greeting"
                ONCLICK="getData()">
        </CENTER>
    </BODY>
</HTML>
```

This Web page displays a button with the caption "Get The Greeting," as you see in Figure 1.5. When you click the button, the JavaScript code in this page reads in greeting.xml, extracts the text from the `<GREETING>` element, and displays that text, as you also can see in Figure 1.5. In this way, you can see how you can create applications that handle XML documents in a customized way—even customized XML browsers.

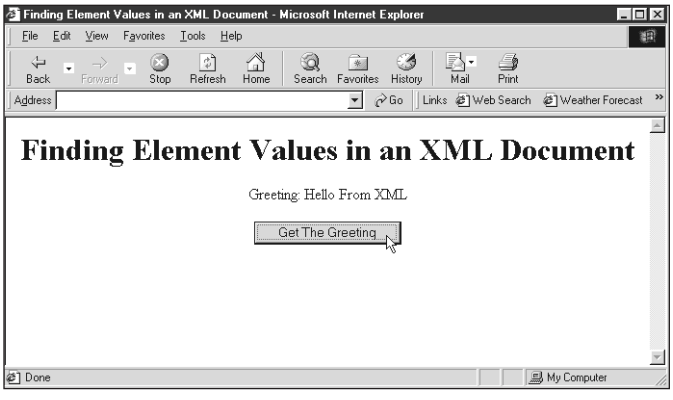


**Figure 1.5** Extracting data from an XML document in Internet Explorer.

We'll see more about using JavaScript to work with XML later in this book, and I'll also cover how JavaScript itself works first. If you haven't used JavaScript before, you won't have a problem doing so now.

Although JavaScript is useful for lightweight XML uses, most XML programming is done in Java today, and we'll take advantage of that in this book. Here's an example Java program, readXML.java, using XML4J, probably the most widely used XML parser, which is free from IBM's AlphaWorks. This program also reads greeting.xml and extracts the text content of the `<GREETING>` element:

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.Text;

public class readXML
{
    static public void main(String[] argv)
    {
        try {
            DOMParser parser = new DOMParser();
            parser.parse("greeting.xml");
            Document doc = parser.getDocument();

            for (Node node = doc.getDocumentElement().getFirstChild();
                node != null; node = node.getNextSibling()) {

                if (node instanceof Element) {
                    if (node.getNodeName().equals("GREETING")) {

                        StringBuffer buffer = new StringBuffer();

                        for (Node subnode = node.getFirstChild();
                            subnode != null; subnode =
                                subnode.getNextSibling()){
                            if (subnode instanceof Text) {
                                buffer.append(subnode.getNodeValue());
                            }
                        }
                        System.out.println(buffer.toString());
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

When you compile and run this program (you'll see how to do so in Chapter 11), the output looks like this. (I'll use "%" to represent the command-line prompt in this book; if you're using UNIX, this prompt

may look familiar, or your prompt may look something like `/home/steve:`. If you're using Windows, you get a command-line prompt by opening an MS-DOS window, and your prompt may look something like `C:\XML>`.)

