

An Introduction to XML Applications

In this chapter we'll be looking at some examples of XML applications, markup languages used to further refine XML, and behind-the-scene uses of XML. It is inspiring to look at some of the uses to which XML has already been put, even in this early stage of its development. This chapter will give you some idea of the wide applicability of XML. Many more XML applications are being created or ported from other formats as I write this.



Part V covers some of the XML applications discussed in this chapter in more detail.

What Is an XML Application?

XML is a meta-markup language for designing domain-specific markup languages. Each XML-based markup language is called an XML application. This is not an application that uses XML like the Mozilla Web browser, the Gnumeric spreadsheet, or the XML Pro editor, but rather an application of XML to a specific domain such as Chemical Markup Language (CML) for chemistry or GedML for genealogy.

Each XML application has its own syntax and vocabulary. This syntax and vocabulary adheres to the fundamental rules of XML. This is much like human languages, which each have their own vocabulary and grammar, while at the same time adhering to certain fundamental rules imposed by human anatomy and the structure of the brain.



In This Chapter

What is an XML application?

XML for XML

Behind-the-scene uses of XML



XML is an extremely flexible format for text-based data. The reason XML was chosen as the foundation for the wildly different applications discussed in this chapter (aside from the hype factor) is that XML provides a sensible, well-documented format that's easy to read and write. By using this format for its data, a program can offload a great quantity of detailed processing to a few standard free tools and libraries. Furthermore, it's easy for such a program to layer additional levels of syntax and semantics on top of the basic structure XML provides.

Chemical Markup Language

Peter Murray-Rust's Chemical Markup Language (CML) may have been the first XML application. CML was originally developed as an SGML application, and gradually transitioned to XML as the XML standard developed. In its most simplistic form, CML is "HTML plus molecules", but it has applications far beyond the limited confines of the Web.

Molecular documents often contain thousands of different, very detailed objects. For example, a single medium-sized organic molecule may contain hundreds of atoms, each with several bonds. CML seeks to organize these complex chemical objects in a straightforward manner that can be understood, displayed, and searched by a computer. CML can be used for molecular structures and sequences, spectrographic analysis, crystallography, publishing, chemical databases, and more. Its vocabulary includes molecules, atoms, bonds, crystals, formulas, sequences, symmetries, reactions, and other chemistry terms. For instance Listing 2-1 is a basic CML document for water (H₂O):

Listing 2-1: The water molecule H₂O

```
<?xml version="1.0"?>
<CML>
  <MOL TITLE="Water">
    <ATOMS>
      <ARRAY BUILTIN="ELSYM">H O H</ARRAY>
    </ATOMS>
    <BONDS>
      <ARRAY BUILTIN="ATID1">1 2</ARRAY>
      <ARRAY BUILTIN="ATID2">2 3</ARRAY>
      <ARRAY BUILTIN="ORDER">1 1</ARRAY>
    </BONDS>
  </MOL>
</CML>
```

The biggest improvement CML offers over traditional approaches to managing chemical data is ease of searching. CML also enables complex molecular data to be sent over the Web. Because the underlying XML is platform-independent, the problem of platform-dependency that plagues the binary formats used by

traditional chemical software and documents like the Protein Data Bank (PDB) format or MDL Molfiles, is avoided.

Murray-Rust also created JUMBO, the first general-purpose XML browser. Figure 2-1 shows JUMBO displaying a CML file. Jumbo works by assigning each XML element to a Java class that knows how to render that element. To allow Jumbo to support new elements, you simply write Java classes for those elements. Jumbo is distributed with classes for displaying the basic set of CML elements including molecules, atoms, and bonds, and is available at <http://www.xml-cml.org/>.

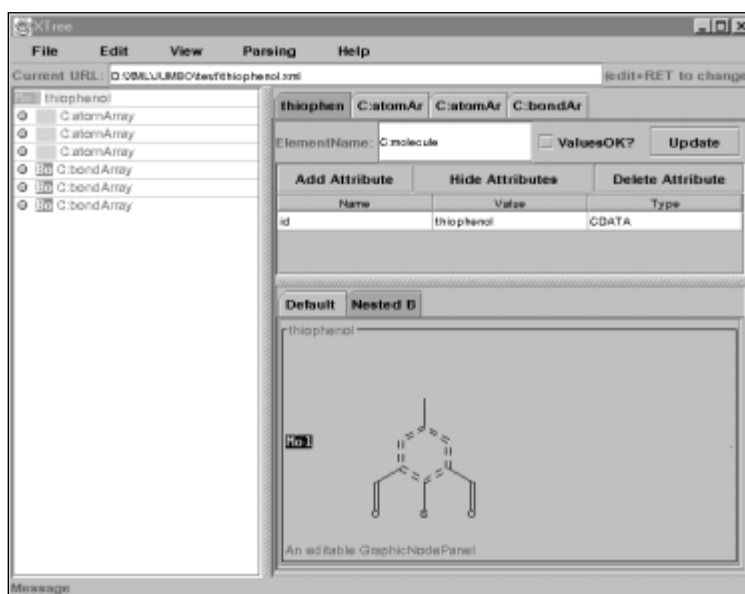


Figure 2-1: The JUMBO browser displaying a CML file

Mathematical Markup Language

Legend claims that Tim Berners-Lee invented the World Wide Web and HTML at CERN so that high-energy physicists could exchange papers and preprints. Personally I've never believed that. I grew up in physics; and while I've wandered back and forth between physics, applied math, astronomy, and computer science over the years, one thing the papers in all of these disciplines had in common was lots and lots of equations. Until now, nine years after the Web was invented, there hasn't been any good way to include equations in Web pages.

There have been a few hacks — Java applets that parse a custom syntax, converters that turn LaTeX equations into GIF images, custom browsers that read TeX files — but none of these have produced high quality results, and none of them have caught on with Web authors, even in scientific fields. Finally, XML is starting to change this.

The Mathematical Markup Language (MathML) is an XML application for mathematical equations. MathML is sufficiently expressive to handle pretty much all forms of math — from grammar-school arithmetic through calculus and differential equations. It can handle many considerably more advanced topics as well, though there are definite gaps in some of the more advanced and obscure notations used by certain sub-fields of mathematics. While there are limits to MathML on the high end of pure mathematics and theoretical physics, it is eloquent enough to handle almost all educational, scientific, engineering, business, economics, and statistics needs. And MathML is likely to be expanded in the future, so even the purest of the pure mathematicians and the most theoretical of the theoretical physicists will be able to publish and do research on the Web. MathML completes the development of the Web into a serious tool for scientific research and communication (despite its long digression to make it suitable as a new medium for advertising brochures).

Netscape Navigator and Internet Explorer do not yet support MathML. Nonetheless, it is the fervent hope of many mathematicians that they soon will. The W3C has integrated some MathML support into their test-bed browser, Amaya. Figure 2-2 shows Amaya displaying the covariant form of Maxwell's equations written in MathML.



Amaya is on the CD-ROM in the browsers/amaya directory.

