Introduction

Everyone seems to be talking about Extensible Markup Language (XML) these days. You know how mothers are — I can't even visit my relatives over the holidays without my mom broaching the topic of XML at Thanksgiving dinner. Yes, XML has become quite a buzzword, but Extensible Stylesheet Language Transformations (XSLT) is the power that takes all this XML and turns it into something useful and flexible.

XSLT is a language used to transform XML documents into something new. It can take one XML document and output that document's information into a completely different structure or turn XML into an HTML document for viewing on the Web. XSLT isn't a general-purpose programming language, such as Java or Visual Basic; its focus is solely on transforming XML.

Before I get any farther along, I have to point out the "elephant in the room" — XSLT's long-winded name. Who came up with that name anyway? I think the people responsible should be forced to say, "Extensible Stylesheet Language Transformations," aloud ten times and hope their tongues don't fall off! XSLT's full name may be a mouthful, but this book carves up each piece of XSLT into manageable, chewable morsels.

XSLT can be confusing if you don't have a solid understanding of its syntax, quirky abbreviations, and the esoteric terminology it sometimes uses. But *XSLT For Dummies* can help you write XSLT stylesheets and, just as important, grasp why and how transformations work. In addition, see `www.dummies.com/extras/xsltfd` for code examples as well as a helpful editing tool of my own creation which I call the X-Factor.

Above all, you can use *XSLT For Dummies* as your guide for quickly developing the know-how of XSLT — without drowning in technical gobbledygook.

What I Assume About You

XSLT For Dummies starts from the bottom and works its way up: It doesn't assume that you have any previous knowledge of XSLT or XML. If you have some know-how of HTML or programming, I think you can grasp some of the concepts of XSLT quicker than a beginner, but I don't expect you to know HTML or anything about programming beforehand.

How This Book Is Organized

My aim is to help you find out how to become productive with XSLT quickly so that you can transform XML documents into virtually any kind of output imaginable. With that goal in mind, *XSLT For Dummies* is divided into five parts, each of which builds upon the previous ones.

Part I: Getting Started with XSLT

Part I kicks off your journey. You start by finding out about the core concepts of XSLT and how it fits in with HTML and all the other X technologies out there, such as XML, XSL, and XPath. You get your feet wet by writing your first XSLT stylesheet and transforming an XML document.

After you finish that, you can say *XSLT* to your buddies at work and actually have a grasp of what you're talking about when you throw around those X terms.

Part II: Becoming an XSLT Transformer

This part takes you into the belly of the beast: stylesheets, template rules, and XPath expressions. But don't worry — you won't lose your appetite. You begin by looking at stylesheets and find out, in everyday terms, the often-confusing subject of document trees and recursion. From there, you start pulling information out of XML documents and outputting it into various forms.

By the time you're done with this part, you'll be able to say, "Extensible Stylesheet Language Transformations," without stumbling over any of the twelve syllables.

Part III: Prime Time XSLT

In this part, you dive deeper into the thick of things. You find out how XSLT handles programming concepts such as if/then statements, loops, and variables, and how to include them in stylesheets. Don't worry if you've never programmed before; these concepts become clear as you read through the chapters. You also find out about how to take advantage of the more advanced capabilities of XSLT and XPath to create more powerful transformations.

I predict that after you finish this part, at least once you'll have unintentionally ordered an XSLT sandwich on rye at your local deli.

Part IV: eXtreme XSLT

As you read through Part IV, you can begin to call yourself a bona fide XSLT Transformer. You find out how to create effective XSLT stylesheets and apply them under various conditions. You find out about how to combine stylesheets and even add your own extensions. You also get the inside scoop on debugging transformations.

A word of warning: By now, all this XSLT will be swimming around in your head. You may find yourself mingling at a social event and leading with the line: "Apply any good templates lately?"

Part V: The Part Of Tens

In this part, I guide you through some practical tips and information that can make your life easier when you work with XSLT. I start out by demystifying the ten most confusing things about XSLT. Then I detail what I consider to be the ten best XSLT resources on the Web. I conclude by giving you the details on ten XSLT processors that you can download online.

Conventions Used in This Book

Snippets of XSLT code are scattered throughout the book and are often used to introduce you to a feature of the markup language. They appear in the chapters like this:

```
<xsl:stylesheet
           xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           version="1.0">
  <xsl:template match="id">
     <h1><xsl:apply-templates/></h1>
  </xsl:template>

</xsl:stylesheet>
```

If you type these stylesheets by hand, pay close attention and type the text exactly as shown to ensure that the stylesheet transforms properly. Generally, spaces don't matter much, although depending on where they are, they could change the look of the output from a transformation. However, case sensitivity is important for any XML or XSLT element. I use lowercase text in all the examples, so I recommend getting used to typing lowercase to avoid confusion or problems down the line.