```
%java readXML
        Hello From XML
```

(Note that this program returns all the text in the `<GREETING>` element, including the leading spaces.) We'll see how to use Java to parse XML documents later in this book, mostly using the IBM AlphaWorks XML4J parser, which adheres closely to the W3C Document Object Model. I'll also cover the Java you need to know before getting into the programming, so if you haven't programmed in Java before, that's no problem.

We now have a good overview of how XML works. It's time to take a look at how it's already working in the real world, starting with an overview of the XML resources available to you.

## XML Resources

Many XML resources are available to you online. Because it's very important that you know about them to get a solid background in XML, I list them here.

The XML specification is defined by W3C, and that's where you should start looking for XML resources. Here's a good starter list (we'll see all these topics in this book):

- `www.w3c.org/xml`. W3C's main XML site, the starting point for all things XML.
- `www.w3.org/XML/1999/XML-in-10-points`. "XML in 10 Points" (actually only seven), an XML overview.
- `www.w3.org/TR/REC-xml`. The official W3C recommendation for XML 1.0, the current (and only) version. It won't be easy to read, however—that's what this book is all about, translating that kind of document to English.
- `www.w3.org/TR/xml-stylesheet/`. All about using style sheets and XML.
- `www.w3.org/TR/REC-xml-names/`. All about XML namespaces.
- `www.w3.org/Style/XSL/`. All about Extensible Style Language (XSL).
- `www.w3.org/TR/xslt`. All about XSL transformations (XSLT).
- `www.w3.org/XML/Activity.html`. An overview of current XML activity at W3C.

- `www.w3.org/TR/xmlschema-0/`, `www.w3.org/TR/xmlschema-1/`, and `www.w3.org/TR/xmlschema-2/ XML`. Information on schemas, the alternative to DTDs.
- `www.w3.org/TR/xlink/`. The XLinks specification.
- `www.w3.org/TR/xptr`. The XPointers specification.
- `www.w3.org/TR/xhtml1/`. The XHTML 1.0 specification.
- `www.w3.org/TR/xhtml11/`. The XHTML 1.1 specification.
- `www.w3.org/DOM/`. The W3C Document Object Model, (DOM).

Many non-W3C XML resources are out there, too—a casual search for "XML" on the Web turns up a mere 561,870 pages. Here's a list to get you started:

- `www.xml.com; XML.com`. A site filled with XML resources, discussions, and notifications of public events.
- `www.xml-zone.com`. Excellent XML overviews and listings of events.
- `www.oasis-open.org`. The Organization for the Advancement of Structured Information Standards (OASIS) is dedicated to the adoption of product-independent formats such as XML.
- `www.xml.org`. XML.ORG is designed to provide information about the use of XML in industrial and commercial settings. It's hosted by OASIS and is a reference for XML vocabularies, DTDs, schemas, and namespaces.
- `http://msdn.microsoft.com/xml/default.asp`. Microsoft's XML page.

You'll also find quite a few XML tutorials online (searching for "XML Tutorial" brings up more than 500 matches). Here are a few to start with:

- `www2.software.ibm.com/developer/education.nsf/xml-onlinecourse-bytitle`. IBM's free tutorials.
- **`www.ucc.ie/xml/`.** A comprehensive frequently asked questions (FAQ) list about XML, kept up by some of the contributors to the W3C's XML Working Group. This is considered by many to be the definitive FAQ on XML.
- `msdn.microsoft.com/xml/tutorial/default.asp`. Microsoft's XML tutorial.
- `www.xml.com/pub/98/10/guide0.html`. XML.com's XML overview.
- `web2.javasoft.com/xml/docs/tutorial/TOC.html`. JavaSoft's XML tutorial.

In addition, you might find some newsgroups on Usenet useful (note that your news server may not carry all these groups):

- `comp.text.xml`. A good, general-purpose, free-floating XML forum.
- `microsoft.public.inetexplorer.ie5beta.programming.xml`. XML discussions and questions concerning Internet Explorer 5.
- `microsoft.public.xml`. The general Microsoft XML forum.

That's a good start on XML resources available on the Internet. What about XML software? Let's take a look at what's out there, starting with XML editors.

# XML Editors

To create the XML documents we'll use in this book, all you need is a text editor of some kind, such as vi, emacs, pico, Windows Notepad, or Windows WordPad. By default, XML documents are supposed to be written in Unicode, although in practice you can write them in ASCII, and nearly all of them are written that way so far. Just make sure that when you write an XML document, you save it in your editor's plain-text format.

**Using Windows Text Editors**

Windows text editors such as WordPad or Notepad have an annoying habit of appending the extension .txt to a filename if they don't understand the extension you've given the file. That's not a problem with .xml files, though, because WordPad understands the extension .xml. For example, if you try to save an XML-based User Interface Language document with the correct extension of .xul, WordPad will give it the extension .xul.txt. To avoid that, place the name of the file in quotation marks when you save it, as in "scrollbars.xul."

However, it can be a lot easier to use an actual XML editor, which is designed explicitly for the job of handling XML. Here's a list of some programs that let you edit your XML:

- **Adobe FrameMaker,** `www.adobe.com`. Great, but expensive, XML support in FrameMaker.
- **XML Pro,** `www.vervet.com/`. A costly but powerful XML editor.
- **XML Writer, on disk, XMLWriter,** `http://xmlwriter.net/`. Color syntax highlighting, with a nice interface.
- **XML Notepad,** `msdn.microsoft.com/xml/notepad/intro.asp`. Microsoft's free XML editor, a little difficult to use.

- **eNotepad,** `www.edisys.com/Products/eNotepad/enotepad.asp`**.** A WordPad replacement that does well with XML and has a good user interface.
- **XMetal from SoftQuad,** `xmetal.com`**.** An expensive but very powerful XML editor, and many authors' editor of choice.
- **XML Spy,** `www.xmlspy.com/`**.** A good and easy-to-use user interface.

You can see XML Spy at work in Figure 1.6, XML Writer in Figure 1.7, XML Notepad in Figure 1.8, and eNotepad in Figure 1.9.

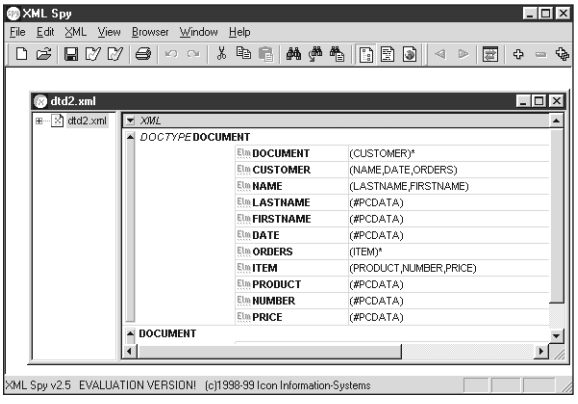


**Figure 1.6** XML Spy editing XML.0

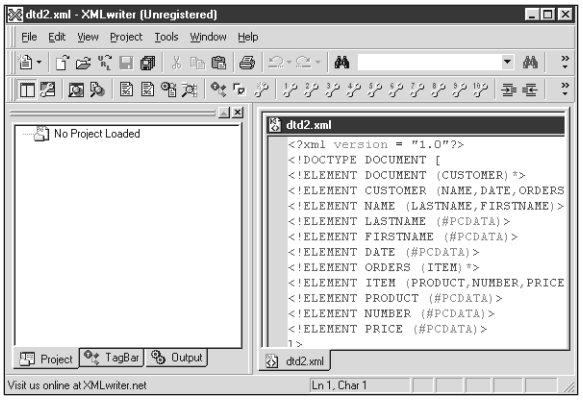


**Figure 1.7** XML Writer editing XML.

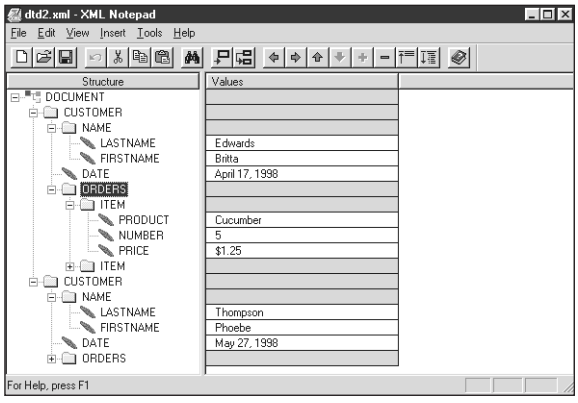


**Figure 1.8** XML Notepad editing XML.

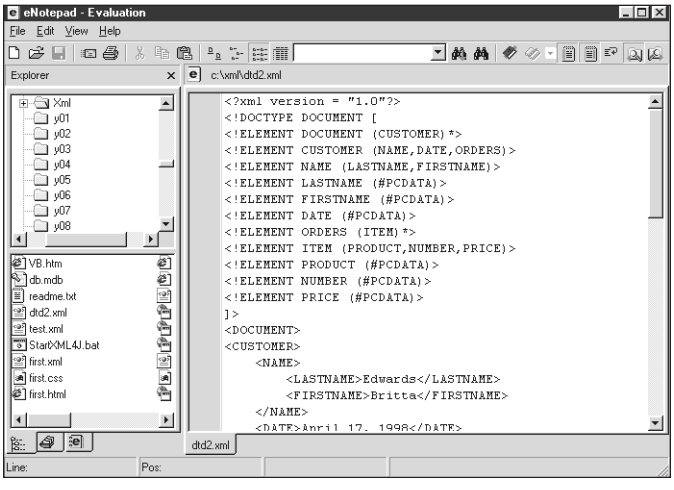


**Figure 1.9** eNotepad editing XML.

Now that we've gotten an overview of creating XML documents, what about XML browsers? The list is more limited, but there are a few out there. See the next topic.

# XML Browsers

Creating a true XML browser is not easy. Besides supporting XML, the browser would have to support a style language such as CSS or XSL. It also should support a scripting language, such as JavaScript. These are heavy

requirements for most third-party vendors, so the true XML browsers out there are few. In fact, no complete, general XML browsers currently exist. None of the browsers listed here validate XML documents—they just check for well-formedness—but a few come close.

## Internet Explorer 5

Whether you love or hate Microsoft, the fact remains that Internet Explorer is the most powerful XML browser available now; you currently can get it at `www.microsoft.com/windows/ie/default.htm`.

Internet Explorer can display XML documents directly, as you saw in Figure 1.2, and also can handle them in scripting languages (JScript, Microsoft's version of JavaScript, and Microsoft's VBScript are supported). There is good support for style sheets and other features such as the `<XML>` element, which lets you create XML data islands into which you can load XML documents, and ways of binding XML to ActiveX Data Object (ADO) database recordsets.

Internet Explorer 5.5, in preview at this writing, also supports additional XML features, such as the XPath specification. There's no question that Microsoft's XML commitment is strong—XML has been integrated even into the Office 2000 suite of applications—but Microsoft sometimes swerves significantly from the W3C standards (that is, when Microsoft isn't writing those standards itself).

## Netscape Navigator 6

Netscape has just released the preview version of Netscape Navigator 6 (available at `www.netscape.com/download/previewrelease.html`), which has significant XML support. You can see Netscape Navigator 6 at work back in Figure 1.3. This preview version is based on Netscape's open source Mozilla browser, which you can pick up at `www.mozilla.org`. Unfortunately, both Mozilla and the preview version of Netscape Navigator 6 have a reputation for crashing machines frequently.

As with Internet Explorer, support for style sheets is good in Netscape Navigator. The preview version of Netscape Navigator 6 also supports the XML-based User Interface Language (XUL), which lets you configure the controls in the browser. In fact, the preview version's user interface is based on XUL. More XML features will come in Netscape 6, but right now documentation is virtually nonexistent.

### Jumbo

One of the more famous true XML browsers that exist is Jumbo, an XML browser designed to work with XML and the CML. You can pick up Jumbo for free at `www.xml-cml.org/jumbo.html`. This browser not only can display XML (although not with style sheets), but it also can use CML to draw molecules, as you see in Figure 1.10.

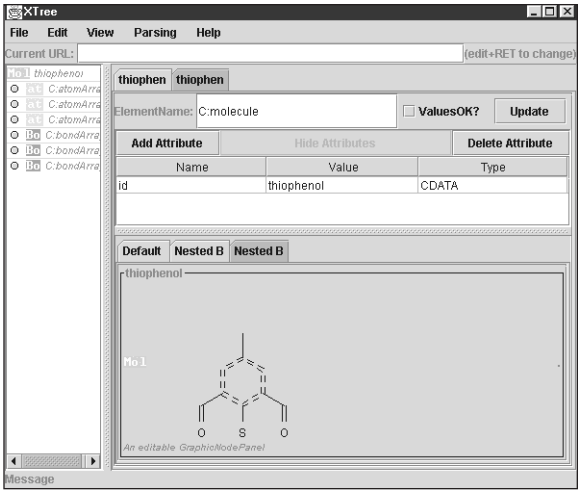


Figure 1.10 Jumbo at work.

Relatively few real XML browsers exist, but there *are* a large number of XML parsers. You can use these parsers to read XML documents and break them up into their component parts.

## XML Parsers

XML *parsers* are software packages that you use as part of an application such as Oracle 8i (which includes good XML support) or as part of your own programs. For example, later in this book, I'll use the IBM AlphaWorks XML for Java (XML4J) parser; it is written in Java and connects well to your own Java code. Here's a list of some of the parsers out there:

- **SAX: The Simple API for XML.** Written by David Megginson et al. (`www. megginson.com/SAX/index.html`), SAX is a well-known parser that uses event-based parsing. I'll use SAX in this book.

- **expat.** This famous XML parser was written in the C programming language by James Clark (`www.jclark.com/xml/expat.html`). This parser is used in Netscape Navigator 6 and in the Perl language's XML::Parser module.
- **expat as a Perl Module.** XML::Parser is maintained by Clark Cooper (`ftp://ftp.perl.org/pub/CPAN/modules/by-module/XML/`).
- **TclExpat.** This is expat written for use in the Tcl programming language by Steve Ball. Superceded by TclXML. (`www.zveno.com/zm.cgi/in-tclxml`).
- **LT XML.** This is an XML developers' toolkit from the Language Technology Group at the University of Edinburgh (`www.ltg.ed.ac.uk/software/xml/`).
- **XML for Java –(XML4J).** From IBM AlphaWorks (`www.alphaworks.ibm.com/tech/xml4j`), this is a famous and very widely used XML parser that adheres well to the W3C standards.
- **XML Microsoft's validating XML processor.** This parser requires Internet Explorer 4.01 SP1 and later in order to be fully functional. In addition to various tools, samples, tutorials, and online documentation, this can be found at `msdn.microsoft.com/xml/default.asp`.
- **Lark.** This is a nonvalidating XML processor that was written in Java by Tim Bray (`www.textuality.com/Lark/`), and it's one of the famous ones that have been around a long time.
- **XP.** XP is a nonvalidating XML processor written in Java by James Clark (`www.jclark.com/xml/xp/index.html`).
- **Python and XML Processing Preliminary XML Parser.** This parser offers XML support to the Python programming language (`www.python.org/topics/xml/`).
- **TclXML.** This XML parser was written in Tcl by Steve Ball (`www.zveno.com/zm.cgi/in-tclxml/`).
- **XML Testbed.** This parser was written by Steve Withall (`www.w3.org/XML/1998/08withall/`).
- **SXP.** Silfide XML Parser (SXP) is another famous XML parser and, in fact, a complete XML Application Programming Interface (API) in Java (`www.loria.fr/projets/XSilfide/EN/sxp/`).
- **The Microsoft XML Parser.** The parser used in Internet Explorer is implemented as a COM component at `www.msdn.microsoft.com/downloads/tools/xmlparser/xmlparser.asp`.

- **OmiMark 5 Programming Language.** Includes integrated support for parsing and validation of XML (`www.omnimark.com/`).
- **Java Standard Extension for XML.** Because XML and Sun Microsystem's Java is such a popular mix, Sun is getting into the act with its own Java package for XML (`java.sun.com/products/xml/`).

Parsers will break up your document into its component pieces and make them accessible to other parts of a program; some parsers also check for well-formedness and fewer check for document validity. However, if you just want to check whether your XML is both well-formed and valid, all you need is an XML validator.

## XML Validators

How do you know whether your XML document is well-formed and valid? One way is to check it with an XML *validator*, and you have plenty to choose from. Validators are packages that will check your XML and give you feedback. For example, if you have the XML for Java parser from IBM's AlphaWorks installed, you can use the DOMWriter example as a complete XML validator. Let's say you wanted to check this document, greeting.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

To do this, you'd set things up for the XML4J package (we'll see how to do so later in the book) and run the DOMWriter sample on it, like this:

```
%java dom.DOMWriter greeting.xml
greeting.xml:
[Error] greeting.xml:2:11: Element type "DOCUMENT" must be declared
[Error] greeting.xml:3:15: Element type "GREETING" must be declared
[Error] greeting.xml:6:14: Element type "MESSAGE" must be declared.
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
```

```
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

If all goes well, DOMWriter simply displays the document you've asked it to validate, but if there are errors, it will display them. Here, DOMWriter is indicating that because we haven't included a DTD in greeting.xml, it can't check for the validity of the document.

That's fine if you have the XML for Java package installed, but more accessible XML validators are available to you as well. Here's a list of some of the XML validators on the Web:

- **W3C XML Validator,** `validator.w3.org/`**.** This is the official W3C HTML validator. Although it's officially for HTML, it also includes some XML support. Your XML document must be online to be checked with this validator.
- **Tidy,** `www.w3.org/People/Raggett/tidy/`**.** Tidy is a beloved utility for cleaning up and repairing Web pages, and it includes limited support for XML. Your XML document must be online to be checked with this validator.
- `www.xml.com/xml/pub/tools/ruwf/check.html`**.** This is XML.com's XML validator based on the Lark processor. Your XML document must be online to be checked with this validator.
- `www.ltg.ed.ac.uk/~richard/xml-check.html`**.** This is the Language Technology Group at the University of Edinburgh's validator, based on the RXP parser. Your XML document must be online to be checked with this validator.
- `www.stg.brown.edu/service/xmlvalid/`**.** This is an excellent XML validator from the Scholarly Technology Group at Brown University. This is the only online XML validator I know of that allows you to check XML documents that are not online. You can use the Web page's file upload control to specify the name of the file on your hard disk that you want to have uploaded and checked.

To see a validator at work, take a look at Figure 1.11. There, I'm asking the XML validator from the Scholarly Technology Group to validate this XML document, c:\xml\greeting.xml. I've intentionally exchanged the order of the `<MESSAGE>` and `</GREETING>` tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DOCUMENT [
    <!ELEMENT DOCUMENT (GREETING, MESSAGE)>
    <!ELEMENT GREETING (#PCDATA)>
    <!ELEMENT MESSAGE (#PCDATA)>
]>
<DOCUMENT>
    <GREETING>
        Hello From XML
    <MESSAGE>
    </GREETING>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

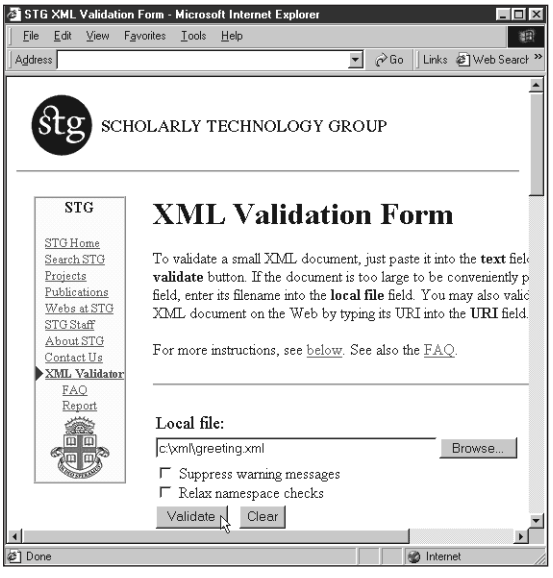


**Figure 1.11** Using an XML validator.

You can see the results in Figure 1.12. As you can see, the validator is indicating that there is a problem with these two tags.