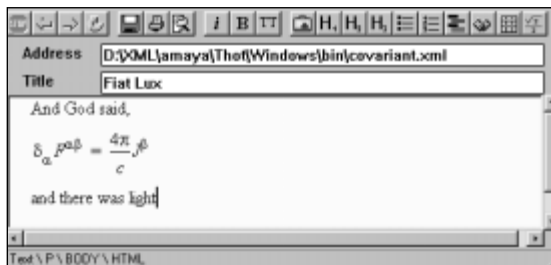


Figure 2-2: The Amaya browser displaying the covariant form of Maxwell's equations written in MathML

The XML file the Amaya browser is displaying is given in Listing 2-2:

Listing 2-2: Maxwell's Equations in MathML

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/TR/REC-html40"
      xmlns:m="http://www.w3.org/TR/REC-MathML/"
>
```

```

<head>
<title>Fiat Lux</title>
<meta name="GENERATOR" content="amaya V1.3b" />
</head>
<body>

<P>
And God said,
</P>

<math>
  <m:mrow>
    <m:msub>
      <m:mi>&delta;</m:mi>
      <m:mi>&alpha;</m:mi>
    </m:msub>
    <m:msup>
      <m:mi>F</m:mi>
      <m:mi>&alpha;&beta;</m:mi>
    </m:msup>
    <m:mi></m:mi>
    <m:mo>=</m:mo>
    <m:mi></m:mi>
    <m:mfrac>
      <m:mrow>
        <m:mn>4</m:mn>
        <m:mi>&pi;</m:mi>
      </m:mrow>
      <m:mi>c</m:mi>
    </m:mfrac>
    <m:mi></m:mi>
    <m:msup>
      <m:mi>J</m:mi>
      <m:mrow>
        <m:mi>&beta;</m:mi>
        <m:mo></m:mo>
      </m:mrow>
    </m:msup>
  </m:mrow>
</math>

<P>
and there was light
</P>
</body>
</html>

```

Listing 2-2 is an example of a mixed HTML/XML page. The headers and paragraphs of text (“Fiat Lux”, “Maxwell’s Equations”, “And God said”, “and there was light”) is given in classic HTML. The actual equations are written in MathML, an application of XML.

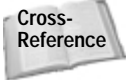
In general, such mixed pages require special support from the browser, as is the case here, or perhaps plug-ins, ActiveX controls, or JavaScript programs that parse and display the embedded XML data. Ultimately, of course, you want a browser like Mozilla 5.0 or Internet Explorer 5.0 that can parse and display pure XML files without an HTML intermediary.

Channel Definition Format

Microsoft's Channel Definition Format (CDF) is an XML application for defining channels. Web sites use channels to upload information to readers who subscribe to the site rather than waiting for them to come and get it. This is alternately called *Webcasting* or *push*. CDF was first introduced in Internet Explorer 4.0.

A CDF document is an XML file, separate from, but linked to an HTML document on the site being pushed. The channel defined in the CDF document determines which pages are sent to the readers, how the pages are transported, and how often the pages are sent. Pages can either be pushed by sending notifications, or even whole Web sites, to subscribers; or pulled down by the readers at their convenience.

You can add CDF to your site without changing any of the existing content. You simply add an invisible link to a CDF file on your home page. Then when a reader visits the page, the browser displays a dialog box asking them if they want to subscribe to the channel. If the reader chooses to subscribe, the browser downloads a copy of the CDF document describing the channel. The browser then combines the parameters specified in the CDF document with the user's own preferences to determine when to check back with the server for new content. This isn't true push, because the client has to initiate the connection, but it still happens without an explicit request by the reader. Figure 2-3 shows the IDG Active Channel in Internet Explorer 4.0.



Cross-Reference

CDF is covered in more detail in Chapter 21, *Pushing Web Sites with CDF*.



On the CD-ROM

Internet Explorer 4.0 is on the CD-ROM in the `browsers/ie4` directory.

Classic Literature

Jon Bosak has translated the complete plays of Shakespeare into XML. The complete text of the plays is included, and XML markup is used to distinguish between titles, subtitles, stage directions, speeches, lines, speakers, and more.



On the CD-ROM

The complete set of plays is on the CD-ROM in the `examples/shakespeare` directory.



Figure 2-3: The IDG Active Channel in Internet Explorer 4.0

You may ask yourself what this offers over a book, or even a plain text file. To a human reader, the answer is not much. But to a computer doing textual analysis, it offers the opportunity to easily distinguish between the different elements into which the plays have been divided. For instance, it makes it quite simple for the computer to go through the text and extract all of Romeo's lines.

Furthermore, by altering the style sheet with which the document is formatted, an actor could easily print a version of the document in which all their lines were formatted in bold face, and the lines immediately before and after theirs were italicized. Anything else you might imagine that requires separating a play into the lines uttered by different speakers is much more easily accomplished with the XML-formatted versions than with the raw text.

Bosak has also marked up English translations of the old and new testaments, the Koran, and the Book of Mormon in XML. The markup in these is a little different. For instance, it doesn't distinguish between speakers. Thus you couldn't use these particular XML documents to create a red-letter Bible, for example, although a different set of tags might allow you to do that. (A red-letter Bible prints words spoken by Jesus in red.) And because these files are in English rather than the original languages, they are not as useful for scholarly textual analysis. Still, time and resources permitting, those are exactly the sorts of things XML would allow you to do if you wanted to. You'd simply need to invent a different vocabulary and syntax than the one Bosak used that would still describe the same data.



The XML-ized Bible, Koran, and Book of Mormon are all on the CD-ROM in the examples/religion directory.

Synchronized Multimedia Integration Language

The Synchronized Multimedia Integration Language (SMIL, pronounced “smile”) is a W3C recommended XML application for writing “TV-like” multimedia presentations for the Web. SMIL documents don’t describe the actual multimedia content (that is the video and sound that are played) but rather when and where they are played.

For instance, a typical SMIL document for a film festival might say that the browser should simultaneously play the sound file `beethoven9.mid`, show the video file `corange.mov`, and display the HTML file `clockwork.htm`. Then, when it’s done, it should play the video file `2001.mov`, the audio file `zarathustra.mid`, and display the HTML file `aclarke.htm`. This eliminates the need to embed low bandwidth data like text in high bandwidth data like video just to combine them. Listing 2-3 is a simple SMIL file that does exactly this.

Listing 2-3: A SMIL film festival

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
    "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
<smil>
  <body>
    <seq id="Kubrick">
      <audio src="beethoven9.mid"/>
      <video src="corange.mov"/>
      <text src="clockwork.htm"/>
      <audio src="zarathustra.mid"/>
      <video src="2001.mov"/>
      <text src="aclarke.htm"/>
    </seq>
  </body>
</smil>
```

Furthermore, as well as specifying the time sequencing of data, a SMIL document can position individual graphics elements on the display and attach links to media objects. For instance, at the same time the movie and sound are playing, the text of the respective novels could be subtitled the presentation.

HTML+TIME

SMIL operates independently of the Web page. The streaming media pushed through SMIL has its own pane in the browser frame, but it doesn't really have any interaction with the content in the HTML on the rest of the page. For instance, SMIL only lets you time SMIL elements like audio, video, and text. It doesn't let you add timing information to basic HTML tags like <P>, , or . And SMIL duplicates some aspects of HTML, such as how elements are positioned on the page.