If XSLT element names or instructions appear within the regular text, they look `like this`.

Icons Used in This Book

Tips draw attention to key points that can save you time and effort.

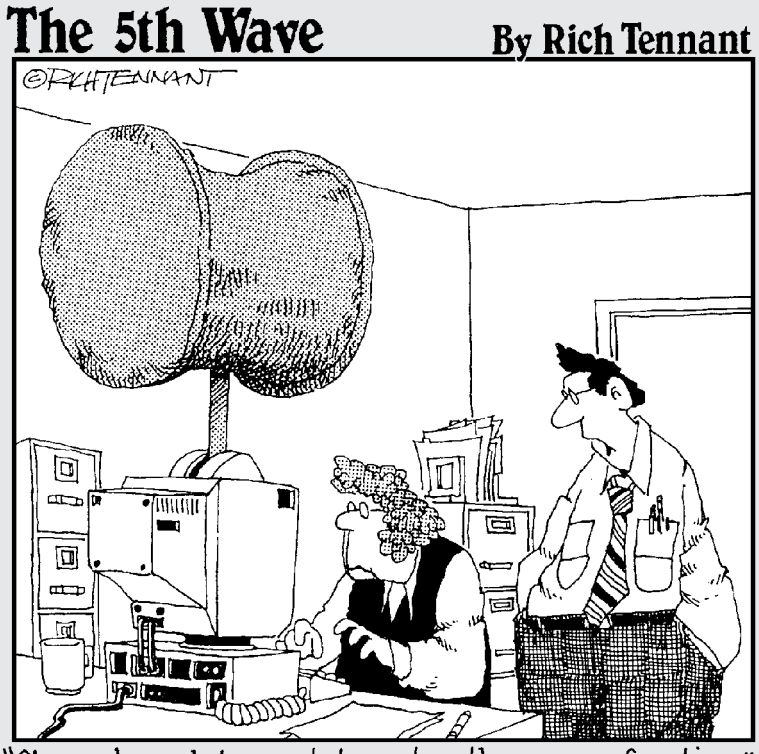
Pay close attention to this icon; it highlights something that's important to your understanding of XSLT or how to use it.

Heed the Warning icon because it can save you from the pitfalls of XSLT pain and agony.

Technical Stuff is the techno-mumbo-jumbo that's interesting but probably only for geeks. So, reading these sections can provide useful information, but feel free to skip them.

Part I

Getting Started with XSLT



In this part . . .

You watched the *X-Files* and then you saw *X-Men* on the big screen, but these were only warm-ups for the real deal — the X-Team. In this part, you find out all about the X-Team members, including XML, XSL, XSLT, and XPath, and how they work together. You also get your feet wet by transforming your first XML document using XSLT.

Chapter 1

Introducing the X-Team

In This Chapter

- Finding out about XML, XSL, XSLT, and XPath
- Knowing the difference between XSL and XSLT
- Looking at the X-Team from an HTML perspective

As a sports fan, I enjoy watching all kinds of team sports, whether football, basketball, baseball, or an Olympic team competition. I've noticed that regardless of the sport, great teams have two things in common. First, they have very talented individuals on them. Second, they function well as a team; I find hardly anything more thrilling in sports than seeing a squad of talented athletes working together cohesively. (Of course, it goes without saying that the *most* exciting part of sports is the "I'm going to Disneyworld" commercials!)

Although this book focuses on eXtensible Stylesheet Language Transformations, or XSLT, you'll quickly discover that XSLT is an important component of a closely related set of technologies that I affectionately call the X-Team. This "Dream X-Team" consists of: XML, XSL, XSLT, and XPath. (For the techies out there, that's shorthand for eXtensible Markup Language, eXtensible Stylesheet Language, XSL Transformations, and XML Path Language.) Each of these technologies is powerful, but each gets its true strength through interrelationships. So, although I concentrate much of the attention in this book on XSLT, never think of it as something independent of its teammates.

As you start your XSLT journey, I begin by introducing you to the X-Team members, each of which has a separate but intertwined responsibility.