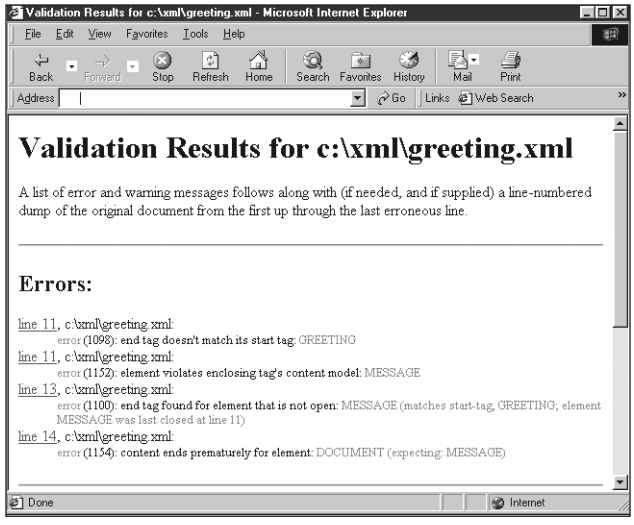


**Figure 1.12** The results from an XML validator.

XML validators give you a powerful way of checking your XML documents. That's useful because XML is much stricter than HTML about making sure that a document is correct. (Recall that XML browsers are not supposed to make attempts to fix XML documents if they find a problem; they're just supposed to stop loading the document.)

We've gotten a good overview of XML already in this chapter. In a few pages, I'll start taking a look at a number of XML languages that are already developed. But there are a few more useful topics to cover first, especially if you have programmed in HTML and want to know the differences between XML and HTML.

## CSS and XSL

Style sheets are becoming increasingly important in HTML because, in HTML 4, many built-in style features such as the `<CENTER>` element have become deprecated (declared obsolete) in favor of style sheets. However, most HTML programming ignores style sheets entirely.

The story is different in XML because you create your own elements in XML. Thus, if you want a browser to display them, you have to tell it how. This is both good and bad: It's good because it enables you to use the powerful CSS and XSL specifications to customize the appearance of your XML elements far beyond what's possible with standard HTML. It's bad because it can demand a lot of additional work. (One way of getting around the necessity of designing your own style sheets is to use an established XML language that has its own style sheets.)

All this is to say that XML defines the structure and semantics of the document, not its format; if you want to display XML directly, you can either use the default presentation in Internet Explorer, or use a style sheet to set up the presentation yourself.

You have two main ways to specify a style sheet for an XML document: with CSS and with XSL, both of which I'll dig into in this book. CSS is popular with those creating HTML documents and is widely supported. Using CSS, you can specify the formatting of individual elements, create style classes, set up fonts, use colors, and even specify placement of elements in the page.

XSL, on the other hand, is ultimately a better choice to work with XML documents because it's more powerful (in fact, XSL style sheets themselves are well-formed XML documents). XSL documents are made up of rules that are applied to XML documents. When a pattern that you've specified in the XSL document is recognized in the XML document, the rules transform the matched XML into something entirely new. You can even transform XML into HTML in this way.

Although CSS can set only the format and placement of elements, XSL can reorder elements in a document, change them entirely, display some but hide others, select styles based not just on elements but also on element attributes (XML elements can have attributes just as HTML elements can, and I'll introduce them in the next chapter), select styles based on element location, and much more. There are two ways to approach XSL: with XSL transformations and with XSL formatting objects. We'll take a look at both in this book.

Here are some good online resources for style sheets that provide a good reference:

- `www.w3.org/Style/CSS/`. The W3C outline and overview of CSS programming.
- `www.w3.org/TR/REC-CSS1`. The W3C CSS1 specification.

- www.w3.org/TR/REC-CSS2. The W3C CSS2 specification.
- www.w3.org/Style/XSL/. The W3C XSL page.

## XLinks and XPointers

It's hard to imagine the World Wide Web without hyperlinks; of course, HTML documents excel at letting you link from one page to another. How about XML? In XML, it turns out, you use XLinks and XPointers.

XLinks let any element become a link, not just a single element such as the HTML <A> element. That's a good thing because XML doesn't have a built-in <A> element. In XML, you define your own elements, and it only makes sense that you can define which of those represent links to other documents.

In fact, XLinks are more powerful than simple hyperlinks. XLinks can be bidirectional, allowing the user to return after following a link. They can even be multidirectional—in fact, they can be sophisticated enough to point to the nearest mirror site from which a resource can be fetched.

XPointers, on the other hand, point not to a whole document, but to a part of a document. In fact, XPointers are smart enough to point to a specific element in a document, or the second instance of such an element, or the 11,904th instance. They can even point to the first child element of another element, and so on. The idea is that XPointers are powerful enough to locate specific parts of another document without forcing you to add additional markup to the target document.

On the other hand, note that the whole idea of XLinks and XPointers is relatively new and is not fully implemented in any browser. We will see what's possible today later in this book.

Here are some XLink and XPointer references online—take a look for more information on these topics:

- www.w3.org/TR/xlink/. The W3C XLink page.
- www.w3.org/TR/xptr. The W3C XPointer page.

## URLs versus URIs

Having discussed XLinks and XPointers, I should also mention that the XML specification expands the idea of standard uniform resource locators (URLs) into uniform resource identifiers (URIs).

URLs are well understood and well supported on the Internet today. On the other hand (as you'd expect, given the addition of XLinks and XPointers to XML), the idea of URIs is more general than with simple URLs.

URIs let you represent a way of finding resources on the Internet, and they center more on the resource than the actual location. The idea is that, in theory, URIs can locate the nearest mirror site for a resource or even track a document that has been moved from one location to another.

In practice, the concept of URIs is still being developed, and most software still handles only URLs.

## ASCII, Unicode, and the Universal Character System

The actual characters in documents are stored as numeric codes. The most common code set today is the American Standard Code for Information Interchange (ASCII). ASCII codes extend from 0 to 255 (to fit within a single byte); for example, the ASCII code for "A" is 65, the ASCII code for "B" is 66, and so on.

On the other hand, the World Wide Web is just that today: worldwide. Plenty of scripts are not handled by ASCII, such as scripts in Bengali, Armenian, Hebrew, Thai, Tibetan, Japanese Katakana, Arabic, Cyrillic, and other languages.

For that reason, the default character set specified for XML by W3C is Unicode, not ASCII. Unicode codes are made up of 2 bytes, not 1, so they extend from 0 to 65,535, not just 0 to 255. (However, to make things easier, the Unicode codes 0 to 255 do correspond to the ASCII 0 to 255 codes.) Therefore, Unicode can include many of the symbols commonly used in worldwide character and ideograph sets. You can find more on Unicode at `www.unicode.org`.