Microsoft, along with Macromedia and Compaq, has proposed a semi-competing XML application called Timed Interactive Multimedia Extensions for HTML (or HTML+TIME for short). HTML+TIME builds on SMIL to support timing for traditional HTML elements and features much closer integration with the HTML on the Web page. For example, HTML+TIME lets you write a countdown Web page like Listing 2-4 that adds to the page as time progresses.

Listing 2-4: A countdown Web page using HTML+TIME

```
<html>
  <head><title>Countdown</title></head>
  <body>
    <p t:begin="0" t:dur="1">10</p>
    <p t:begin="1" t:dur="1">9</p>
    <p t:begin="2" t:dur="1">8</p>
    <p t:begin="3" t:dur="1">7</p>
    <p t:begin="4" t:dur="1">6</p>
    <p t:begin="5" t:dur="1">5</p>
    <p t:begin="6" t:dur="1">4</p>
    <p t:begin="7" t:dur="1">3</p>
    <p t:begin="8" t:dur="1">2</p>
    <p t:begin="9" t:dur="1">1</p>
    <p t:begin="10" t:dur="1">Blast Off!</p>
  </body>
</html>
```

This is useful for slide shows, timed quizzes, and the like. In HTML+TIME, the film festival example of Listing 2-3 looks like the following:

```
<t:seq id="Kubrick">
  <t:audio src="beethoven9.mid"/>
  <t:video src="corange.mov"/>
  <t:textstream src="clockwork.htm"/>
  <t:audio src="zarathustra.mid"/>
  <t:video src="2001.mov"/>
  <t:textstream src="aclarke.htm"/>
</seq>
```

It's close to, though not quite exactly the same as, the SMIL version. The major difference is that the SMIL version is intended to be stored in separate files and rendered by special players like RealPlayer, whereas the HTML+TIME version is supposed to be included in the Web page and rendered by the browser. Another key difference is that there are several products that can play SMIL files now, including RealPlayer G2, whereas HTML+TIME-enabled Web browsers do not exist at the moment. However, it's likely that future versions of Internet Explorer will include HTML+TIME support.

There are some nice features and some good ideas in HTML+TIME. However, the W3C had already given its blessing to SMIL several months before Microsoft proposed HTML+TIME, and SMIL has a lot more momentum and support in the third-party, content creator community. Thus it seems we're in for yet another knockdown, drag-out, Microsoft-vs.-everybody-else-in-the-known-universe battle which will only leave third party developers bruised and confused. One can only hope that the W3C has the will and energy to referee this fight fairly. Web development really would be a lot simpler if Microsoft didn't pick up its toys and go home every time they don't get their way.

Open Software Description

The Open Software Description (OSD) format is an XML application co-developed by Marimba and Microsoft for updating software automatically. OSD defines XML tags that describe software components. The description of a component includes the version of the component, its underlying structure, and its relationships to and dependencies on other components. This provides enough information for OSD to decide whether a user needs a particular update or not. If they do need the update, it can be automatically pushed to users, rather than requiring them to manually download and install it. Listing 2-5 is an example of an OSD file for an update to WhizzyWriter 1000:

Listing 2-5: An OSD file for an update to WhizzyWriter 1000

```
<?XML version="1.0"?>
<CHANNEL HREF="http://updates.whizzy.com/updateChannel.html">
  <TITLE>WhizzyWriter 1000 Update Channel</TITLE>
  <USAGE VALUE="SoftwareUpdate"/>
  <SOFTPKG HREF="http://updates.whizzy.com/updateChannel.html"
    NAME="{46181F7D-1C38-22A1-3329-00415C6A4D54}"
    VERSION="5,2,3,1"
    STYLE="MSAppLogo5"
    PRECACHE="yes">
    <TITLE>WhizzyWriter 1000</TITLE>
    <ABSTRACT>
      Abstract: WhizzyWriter 1000: now with tint control!
    </ABSTRACT>
```

```
<IMPLEMENTATION>  
  <CODEBASE HREF="http://updates.whizzy.com/tinupdate.exe"/>  
</IMPLEMENTATION>  
</SOFTPKG>  
</CHANNEL>
```

Only information about the update is kept in the OSD file. The actual update files are stored in a separate CAB archive or executable and downloaded when needed. There is considerable controversy about whether or not this is actually a good thing. Many software companies, Microsoft not least among them, have a long history of releasing updates that cause more problems than they fix. Many users prefer to stay away from new software for a while until other, more adventurous souls have given it a shakedown.

Scalable Vector Graphics

Vector graphics are superior to the bitmap GIF and JPEG images currently used on the Web for many pictures including flow charts, cartoons, advertisements, and similar images. However, many traditional vector graphics formats like PDF, PostScript, and EPS were designed with ink on paper in mind rather than electrons on a screen. (This is one reason PDF on the Web is such an inferior replacement for HTML, despite PDF's much larger collection of graphics primitives.) A vector graphics format for the Web should support a lot of features that don't make sense on paper like transparency, anti-aliasing, additive color, hypertext, animation, and hooks to enable search engines and audio renderers to extract text from graphics. None of these features are needed for the ink-on-paper world of PostScript and PDF.

Several vendors have made a variety of proposals to the W3C for XML applications for vector graphics. These include:

- ♦ The Precision Graphics Markup Language (PGML) from IBM, Adobe, Netscape, and Sun.
- ♦ The Vector Markup Language (VML) from Microsoft, Macromedia, Autodesk, Hewlett-Packard, and Visio
- ♦ Schematic Graphics on the World Wide Web from the Central Laboratory of the Research Councils
- ♦ DrawML from Excrosoft AB
- ♦ Hyper Graphics Markup Language (HGML) from PRP and Orange PCSL

Each of these reflects the interests and experience of its authors. For example, not surprisingly given Adobe's participation, PGML has the flavor of PostScript but with XML element-attribute syntax rather than PostScript's reverse Polish notation. Listing 2-6 demonstrates the embedding of a pink triangle in PGML.

Listing 2-6: A pink triangle in PGML

```
<?xml version="1.0"?>
<!DOCTYPE pgml SYSTEM "pgml.dtd">
<pgml>
  <group name="PinkTriangle" fillcolor="pink">
    <path>
      <moveto x="0" y="0"/>
      <lineto x="100" y="173"/>
      <lineto x="200" y="0"/>
      <closepath/>
    </path>
  </group>
</pgml>
```

The W3C has formed a working group with representatives from the above vendors to decide on a single, unified, scalable vector graphics specification called SVG. SVG is an XML application for describing two-dimensional graphics. It defines three basic types of graphics: shapes, images, and text. A shape is defined by its outline, also known as its path, and may have various strokes or fills. An image is a bitmapped file like a GIF or a JPEG. Text is defined as a string of text in a particular font, and may be attached to a path, so it's not restricted to horizontal lines of text like the ones that appear on this page. All three kinds of graphics can be positioned on the page at a particular location, rotated, scaled, skewed, and otherwise manipulated. Since SVG is a text format, it's easy for programs to generate automatically; and it's easy for programs to manipulate. In particular you can combine it with DHTML and ECMAScript to make the pictures on a Web page animated and responsive to user action.

Since SVG describes graphics rather than text — unlike most of the other XML applications discussed in this chapter — it will probably need special display software. All of the proposed style-sheet languages assume they're displaying fundamentally text-based data, and none of them can support the heavy graphics requirements of an application like SVG. It's possible SVG support may be added to future browsers, especially since Mozilla is open source code; and it would be even easier for a plug-in to be written. However, for the time being, the prime benefit of SVG is that it is likely to be used as an exchange format between different programs like Adobe Illustrator and CorelDraw, which use different native binary formats.

SVG is not fully fleshed out at the time of this writing, and there are exactly zero implementations of it. The first working draft of SVG was released by the World Wide Web Consortium in February of 1999. Compared to other working drafts, however, it is woefully incomplete. It's really not much more than an outline of graphics elements that need to be included, without any details about how exactly those elements will be encoded in XML. I wouldn't be surprised if this draft got pushed out the door a little early to head off the adoption of competing efforts like VML.

Vector Markup Language

Microsoft has developed their own XML application for vector graphics called the Vector Markup Language (VML). VML is more finished than SVG, and is already supported by Internet Explorer 5.0 and Microsoft Office 2000. Listing 2-7 is an HTML file with embedded VML that draws the pink triangle. Figure 2-4 shows this file displayed in Internet Explorer 5.0. However, VML is not nearly as ambitious a format as SVG, and leaves out a lot of advanced features SVG includes such as clipping, masking, and compositing.

Listing 2-7: The pink triangle in VML

```
<html xmlns:vml="urn:schemas-microsoft-com:vml">
  <head>
    <title>
      A Pink Triangle, Listing 2-7 from the XML Bible
    </title>
    <object id="VMLRender"
      classid="CLSID:10072CEC-8CC1-11D1-986E-00A0C955B42E">
    </object>
    <style>
      vml\:* { behavior: url(#VMLRender) }
    </style>
  </head>
  <body>

    <div>

      <vml:polyline
        style="width: 200px; height: 200px"
        stroke="false"
        fill="true"
        fillcolor="#FFCCCC"
        points="10pt, 275pt, 310pt, 275pt, 160pt, 45pt">
      </vml:polyline>

    </div>
  </body>
</html>
```

There's really no reason for there to be two separate, mutually incompatible vector graphics standards for the Web, and Microsoft will probably grudgingly support SVG in the end. However, VML is available today, even if its use is limited to Microsoft products, whereas SVG is only an incomplete draft specification. Web artists would prefer to have a single standard, but having two is not unheard of (think Gif and JPEG). As long as the formats are documented and non-proprietary,

it's not out of the question for Web browsers to support both. At the least, the underlying XML makes it a lot easier for programmers to write converters that translate files from one format to the other.

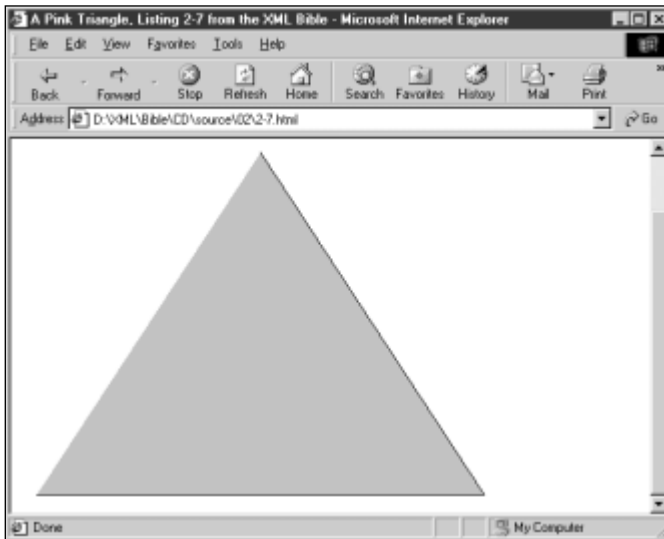


Figure 2-4: The pink triangle created with VML

Cross-
Reference

VML is discussed in more detail in Chapter 22, *The Vector Markup Language*.

MusicML

The Connection Factory has created an XML application for sheet music called MusicML. MusicML includes notes, beats, clefs, staves, rows, rhythms, rests, beams, rows, chords and more. Listing 2-8 shows the first bar from Beth Anderson's *Flute Swale* in MusicML.

Listing 2-8: The first bar of Beth Anderson's *Flute Swale*

```
<?xml version="1.0"?>
<!DOCTYPE sheetmusic SYSTEM "music.dtd">
<sheetmusic>
  <musicrow size="one">

    <entrysegment>
```

```

<entrypart cleff="bass" rythm="fourquarter"
           position="one">
  <molkruis level="plus1" name="f" notetype="sharp"/>
  <molkruis level="plus1" name="c" notetype="sharp"/>
</entrypart>
</entrysegment>

<segment>

<subsegment position="one">
  <beam size="double">
    <note beat="sixteenth" name="a" level="zero"
          dynamics="mf"/>
    <note beat="sixteenth" name="b" level="zero"></note>
    <note beat="sixteenth" name="c" level="plus1"></note>
    <note beat="sixteenth" name="a" level="zero"></note>
  </beam>
  <beam size="single">
    <note beat="eighth" name="d" level="plus1"/>
    <note beat="eighth" name="c" level="plus1"/>
  </beam>
  <note beat="quarter" name="b" level="zero"/>
  <note beat="quarter" name="a" level="zero"/>
</subsegment>

</segment>

</musicrow>
</sheetmusic>

```

The Connection Factory has also written a Java applet that can parse and display MusicML. Figure 2-5 shows the above example rendered by this applet. The applet has a few bugs (for instance the last note is missing) but overall it's a surprisingly good rendition.



Figure 2-5: The first bar of Beth Anderson's *Flute Swale* in MusicML

MusicML isn't going to replace Finale or Nightingale anytime soon. And it really seems like more of a proof of concept than a polished product. MusicML has a lot of discrepancies that will drive musicians nuts (e.g., rhythm is misspelled, treble and bass clefs are reversed, segments should really be measures, and so forth).

Nonetheless something like this is a reasonable output format for music notation programs that enables sheet music to be displayed on the Web. Furthermore, if the various notation programs all support MusicML or something like it, then it can be used as an interchange format to move data from one program to the other, something composers desperately need to be able to do now.

VoxML

Motorola's VoxML (<http://www.voxml.com/>) is an XML application for the spoken word. In particular, it's intended for those annoying voice mail and automated phone response systems ("If your hair turned green after using our product, please press one. If your hair turned purple after using our product, please press two. If you found an unidentifiable insect in the product, please press 3. Otherwise, please stay on the line until your hair grows back to its natural color.").

VoxML enables the same data that's used on a Web site to be served up via telephone. It's particularly useful for information that's created by combining small nuggets of data, such as stock prices, sports scores, weather reports, and test results. The Weather Channel and CBS MarketWatch.com are considering using VoxML to provide more information over regular voice phones.

A small VoxML file for a shampoo company's automated phone response system might look something like the code in Listing 2-9.