Only about 40,000 Unicode codes are reserved at this point (of which about 20,000 codes are used for Han ideographs, although more than 80,000 such ideographs are defined; 11,000 are used for Korean Hangul syllables).

In practice, Unicode support, like many parts of the XML technology, is not fully supported on most platforms today. Windows 95/98 does not offer full support for Unicode, although Windows NT and Windows 2000 come much closer (and XML Spy lets you use Unicode to write XML documents in Windows NT). Most often, this means that XML documents are written in simply ASCII, or in UTF-8, which is a compressed version of Unicode that uses 8 bits to represent characters. (In practice, this is well suited to

ASCII documents because multiple bytes are needed for many non-ASCII symbols, and ASCII documents converted to Unicode are two times as long.) Here's how to specify the UTF-8 character encoding in an XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
    <GREETING>
        Hello From XML
    </GREETING>
    <MESSAGE>
        Welcome to the wild and woolly world of XML.
    </MESSAGE>
</DOCUMENT>
```

The default for XML processors today is to assume that your document is in UTF-8, so if you omit the encoding specification, UTF-8 is assumed. If you're writing XML documents in ASCII, you'll have no trouble.

Actually, not even Unicode has enough space for all symbols in common use. A new specification, the Universal Character System (UCS, also called ISO 10646) uses 4 bytes per symbol, which gives it a range of two billion symbols, far more than needed. You can specify that you want to use pure Unicode encoding in your XML documents by using the UCS-2 encoding (also called ISO-10646-UCS-2), which is compressed 2-byte UCS. You also can use UTF-16, which is a special encoding that represents UCS symbols using 2 bytes so that the result corresponds to UCS-2. Straight UCS encoding is referred to as UCS-4 (also called ISO-10646-UCS-4).

I'll stick to ASCII for most XML documents in this book because support for Unicode and UCS is not yet widespread. For example, I know of no true Unicode editors. On the other hand, you can write documents in a local character set and use a translation utility to convert them to Unicode, or you can insert the actual Unicode codes directly into your documents. For example, the Unicode for π is 03C0 in hexadecimal, so you can insert π into your document with the character entity (more on entities in the next chapter) `&#x03C0;`.

More character sets are available than those mentioned here; for a longer list, take a look at the list posted by the Internet Assigned Numbers Authority (IANA) at `www.isi.edu/in-notes/iana/assignments/character-sets`.

**Converting ASCII to Unicode**

If you want to convert ASCII files to straight Unicode, you can use the native2ascii program that comes with Sun Microsystem's Java Software Development Kit (the SDK, formerly the JDK). Using this tool, you can convert to Unicode like this: `native2ascii file.txt file.uni`. You also can convert to a number of other encodings besides Unicode, such as compressed Unicode, UTF-8.

# XML Applications

We've seen a lot of theory in this chapter, so I'm going to spend the rest of this chapter taking a look at how XML is used in the real world. The world of XML is huge these days; in fact, XML is now used internally even in Netscape and Microsoft products, as well as installations of programming languages such as Perl. You can find a good list of organizations that produce their own XML-based languages at `www.xml.org/xmlorg_catalog.htm`.

It's useful and encouraging to see how XML is being used today in these XML-based languages. It's a new piece of terminology: As you know, XML is a meta-markup language, so it's actually used to create languages. The languages so created are applications of XML, so they're called *XML applications*.

Note that the term *XML application* refers to an application of XML to a specific domain, such as MathML, the mathematics markup language; it does not refer to a program that uses XML (a fact that causes a lot of confusion among people who know nothing about XML).

Thousands of XML applications exist today, and we'll see some of them here. You can see the advantage to various groups (such as physicists or chemists) for defining their own markup languages, allowing them to use the symbols and graphics of their discipline in customized browsers. I'll start with discussing CML.

## XML at Work: Chemical Markup Language

Peter Murray-Rust developed CML as a very early XML application, so it has been around quite a while. Many people think of CML as a sort of HTML+Molecules, and that's not a bad characterization. Using CML, you can display the structure of complex molecules.

With CML, chemists can create and publish molecule specifications for easy interchange. Note that the real value of this is not so much in looking at individual chemicals as it is in being able to search CML repositories for molecules matching specific characteristics.

I've already mentioned Jumbo, a famous CML browser that you can download for free from www.xml-cml.org/jumbo.html. Jumbo not only works for handling CML, but you also can use it to display the structure of an XML document in general. However, there's no question that the novelty of Jumbo is that it can use CML to create graphical representations of molecules.

We've already seen an example in Jumbo in Figure 1.10, in which Jumbo is displaying the molecule thiophenol. Here is the file, thiophenol.xml, that it's reading to display that molecule (this document is an example that comes with the Jumbo browser):

```
<?jumbo:namespace ns="http://www.xml-cml.org" prefix="C"
    java="jumbo.cmlxml.*Node" ?>
<C:molecule id="thiophenol">
    <C:atomArray builtin="elsym">
        C C C C C C S C C O O
    </C:atomArray>
    <C:atomArray builtin="x2" type="float">
        0 0.866 0.866 0 -0.866 -0.866
        0.0 0.0  1.732 -1.732 1.732 -1.732
    </C:atomArray>
    <C:atomArray builtin="y2" type="float">
        1 0.5   -0.5  -1.0 -0.5   0.5
        -2.0  2.0  1.0   1.0    2.0   2.0
    </C:atomArray>
    <C:bondArray builtin="atid1">
        1 2 3 4 5 6 1 4 2 9  6  10
    </C:bondArray>
    <C:bondArray builtin="atid2">
        2 3 4 5 6 1 8 7 9 11 10 12
    </C:bondArray>
    <C:bondArray builtin="order" type="integer">
        4 4 4 4 4 4 1 1 1 2  1  2
    </C:bondArray>
</C:molecule>
```

## XML at Work: Mathematical Markup Language

Mathematical Markup Language was designed to fill a significant gap in Web documents: equations. In fact, Tim Berners-Lee first developed the World Wide Web at CERN so that high-energy physicists could exchange papers and documents. Still, there has been no way to display true equations in Web browsers for nearly a decade.

MathML fixes that. MathML is itself a W3C specification, and you can find it at `www.w3.org/Math/`. Using MathML, you can display equations and all kinds of mathematical terms. It's not powerful enough for many specialized areas of the sciences or mathematics yet, but it's growing all the time.

Because of the limited audience for this kind of presentation, no major browser yet supports MathML. However, there is the Amaya browser, which is W3C's own testbed browser for testing new HTML and XHTML elements (unfortunately, it's not an XML browser). You can download Amaya for free from `www.w3.org/Amaya/`.

Here's a MathML document that displays the equation $3Z^2 + 6Z + 12 = 0$ (this document uses an XML namespace, which we'll see more about in the next chapter):

```
<?xml version="1.0"?>
<html xmlns:m="http://www.w3.org/TR/REC-MathML/">
<math>
    <m:mrow>
        <m:mrow>
            <m:mn>3</m:mn>
            <m:mo>&InvisibleTimes;</m:mo>
            <m:msup>
                <m:mi>Z</m:mi>
                <m:mn>2</m:mn>
            </m:msup>
            <m:mo>-</m:mo>
            <m:mrow>
                <m:mn>6</m:mn>
                <m:mo>&InvisibleTimes;</m:mo>
                <m:mi>Z</m:mi>
            </m:mrow>
            <m:mo>+</m:mo>
            <m:mn>12</m:mn>
```