Listing 2-9: A VoxML file

```
<?xml version="1.0"?>
<DIALOG>
  <CLASS NAME="help_top">
    <HELP>Welcome to TIC consumer products division.
      For shampoo information, say shampoo now.
    </HELP>
  </CLASS>

  <STEP NAME="init" PARENT="help_top">
    <PROMPT>Welcome to Wonder Shampoo
    <BREAK SIZE="large"/>
    Which color did Wonder Shampoo turn your hair?
    </PROMPT>
    <INPUT TYPE="OPTIONLIST">
      <OPTION NEXT="#green">green</OPTION>
      <OPTION NEXT="#purple">purple</OPTION>
      <OPTION NEXT="#bald">bald</OPTION>
      <OPTION NEXT="#bye">exit</OPTION>
    </INPUT>
  </STEP>
```

```
<STEP NAME="green" PARENT="help_top">
  <PROMPT>
    If Wonder Shampoo turned your hair green and you wish
    to return it to its natural color, simply shampoo seven
    times with three parts soap, seven parts water, four
    parts kerosene, and two parts iguana bile.
  </PROMPT>
  <INPUT TYPE="NONE" NEXT="#bye"/>
</STEP>

<STEP NAME="purple" PARENT="help_top">
  <PROMPT>
    If Wonder Shampoo turned your hair purple and you wish
    to return it to its natural color, please walk
    widdershins around your local cemetery
    three times while chanting "Surrender Dorothy".

  </PROMPT>
  <INPUT TYPE="NONE" NEXT="#bye"/>
</STEP>

<STEP NAME="bald" PARENT="help_top">
  <PROMPT>
    If you went bald as a result of using Wonder Shampoo,
    please purchase and apply a three months supply
    of our Magic Hair Growth Formula(TM). Please do not
    consult an attorney as doing so would violate the
    license agreement printed on inside fold of the Wonder
    Shampoo box in 3 point type which you agreed to
    by opening the package.
  </PROMPT>
  <INPUT TYPE="NONE" NEXT="#bye"/>
</STEP>

<STEP NAME="bye" PARENT="help_top">
  <PROMPT>
    Thank you for visiting TIC Corp. Goodbye.
  </PROMPT>
  <INPUT TYPE="NONE" NEXT="#exit"/>
</STEP>

</DIALOG>
```

I can't show you a screen shot of this example, because it's not intended to be shown in a Web browser. Instead, you would listen to it on a telephone.

Open Financial Exchange

Software cannot be changed willy-nilly. The data that software knows how to read has inertia. The more data you have in a given program's proprietary, undocumented format, the harder it is to change programs. For example, my personal finances for the last five years are stored in Quicken. How likely is it that I will change to Microsoft Money even if Money has features I need that Quicken doesn't have? Unless Money can read and convert Quicken files with zero loss of data, the answer is "NOT BLOODY LIKELY!"

The problem can even occur within a single company or a single company's products. Microsoft Word 97 for Windows can't read documents created by some earlier versions of Word. And earlier versions of Word can't read Word 97 files at all. And Microsoft Word 98 for the Mac can't quite read everything that's in a Word 97 for Windows file, even though Word 98 for the Mac came out a year later!

As noted in Chapter 1, the Open Financial Exchange Format (OFX) is an XML application used to describe financial data of the type you're likely to store in a personal finance product like Money or Quicken. Any program that understands OFX can read OFX data. And since OFX is fully documented and non-proprietary (unlike the binary formats of Money, Quicken, and other programs) it's easy for programmers to write the code to understand OFX.

OFX not only allows Money and Quicken to exchange data with each other. It allows other programs that use the same format to exchange the data as well. For instance, if a bank wants to deliver statements to customers electronically, it only has to write one program to encode the statements in the OFX format rather than several programs to encode the statement in Quicken's format, Money's format, Managing Your Money's format, and so forth.

The more programs that use a given format, the greater the savings in development cost and effort. For example, six programs reading and writing their own and each other's proprietary format require 36 different converters. Six programs reading and writing the same OFX format require only six converters. Effort is reduced to $O(n)$ rather than $O(n^2)$. Figure 2-6 depicts six programs reading and writing their own and each other's proprietary format. Figure 2-7 depicts six programs reading and writing the same OFX format. Every arrow represents a converter that has to trade files and data between programs. In Figure 2-6, you can see the connections for six different programs reading and writing each other's proprietary binary format. In Figure 2-7, you can see the same six different programs reading and writing one open XML format. The XML-based exchange is much simpler and cleaner than the binary-format exchange.

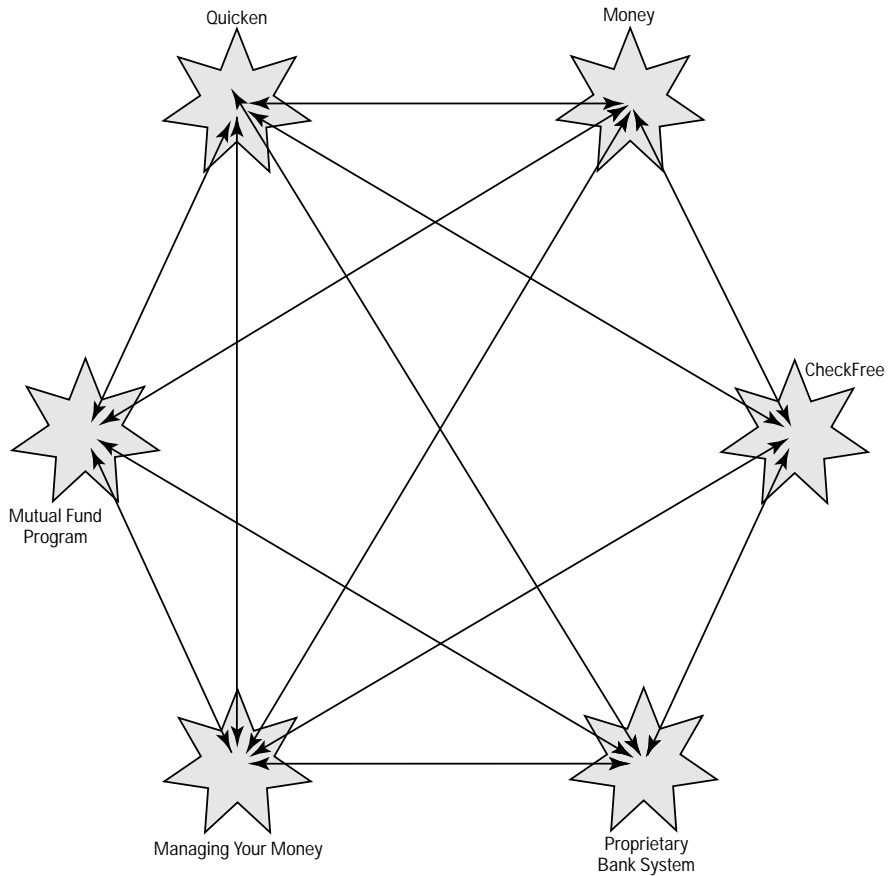


Figure 2-6: Six different programs reading and writing their own and each other's formats

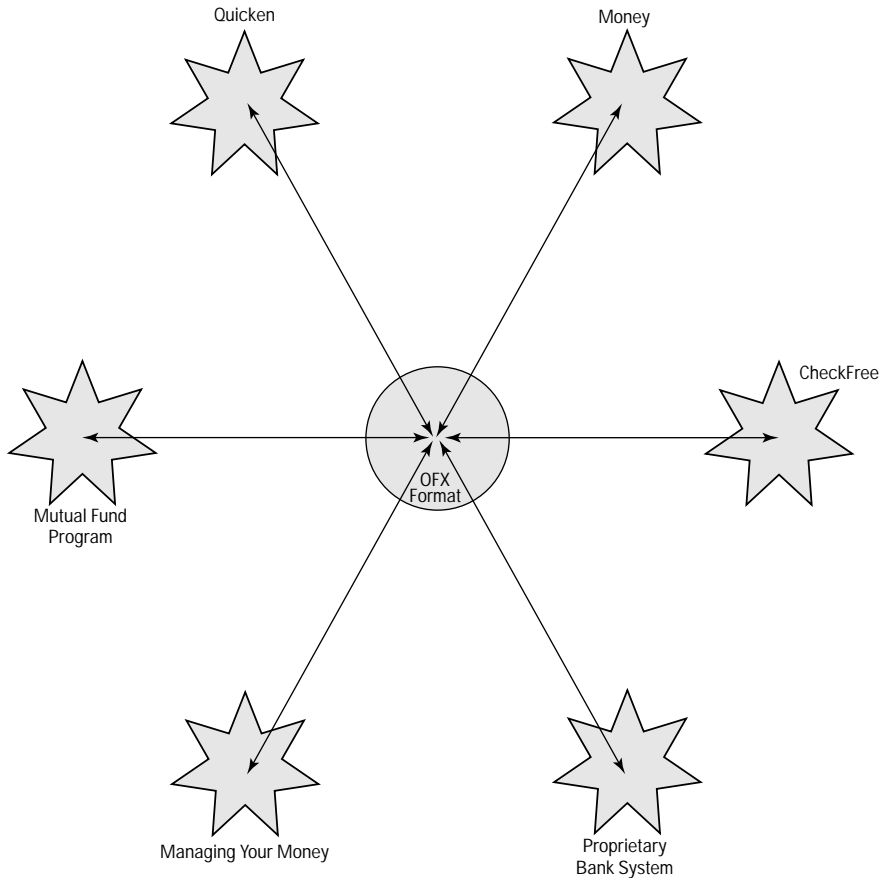


Figure 2-7: Six programs reading and writing the same OFX format

Extensible Forms Description Language

I went down to my local bookstore today and bought a copy of Armistead Maupin's novel *Sure of You*. I paid for that purchase with a credit card, and when I did so I signed a piece of paper agreeing to pay the credit card company \$14.07 when billed. Eventually they will send me a bill for that purchase, and I'll pay it. If I refuse to pay it, then the credit card company can take me to court to collect, and they can use my signature on that piece of paper to prove to the court that on October 15, 1998 I really did agree to pay them \$14.07.

The same day I also ordered Anne Rice's *The Vampire Armand* from the online bookstore amazon.com. Amazon charged me \$16.17 plus \$3.95 shipping and handling and again I paid for that purchase with a credit card. But the difference is

that Amazon never got a signature on a piece of paper from me. Eventually the credit card company will send me a bill for that purchase, and I'll pay it. But if I did refuse to pay the bill, they don't have a piece of paper with my signature on it showing that I agreed to pay \$20.12 on October 15, 1998. If I claim that I never made the purchase, the credit card company will bill the charges back to Amazon. Before Amazon or any other online or phone-order merchant is allowed to accept credit card purchases without a signature in ink on paper, they have to agree that they will take responsibility for all disputed transactions.

Exact numbers are hard to come by, and of course vary from merchant to merchant, but probably a little under 10% of Internet transactions get billed back to the originating merchant because of credit card fraud or disputes. This is a *huge* amount! Consumer businesses like Amazon simply accept this as a cost of doing business on the Net and work it into their price structure, but obviously this isn't going to work for six figure business-to-business transactions. Nobody wants to send out \$200,000 of masonry supplies only to have the purchaser claim they never made or received the order. Before business-to-business transactions can move onto the Internet, a method needs to be developed that can verify that an order was in fact made by a particular person and that this person is who he or she claims to be. Furthermore, this has to be enforceable in court. (It's a sad fact of American business that many companies won't do business with anyone they can't sue.)

Part of the solution to the problem is digital signatures — the electronic equivalent of ink on paper. To digitally sign a document, you calculate a hash code for the document using a known algorithm, encrypt the hash code with your private key, and attach the encrypted hash code to the document. Correspondents can decrypt the hash code using your public key and verify that it matches the document. However, they can't sign documents on your behalf because they don't have your private key. The exact protocol followed is a little more complex in practice, but the bottom line is that your private key is merged with the data you're signing in a verifiable fashion. No one who doesn't know your private key can sign the document.

The scheme isn't foolproof — it's vulnerable to your private key being stolen, for example-but it's probably as hard to forge a digital signature as it is to forge a real ink-on-paper signature. However, there are also a number of less obvious attacks on digital signature protocols. One of the most important is changing the data that's signed. Changing the data that's signed should invalidate the signature, but it doesn't if the changed data wasn't included in the first place. For example, when you submit an HTML form, the only things sent are the values that you fill into the form's fields and the names of the fields. The rest of the HTML markup is not included. You may agree to pay \$1500 for a new 450 MHz Pentium II PC running Windows NT, but the only thing sent on the form is the \$1500. Signing this number signifies what you're paying, but not what you're paying for. The merchant can then send you two gross of flushometers and claim that's what you bought for your \$1500. Obviously, if digital signatures are to be useful, all details of the transaction must be included. Nothing can be omitted.

The problem gets worse if you have to deal with the U.S. federal government. Government regulations for purchase orders and requisitions often spell out the contents of forms in minute detail, right down to the font face and type size. Failure to adhere to the exact specifications can lead to your invoice for \$20,000,000 worth of depleted uranium artillery shells being rejected. Therefore, you not only need to establish exactly what was agreed to; you also need to establish that you met all legal requirements for the form. HTML's forms just aren't sophisticated enough to handle these needs.

XML, however, can. It is almost always possible to use XML to develop a markup language with the right combination of power and rigor to meet your needs, and this example is no exception. In particular UWI.COM has proposed an XML application called the Extensible Forms Description Language (XFDL) for forms with extremely tight legal requirements that are to be signed with digital signatures. XFDL further offers the option to do simple mathematics in the form, for instance to automatically fill in the sales tax and shipping and handling charges and total up the price.

UWI.COM has submitted XFDL to the W3C, but it's really overkill for Web browsers, and thus probably won't be adopted there. The real benefit of XFDL, if it becomes widely adopted, is in business-to-business and business-to-government transactions. XFDL can become a key part of electronic commerce, which is not to say it *will* become a key part of electronic commerce. It's still early, and there are other players in this space.

Human Resources Markup Language

HireScape's Human Resources Markup Language (HRML) is an XML application that provides a simple vocabulary for describing job openings. It defines elements matching the parts of a typical classified want ad such as companies, divisions, recruiters, contact information, terms, experience, and more. A job listing in HRML might look something like the code in Listing 2-10.