```
        </m:mrow>
        <m:mo>=</m:mo>
        <m:mn>0</m:mn>
    </m:mrow>
</math>
```

You can see the results of this document in the Amaya browser shown in Figure 1.13.

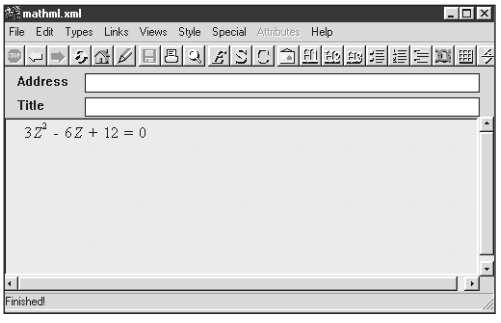


**Figure 1.13** Displaying MathML in the Amaya browser.

## XML at Work: Channel Definition Format

With the growth of the Web, people are always trying to come up with new ways to use it, and Microsoft is hard at work on this, too. One such innovation from Microsoft is the idea of Web site *channels*, which send documents to the user rather than waiting for the user to come and get them. Channels introduce the idea of Webcasting, or "push" (although it's not true server push in the HTML sense).

CDF documents are actually XML files, and you can learn about them at `msdn.microsoft.com/workshop/delivery/cdf/reference/CDF.asp`. I'll also discuss them later in this book.

Here's the way CDF works: you add a link to a .cdf file to a Web page, and then you give the link text, something like "Subscribe to this channel!" If the user is using Internet Explorer and clicks the hyperlink to navigate to the CDF file, Internet Explorer adds the site to the user's Favorites folder and subscribes to the channel, checking back periodically for updates.

Here's an example: This document, w3c.cdf, lets the user subscribe to the W3C XML and XSL pages:

```
<?xml version="1.0"?>
<CHANNEL HREF="http://www.w3.org/">
  <TITLE>World Wide Web Consortium</TITLE>
  <ABSTRACT>
    Leading the Web to its Full Potential
  </ABSTRACT>

  <ITEM HREF="http://www.w3.org/XML/">
    <TITLE>Extensible Markup Language (XML)
</TITLE>
    <ABSTRACT>
      The Extensible Markup Language (XML) is the universal
      format for structured documents and data on the Web.
    </ABSTRACT>
  </ITEM>

  <ITEM HREF="http://www.w3.org/Style/XSL/">
    <TITLE>Extensible Stylesheet Language (XSL)
</TITLE>
    <ABSTRACT>
      Extensible Stylesheet Language (XSL) is a
      language for expressing style sheets.
      </ABSTRACT>
  </ITEM>
</CHANNEL>
```

You can see the results in Figure 1.14. When the user opens this .cdf file, a W3C channel is added to the Internet Explorer's Favorites folder. (You can see the current channels in Internet Explorer by clicking the Favorites button in the standard toolbar and then double-clicking the Channels folder in the Favorites frame that opens at left.)

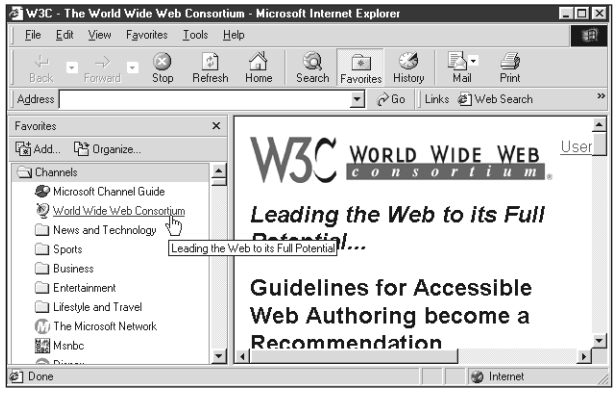


**Figure 1.14** Subscribing to a new channel using CDF.

## XML at Work: Synchronized Multimedia Integration Language

Synchronized Multimedia Integration Language (SMIL, pronounced "smile") has been around for quite some time. It's a W3C standard that you can find more about at `www.w3.org/AudioVideo/`.

SMIL attempts to fix a problem with modern "multimedia" browsers. Usually, such browsers can handle only one aspect of multimedia—video, or audio, or images—at a time, but never more than that. SMIL lets you create television-like fast cuts and true multimedia presentations.

The idea is that SMIL lets you specify what multimedia files are played when; SMIL itself does not describe or encapsulate any multimedia itself.

Microsoft, Macromedia, and Compaq have a semicompeting specification, HTML+TIME, which I'll take a look at next. As a result, Microsoft hasn't implemented much of SMIL in Internet Explorer yet, although there is limited support for SMIL in the preview version of Internet Explorer 5.5. You can find a SMIL applet written in Java at `www.empirenet.com/~joseram`, as well as some stunning examples of symphonies coordinated with images.

SMIL has become a core part of the RealNetworks streaming software (`http://service.real.com/help/library/guides/production/realpgd.htm`) and Apple Quicktime (`www.apple.com/quicktime/authoring/qtsmil.html`). In addition, the SMIL Boston project (`www.w3.org/TR/smil-boston/`) adds transition effects and event handling to SMIL 1.0. More implementations are also listed at `www.w3.org/AudioVideo`.

Here's an example SMIL document that creates a multimedia sequence, first playing mozart1.wav and amadeus1.mov; then displaying mozart1.htm; next playing mozart2.wav and amadeus2.mov; and finally displaying mozart2.htm:

```
<?xml version="1.0"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
  "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
<smil>
    <body>
        <seq id="mozart">
            <audio src="mozart1.wav"/>
            <video src="amadeus1.mov"/>
            <text src="mozart1.htm"/>
            <audio src="mozart2.wav"/>
            <video src="amadeus2.mov"/>
            <text src="mozart2.htm"/>
        </seq>
    </body>
</smil>
```

## XML at Work: HTML+TIME

Microsoft, Macromedia, and Compaq have a multimedia alternative to SMIL called Timed Interactive Multimedia Extension (referred to as HTML+TIME), which is an XML application. Although SMIL documents let you manipulate other files, HTML+TIME lets you handle both HTML and multimedia presentations in the same page.

HTML+TIME is not nearly as powerful as SMIL, but Microsoft has shown relatively little interest in SMIL. You can find out about HTML+TIME at `msdn.microsoft.com/workshop/Author/behaviors/time.asp`. HTML+TIME is implemented in Internet Explorer as a *behavior*, which is a new construct in Internet Explorer 5 that lets you separate code from data. You can find more information about Internet Explorer behaviors at `msdn.microsoft.com/workshop/c-frame.htm#/workshop/author/default.asp`.

Here's an example HTML+TIME document that displays the words `Hello`, `there`, `from`, and `HTML+TIME`, spacing the words' appearance apart by two seconds and then repeating:

```
<HTML>
    <HEAD>
        <TITLE>
            Using HTML+TIME
        </TITLE>
        <STYLE>
            .time {behavior: url(#default#time);}
        </STYLE>
    </HEAD>
```

```
    <BODY>
        <DIV CLASS="time" t:REPEAT="5" t:DUR="10" t:TIMELINE="par">
            <DIV CLASS="time" t:BEGIN="0" t:DUR="10">Hello</DIV>
            <DIV CLASS="time" t:BEGIN="2" t:DUR="10">there</DIV>
            <DIV CLASS="time" t:BEGIN="4" t:DUR="10">from</DIV>
            <DIV CLASS="time" t:BEGIN="6" t:DUR="10">HTML+TIME.</DIV>
        </DIV>
    </BODY>
</HTML>
```

You can see the results of this HTML+TIME document in Figure 1.15.


**Figure 1.15** An HTML+TIME document at work.

HTML+TIME actually builds on SMIL to a great extent; the example from the previous topic on SMIL would look this way in HTML+TIME:

```
<t:seq id="mozart">
    <t:audio src="mozart1.wav"/>
    <t:video src="amadeus1.mov"/>
    <t:textstream src="mozart1.htm"/>
    <t:audio src="mozart2.wav"/>
    <t:video src="amadeus2.mov"/>
    <t:textstream src="mozart2.htm"/>
</seq>
```

## XML at Work: XHTML

One of the biggest XML applications around today is XHTML, the translation of HTML 4.0 into XML by W3C. I'll dig into XHTML in some depth in this book.

W3C introduced XHTML to bridge the gap between HTML and XML, and to introduce more people to XML. XHTML is simply an application that mimics HTML 4.0 in such a way that you can display the results—true XML documents—in current Web browsers. XHTML is an exciting development in the XML world, and we'll be spending some time with it later in this book, in Chapter 16, "Essential XHTML," and Chapter 17, "XHTML at Work."

Here are some XHTML resources online:

- `www.w3.org/MarkUp/Activity.html`. The W3C Hypertext Markup activity page, which has an overview of XHTML.
- `www.w3.org/TR/xhtml1/`. The XHTML 1.0 specification (in more common use than XHTML 1.1 today).
- `www.w3.org/TR/xhtml11/`. The XHTML 1.1 working draft of the XHTML 1.1 module-based specification.

XHTML 1.0 comes in three different versions: transitional, frameset, and strict. The transitional version is the most popular because it supports HTML more or less as it's used today. The frameset version supports XHTML documents that display frames; this version is different from the transitional version because documents in the transitional version are based on the `<body>` element, whereas documents that use frames are based on the `<frameset>` element. The strict version omits all the HTML elements deprecated in HTML 4.0 (of which there were quite a few).

XHTML 1.1 is a form of the XHTML 1.0 strict version made a little more strict by omitting support for some elements and adding support for a few more (such as `<ruby>` for annotated text). You can find a list of the differences between XHTML 1.0 and XHTML 1.1 at `www.w3.org/TR/xhtml11/changes.html#a_changes`. XHTML 1.1 is quite strict, and I think it'll be quite a while before it's in widespread use compared to the XHTML 1.0 transitional version.

Here's an example XHTML document using the XHTML 1.0 transitional DTD. You can display this document in any standard HTML browser, as long as you give the document file the extension .html (note that tag names are all in lowercase text in XHTML):

```
<?xml version="1.0"?>
<!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>
            Web page number one!
        </title>
    </head>

    <body>
        <h1>
            Welcome to XHTML!
        </h1>
        <center>
        This is simple text that appears in this page.
            <p>
                Here's a new paragraph!
            </p>
        </center>
    </body>
</html>
```

You can see the results of this XHTML in Figure 1.16. Writing XHTML is a lot like HTML, except that you must adhere to XML syntax (such as making sure that every element has a closing tag).

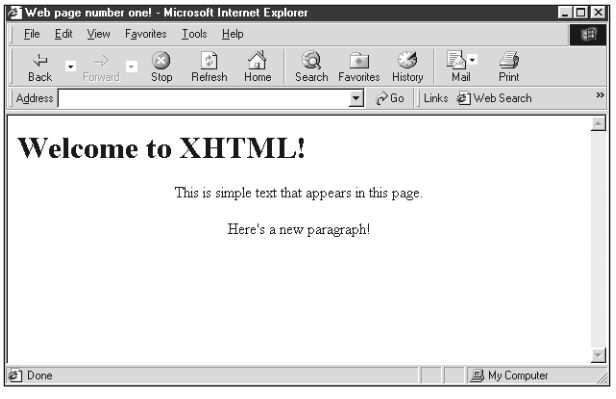


**Figure 1.16** Displaying XHTML.

## XML at Work: Open Software Description

Open Software Description (OSD) was developed by Marimba and Microsoft; you can find more about this XML application at `www.w3.org/TR/NOTE-OSD.html`. OSD enables you to specify how and when software is updated via the Internet. In fact, you can use OSD with CDF to make periodic software updates to the user's machine.

Not everyone thinks OSD is a great idea: After all, many users want control over when their software is updated. New versions may have incompatibilities with old versions, for example.

Here's an example .osd file that handles updates for a word processor named SuperDuperTextPro from SuperDuperSoft:

```
<?xml version="1.0"?>
<CHANNEL HREF="http://www.superdupersoft.com/updates.html">
    <TITLE>
        SuperDuperTextPro Updates
    </TITLE>
    <USAGE VALUE="SoftwareUpdate"/>
    <SOFTPKG
        HREF="http://updates.superdupersoft.com/updates.html"
        NAME="{34567A7E-8BE7-99C0-8746-0034829873A3}"
        VERSION="2,4,6">
        <TITLE>
            SuperDuperTextPro
        </TITLE>
        <ABSTRACT>
            SuperDuperTextPro version 206 with sideburns!!!
        </ABSTRACT>
        <IMPLEMENTATION>
             <CODEBASE HREF=
                 "http://www.superdupersoft.com/new.exe"/>
        </IMPLEMENTATION>
    </SOFTPKG>
</CHANNEL>
```

## XML at Work: Scalable Vector Graphics

Scalable Vector Graphics (SVG) is another W3C-based XML application that is a good idea but that has found only limited implementation so far (notably, in such programs as CorelDraw and various Adobe products such as Adobe Illustrator). Using SVG, you can draw two-dimensional graphics

using markup. You can find the SVG specification at www.w3.org/TR/SVG/ and an overview at www.w3.org/Graphics/SVG/Overview.htm8.

Note that because SVG describes graphics, not text, it's harder for current browsers to implement; no browsers currently have full SVG implementations. Other graphics standards are proposed, such as the Precision Graphics Markup Language (PGML), proposed to W3C (www.w3.org/TR/1998/NOTE-PGML) by IBM, Adobe, Netscape, and Sun.

Here's an example PGML document that draws a blue box:

```
<?xml version="1.0"?>
<!DOCTYPE pgml SYSTEM "/DTDs/pgml.dtd">
<pgml>
  <group fillcolor="blue">
    <path>
      <moveto x="0" y="0"/>
      <lineto x="0" y="1000"/>
      <lineto x="1000" y="1000"/>
      <lineto x="1000" y="0"/>
      <closepath/>
    </path>
  </group>
</pgml>
```

## XML at Work: Vector Markup Language

Vector Markup Language (VML) is an alternative to SVG that is implemented in Microsoft Internet Explorer. You can find out more about VML at www.w3.org/TR/NOTE-VML. Using VML, you can draw many vector-based graphics figures. Here's an example, vml.html, that draws a yellow oval, a blue box, and a red squiggle:

```
<HTML xmlns:v="urn:schemas-microsoft-com:vml">

    <HEAD>
        <TITLE>
            Using Vector Markup Language
        </TITLE>

        <STYLE>
        v\:* {behavior: url(#default#VML);}
        </STYLE>
    </HEAD>
```

*continues* ▸