Listing 2-10: A Job Listing in HRML

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

  <COMPANY>

    <CO_NAME>IDG Books</CO_NAME>
    <CO_INTERNET_ADDR>
      <CO_HOME_PAGE>http://www.idgbooks.com/</CO_HOME_PAGE>
      <CO_JOBS_PAGE>
        http://www.idgbooks.com/cgi-
        bin/gatekeeper.pl?uidg4841:%2Fcompany%2Fjobs%2Findex.html
      </CO_JOBS_PAGE>
    </CO_INTERNET_ADDR>
```

```

</COMPANY>

<JOB>

  <JOB_METADATA>
    <JOB_LOADED_DT>09/10/1998</JOB_LOADED_DT>
    <JOB_LOADED_URL>
      http://www.idgbooks.com/cgi-
bin/gatekeeper.pl?uidg4841:%2Fcompany%2Fjobs%2Findex.html
    </JOB_LOADED_URL>
  </JOB_METADATA>

  <JOB_DATA>

    <JOB_TITLE>Web Development Manager</JOB_TITLE>

    <JOB_NUMBER_AVAIL>1</JOB_NUMBER_AVAIL>
    <JOB_YEARS_EXP>3</JOB_YEARS_EXP>
    <JOB_DESC>
      This position is responsible for the technical
      and production functions of the Online
      group as well as strategizing and implementing
      technology to improve the IDG Books web sites.
      Skills must include Perl, C/C++, HTML, SQL, JavaScript,
      Windows NT 4, mod-perl, CGI, TCP/IP, Netscape servers
      and Apache server. You must also have excellent
      communication skills, project management, the ability
      to communicate technical solutions to non-technical
      people and management experience.
    </JOB_DESC>

    <JOB_KEYWORDS>
      Perl, C/C++, HTML, SQL, JavaScript, Windows NT 4,
      mod-perl, CGI, TCP/IP, Netscape server, Apache server
    </JOB_KEYWORDS>

    <JOB_TERMS PAY="Salaried" TYPE="Full-time">
      $60,000
    </JOB_TERMS>

    <JOB_LOCATION CITY="Foster City" STATE="California"
      STATE_ABBR="CA" POSTAL_CODE="94404" COUNTRY="USA">
    </JOB_LOCATION>

  </JOB_DATA>

</JOB>

<RESPONSE>

  <RESP_EMAIL>cajobs@idgbooks.com</RESP_EMAIL>
  <POSTAL_ADDR ENTITY_TYPE="Response">

```

Continued

Listing 2-10 (continued)

```
<ADDR_LINE_1>Dee Harris, HR Manager</ADDR_LINE_1>
<ADDR_LINE_2>919 E. Hillsdale Blvd.</ADDR_LINE_2>
<ADDR_LINE_3>Suite 400</ADDR_LINE_3>
<CITY>Foster City</CITY>
<STATE>CA</STATE>
<POSTAL_CODE>94404</POSTAL_CODE>
</POSTAL_ADDR>

</RESPONSE>

</HRML_JOB>
```

Although you could certainly define a style sheet for HRML, and use it to place job listings on Web pages, that's not its main purpose. Instead HRML is designed to automate the exchange of job information between companies, applicants, recruiters, job boards, and other interested parties. There are hundreds of job boards on the Internet today as well as numerous Usenet newsgroups and mailing lists. It's impossible for one individual to search them all, and it's hard for a computer to search them all because they all use different formats for salaries, locations, benefits, and the like.

But if many sites adopt HRML, then it becomes relatively easy for a job seeker to search with criteria like "all the jobs for Java programmers in New York City paying more than \$100,000 a year with full health benefits." The IRS could enter a search for all full-time, on-site, freelance openings so they'd know which companies to go after for failure to withhold tax and pay unemployment insurance.

In practice, these searches would likely be mediated through an HTML form just like current Web searches. The main difference is that such a search would return far more useful results because it can use the structure in the data and semantics of the markup rather than relying on imprecise English text.

Resource Description Framework

XML adds structure to documents. The Resource Description Framework (RDF) is an XML application that adds semantics. RDF can be used to specify anything from the author and abstract of a Web page to the version and dependencies of a software package to the director, screenwriter, and actors in a movie. What links all of these uses is that what's being encoded in RDF is not the data itself (the Web page, the software, the movie) but information about the data. This data about data is called *meta-data*, and is RDF's *raison d'être*.

An RDF vocabulary defines a set of elements and their permitted content that's appropriate for meta-data in a given domain. RDF enables communities of interest to standardize their vocabularies and share those vocabularies with others who may extend them. For example, the Dublin Core is an RDF vocabulary specifically designed for meta-data about Web pages. Educom's Instructional Metadata System (IMS) builds on the Dublin Core by adding additional elements that are useful when describing school-related content like learning level, educational objectives, and price.

Of course, although RDF can be used for print-publishing systems, video-store catalogs, automated software updates, and much more, it's likely to be adopted first for embedding meta-data in Web pages. RDF has the potential to synchronize the current hodge-podge of <META> tags used for site maps, content rating, automated indexing, and digital libraries into a unified collection that all of these tools understand. Once RDF meta-data becomes a standard part of Web pages, search engines will be able to return more focused, useful results. Intelligent agents can more easily traverse the Web to find information you want or conduct business for you. The Web can go from its current state as an unordered sea of information to a structured, searchable, understandable store of data.

As the name implies, RDF describes *resources*. A resource is anything that can be addressed with a URI. The description of a resource is composed of a number of properties. Each property has a type and a value. For example, <DC:Format>HTML </DC:Format> has the type "DC:Format" and the value "HTML". Values may be text strings, numbers, dates, and so forth, or they may be other resources. These other resources can have their own descriptions in RDF. For example, the code in Listing 2-11 uses the Dublin Core vocabulary to describe the Cafe con Leche Web site.

Listing 2-11: An RDF description of the Cafe con Leche home page using the Dublin Core vocabulary

```
<RDF:RDF
  xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:DC="http://purl.org/DC/">

  <RDF:Description about="http://metalab.unc.edu/xml/">
    <DC:Creator>Elliotte Rusty Harold</DC:Creator>
    <DC:Language>en</DC:Language>
    <DC:Format>HTML</DC:Format>
    <DC>Date>1999-08-19</DC:date>
    <DC:Type>home page</DC:Type>
    <DC:Title>Cafe con Leche</DC:Title>
  </RDF:Description>

</RDF:RDF>
```

RDF will be used for version 2.0 of the Platform for Internet Content Selection (PICS) and the Platform for Privacy Preferences (P3P) as well as for many other areas where meta-data is needed to describe Web pages and other kinds of content.

XML for XML

XML is an extremely general-purpose format for text data. Some of the things it is used for are further refinements of XML itself. These include the XSL style-sheet language, the XLL-linking language, and the Document Content Description for XML.

XSL

XSL, the Extensible Style Language, is itself an XML application. XSL has two major parts. The first part defines a vocabulary for transforming XML documents. This part of XSL includes XML tags for trees, nodes, patterns, templates, and other elements needed for matching and transforming XML documents from one markup vocabulary to another (or even to the same one in a different order).

The second part of XSL defines an XML vocabulary for formatting the transformed XML document produced by the first part. This includes XML tags for formatting objects including pagination, blocks, characters, lists, graphics, boxes, fonts, and more. A typical XSL style sheet is shown in Listing 2-12:

Listing 2-12: An XSL style sheet

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/TR/WD-xsl"
  xmlns:fo="http://www.w3.org/TR/WD-xsl/FO"
  result-ns="fo">
  <xsl:template match="/">
    <fo:basic-page-sequence >
      <xsl:apply-templates/>
    </fo:basic-page-sequence>
  </xsl:template>

  <xsl:template match="ATOM">
    <fo:block font-size="10pt" font-family="serif"
      space-before="12pt">
      <xsl:value-of select="NAME"/>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>
```

We'll explore XSL in great detail in Chapters 14 and 15.

XLL

The Extensible Linking Language, XLL, defines a new, more general kind of link called an XLink. XLinks accomplish everything possible with HTML's URL-based hyperlinks and anchors. However, any element can become a link, not just A elements. For instance a `footnote` element can link directly to the text of the note like this:

```
<footnote xlink:form="simple" href="footnote7.xml">7</footnote>
```

Furthermore, XLinks can do a lot of things HTML links can't. XLinks can be bi-directional so readers can return to the page they came from. XLinks can link to arbitrary positions in a document. XLinks can embed text or graphic data inside a document rather than requiring the user to activate the link (much like HTML's `` tag but more flexible). In short, XLinks make hypertext even more powerful.

A small icon of a book with the text "Cross-Reference" written on it.

XLinks are discussed in more detail in Chapter 16, *XLinks*.

DCD

XML's facilities for declaring how the contents of an XML element should be formatted are weak to nonexistent. For example, suppose as part of a date, you set up `MONTH` elements like this:

```
<MONTH>9</MONTH>
```

All you can say is that the contents of the `MONTH` element should be character data. You cannot say that the month should be given as an integer between 1 and 12.

A number of schemes have been proposed to use XML itself to more tightly restrict what can appear in the contents of any given element. One such proposal is the Document Content Description, (DCD). For example, here's a DCD that declares that `MONTH` elements may only contain an integer between 1 and 12:

```
<DCD>
  <ElementDef Type="MONTH" Model="Data" Datatype="i1"
    Min="1" Max="12" />
</DCD>
```

There are more examples I could show you of XML used for XML, but the ones I've already discussed demonstrate the basic point: XML is powerful enough to describe and extend itself. Among other things, this means that the XML specification can remain small and simple. There may well never be an XML 2.0 because any major additions that are needed can be built out of raw XML rather

than becoming new features of the XML. People and programs that need these enhanced features can use them. Others who don't need them can ignore them. You don't need to know about what you don't use. XML provides the bricks and mortar from which you can build simple huts or towering castles.

Behind-the-Scene Uses of XML

Not all XML applications are public, open standards. A lot of software vendors are moving to XML for their own data simply because it's a well-understood, general-purpose format for structured data that can be manipulated with easily available cheap and free tools.

Microsoft Office 2000 promotes HTML to a coequal file format with its native binary formats. However, HTML 4.0 doesn't provide support for all of the features Office requires, such as revision tracking, footnotes, comments, index and glossary entries, and more. Additional data that can't be written as HTML is embedded in the file in small chunks of XML. Word's vector graphics will be stored in VML. In this case, embedded XML's invisibility in standard browsers is the crucial factor.

Federal Express uses detailed tracking information as a competitive advantage over other shippers like UPS and the Post Office. First that information was available through custom software, then through the Web. More recently FedEx has begun beta testing an API/library that third-party and in-house developers can use to integrate their software and systems with FedEx's. The data format used for this service is XML.

Netscape Navigator 5.0 supports direct display of XML in the Web browser, but Netscape actually started using XML internally as early as version 4.5. When you ask Netscape to show you a list of sites related to the current one you're looking at, your browser connects to a CGI program running on a Netscape server. The data that server sends back is XML. Listing 2-13 shows the XML data for sites related to <http://metalab.unc.edu/>.

Listing 2-13: XML data for sites related to <http://metalab.unc.edu/>

```
<?xml version="1.0"?>
<RDF:RDF>
<RelatedLinks>
<aboutPage
href="http://info.netscape.com/fwd/r1/http://metalab.unc.edu:80
/*">
```

```
</aboutPage>
<child instanceOf="Separator1"></child>
<child
href="http://info.netscape.com/fwd/r1/http://www.sun.com/"
name="Sun Microsystems">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://www.unc.edu/"
name="Unc">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://sunsite.sut.ac.jp/"
name="SunSITE Japan">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://sunsite.nus.sg/"
name="SunSITE Singapore">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://sunsite.berkeley.e
du/" name="Berkeley Digital Library SunSITE">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://www.sun.com/sunsit
e" name="SunSITE on the net">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://www.sunsite.auc.dk
/" name="SunSITE Denmark">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://sunsite.edu.cn/"
name="SunSITE China">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://sunsite.stanford.o
rg/" name="Stanford University SunSITE">
</child>
<child
href="http://info.netscape.com/fwd/r1/http://www.cdromshop.com/
cdshop/desc/p.061590000085.html" name="SunSITE Archive">
</child>
<child instanceOf="Separator1"></child>
<child instanceOf="Separator1"></child>
<child href="http://home.netscape.com/escapes/smart_browsing"
name="Learn About Smart Browsing...">
</child>
</RelatedLinks>
</RDF:RDF>
```

This all happens completely behind the scenes. The users never know that the data is being transferred in XML. The actual display is a menu in Netscape Navigator, not an XML or HTML page.

This really just scratches the surface of the use of XML for internal data. Many other projects that use XML are just getting started, and many more will be started over the next year. Most of these won't receive any publicity or write-ups in the trade press, but they nonetheless have the potential to save their companies thousands of dollars in development costs over the life of the project. The self-documenting nature of XML can be as useful for a company's internal data as for its external data. For instance, many companies right now are scrambling to try and figure out whether programmers who retired 20 years ago used two-digit dates. If that were your job, would you rather be pouring over data that looked like this:

```
3c 79 65 61 72 3e 39 39 3c 2f 79 65 61 72 3e
```

or like this:

```
<YEAR>99</YEAR>
```

Unfortunately many programmers are now stuck trying to clean up data in the first format. XML even makes the mistakes easier to find and fix.

Summary

This chapter has just begun to touch the many and varied applications to which XML has been and will be put. Some of these applications like CML, MathML, and MusicML are clear extensions to HTML for Web browsers. But many others, like OFX, XFDL, and HRML, go into completely new directions. And all of these applications have their own semantics and syntax that sits on top of the underlying XML. In some cases, the XML roots are obvious. In other cases, you could easily spend months working with it and only hear of XML tangentially. In this chapter, you explored the following applications to which XML has been put to use:

- ♦ Molecular sciences with CML
- ♦ Science and math with MathML
- ♦ Webcasting with CDF
- ♦ Classic literature
- ♦ Multimedia with SMIL and HTML+TIME
- ♦ Software updates through OSD
- ♦ Vector graphics with both PGML and VML

- ♦ Music notation in MusicML
- ♦ Automated voice responses with VoxML
- ♦ Financial data with OFX
- ♦ Legally binding forms with XFDL
- ♦ Human resources job information with HRML
- ♦ Meta-data through RDF
- ♦ XML itself, including XSL, XLL, and DCD, to refine XML
- ♦ Internal use of XML by various companies, including Microsoft, Federal Express, and Netscape

In the next chapter, you will begin writing your own XML documents and displaying them in Web browsers.