```
    <BODY>
        <CENTER>
            <H1>
                Using Vector Markup Language
            </H1>
        </CENTER>
        <P>
        <v:oval STYLE='width:100pt; height:75pt'
            fillcolor="yellow"> </v:oval>
        <P>
        <v:rect STYLE='width:100pt; height:75pt' fillcolor="blue"
            strokecolor="red" STROKEWEIGHT="2pt"/>
        <P>
        <v:polyline
            POINTS="20pt,55pt,100pt,-10pt,180pt,65pt,260pt,25pt"
            strokecolor="red" STROKEWEIGHT="2pt"/>
    </BODY>
</HTML>
```

You can see the results of this VML in Figure 1.17.

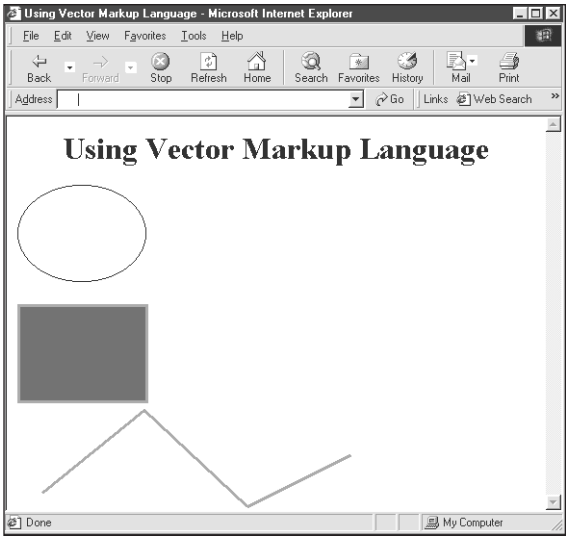


**Figure 1.17** Vector Markup Language at work.

## XML at Work: XML-Based User Interface Language

XML-based User Interface Language (XUL, pronounced "zuul"), which comes from Netscape and Mozilla, enables you to describe what user-interface elements you want those browsers to display. (The only Netscape Navigator that currently supports XUL is Netscape Navigator 6 preview version,) Here are some online references for XUL:

- `www.mozilla.org/projects/intl/xul-styleguide.html`. A XUL style guide.
- `www.mozilla.org/xpfe/xptoolkit/xulintro.html`. An introduction to XUL.

Here's a sample XUL document, scroll.xul, that adds scrollbars around the browser's document display area. (For formatting reasons, I have to break up the expression `"http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"` to two lines in this book; make sure you rejoin that text into a quoted text string on one line before giving this a try).

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

<window align="horizontal"
    xmlns=
    "http://www.mozilla.org/keymaster/gatekeeper/
        there.is.only.xul">
    <scrollbar align="vertical"/>
    <box align="vertical" flex="100%">
        <scrollbar align="horizontal"/>
        <spring flex="100%" style="background-color: white"/>
        <scrollbar align="horizontal"/>
    </box>
    <scrollbar align="vertical"/>
</window>
```

You can see the results of this XUL document in Figure 1.18.

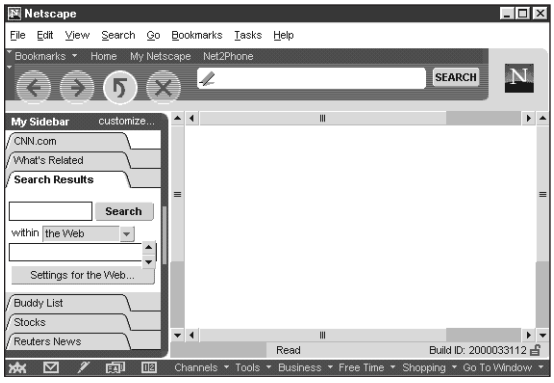


**Figure 1.18** Using XUL to create scrollbars.

## XML at Work: Extensible Business Reporting Language

Extensible Business Reporting Language (XBRL, formerly named XFRML), is an open specification that uses XML to describe financial statements. You can find more on XBRL at `www.xfrml.org/Overview.htm`. Using XBRL, you can codify business financial statements in a way that makes it easy to search them en masse and review them quickly, extracting the information you want.

Here's a sample XBRL document that gives you an idea of what this application looks like at work:

```
<?xml version="1.0" encoding="utf-8" ?>
    <group xmlns="http://www.xbrl.org/us/aicpa-us-gaap"
        xmlns:gpsi="http://www.xbrl.org/TaxonomyCustom.xsd"
        id="543-AB" entity="NASDAQ:GPSI" period="1999-05-31"
        schemaLocation="http://www.xbrl.org/TaxonomyCustom.xsd"
        scaleFactor="6" precision="9" type="USGAAP:Financial"
        unit="ISO4217:USD" decimalPattern="" formatName="">
        <item id="IS-025"
            type="operatingExpenses.researchExpense"
            period="P1Y/1999-05-31">20427</item>
        <item id="IS-026"
            type="operatingExpenses.researchExpense"
            period="P1Y/1998-05-31">12586</item>
    </group>
    <group type="gpsi:detail.quarterly" period="1998-05-31">
```

```
    <item period="1997-06-01/1998-07-31">0.12</item>
    <item period="1997-09-01/1997-11-30">0.16</item>
    <item period="1997-12-01/1998-02-28">0.17</item>
    <item period="1998-03-01/1998-05-31">-0.12</item>
    <item period="1998-06-01/1998-05-31">0.33</item>
</group>
<group type="gpsi:detail.quarterly" period="1999-05-31">
    <item period="1998-06-01/1998-08-31">0.15</item>
    <item period="1998-09-01/1998-11-30">0.20</item>
    <item period="1998-12-01/1999-02-28">0.23</item>
    <item period="1999-03-01/1999-05-31">0.28</item>
    <item period="1998-06-01/1999-05-31">0.86</item>
</group>
<group type="gpsi:detail.quarterly" period="1998-05-31">
    <item period="1997-06-01/1998-07-31">0.11</item>
    <item period="1997-09-01/1997-11-30">0.15</item>
    <item period="1997-12-01/1998-02-28">0.17</item>
    <item period="1998-03-01/1998-05-31">-0.12</item>
    <item period="1998-06-01/1998-05-31">0.32</item>
</group>
```

## XML at Work: Resource Description Framework

Resource Description Framework (RDF) is an XML application that specializes in meta-data—that is, data about other data. You use RDF to specify information about other resources, such as Web pages, movies, automobiles, or practically anything. You can find more information about RDF at `www.w3.org/RDF/`, and I'll be discussing it later in the book as well, in Chapter 18, "Resource Description Framework and Channel Definition Format."

Using RDF, you create vocabularies that describe resources. For example, the Dublin Core is an RDF vocabulary that handles meta-data for Web pages; you can find more information about it at `http://purl.org/DC/`. Using the Dublin Core, you can specify a great deal of information about Web pages that is designed ultimately to replace the unsystematic use of `<META>` tags in today's pages. When systemized, that information will be much more tractable to Web search engines.

Here's an example RDF page using the Dublin Core that gives information about a Web page:

```
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:DC="http://purl.org/DC/">
    <RDF:Description about="http://www.starpowder.com/xml">
        <DC:Format>HTML</DC:Format>
        <DC:Language>en</DC:Language>
        <DC:Date>2002-02-02</DC:date>
        <DC:Type>tutorial</DC:Type>
        <DC:Title>Welcome to XML!</DC:Title>
    </RDF:Description>
</RDF:RDF>
```

Note that many more XML applications exist than can be covered in one chapter—and plenty of them work behind the scenes. For example, Microsoft Office 2000 can handle HTML as well as other types of documents, but because HTML doesn't allow it to store everything it needs in a document, Office 2000 also includes some XML behind the scenes (in fact, Office 2000's vector graphics are done using VML). Even relatively early versions of Netscape Navigator allowed you to look for sites much like the current one you're viewing; to do that, it connected to a CGI program that uses XML internally. As you can see, XML is all around, everywhere you look on the Internet.

That's it for our overview chapter. We've received a solid foundation in XML here. The next step is to start getting all the actual ground rules for creating XML documents under our belts. I'll cover that in Chapter 2.