XML: Storing Your Data

The original member of the X-Team is eXtensible Markup Language (XML), the granddaddy of them all. All other X-Team members are designed to work with or act upon XML. A relatively recent innovation, XML was conceived

primarily by Jon Bosak as a way to make working with information delivered over the Web easier. Then in 1998, XML was standardized by the World Wide Web Consortium (W3C), the international standards body for the Web.

Since its beginnings, the Web has used HyperText Markup Language (HTML) to display content. HTML documents are stored on Web servers and then sent on demand to a Web browser, such as Microsoft Internet Explorer or Netscape Navigator. The browser then displays the HTML as a Web page. Figure 1-1 illustrates this process.

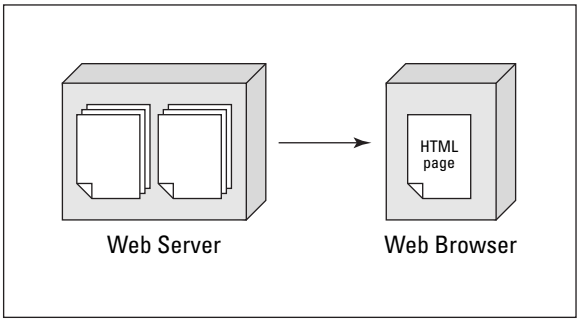


Figure 1-1: Displaying information over the Web.

HTML comes up short

HTML has become so wildly popular largely because it's very easy to learn and work with; heck, even my 7-year-old can create a Web page using Microsoft FrontPage, and my 9-year-old can write HTML by hand. The markup language was originally designed purely as a way to format and lay out information. However, because people have wanted to use the Web for nearly every task under the sun, HTML has been forced to do far more than was ever intended.

Consider a familiar scenario: A company wants to put information stored in a database onto its Web site. A sampling of its data might look something like Table 1-1.

Table 1-1 Sample Customer Database

ID	*Name*	*City*	*State*	*Zip*
100	Ray Kinsella	Anderson	IN	46011
101	Rick Blaine	Manchester	NH	02522

To present this information on the Web, these database records must be converted into HTML text and formatted properly as a table so that they can be viewed in a Web browser.

```
<table border="1">
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>City</th>
    <th>St</th>
    <th>Zip</th>
  </tr>
  <tr>
    <td>100</td>
    <td>Ray Kinsella</td>
    <td>Anderson</td>
    <td>IN</td>
    <td>46011</td>
  </tr>
  <tr>
    <td>101</td>
    <td>Rick Blaine</td>
    <td>Manchester</td>
    <td>NH</td>
    <td>02522</td>
  </tr>
</table>
```

Look closely at the above code to see how HTML falls short. I turn meaningful clusters of information into a format that looks good in a browser but isn't useful for much else. In a database, related fields such as ID, Name, and Address make up a customer record, but after they have been converted to HTML, they're just row and column formatting instructions and their contents — thus, the concept of a *customer* is gone.

Such a solution would be acceptable if you only want to display information in a Web browser, but many people are discovering needs that go far beyond that. For example, searching for information within an HTML document is very limited. How would I be able to retrieve from my HTML file the names of all of my customers from Indiana who spend over $1,000 annually? That kind of query is far beyond the scope of HTML. And, even if I were to develop some convoluted way to get this information through JavaScript, I'd have to throw all that away if I ever wanted to move my information to another non-HTML environment, such as a Java application, Windows program, or even a cellular phone.

Think of HTML as a sort of information blender: Add a dash of data and a pinch of formatting instructions into the pitcher, turn the power on high, and out comes a pureed mixture of the two. Like creating a milkshake by mixing ice

cream, chocolate syrup, vanilla, and milk in a blender, imagine the impossibility of trying to extract the vanilla from the milkshake after it's been blended. This no-win backward mobility is the futile attempt to mine useful information from HTML documents.

In other words: Yes, the shake tastes great, but don't try to use the raw materials again for a different purpose.

XML to the rescue

Developed as a response to the information-blender effect of HTML, XML is simply a practical way to work with structured information on the Web. The motivation of its inventors was to assemble structured data into something that was similar to HTML — so that data could be easily readable by people like you and me — but different enough from HTML so that it's freely expandable to effectively describe the data that it contains.

Whether you realize it or not, almost all the information used on the Web has a natural structure or organization to it and thus can be expressed using XML. Some everyday examples include:

- **The contents of a letter**

```
<letter>
  <date>March 31, 2002</date>
  <salutation>Dear Sir:</salutation>
  <text>Thanks for your recent article on Swiss Cheese
        chips.
  However, I don't think you gave enough credit to the
        farmer
  who invented the Swiss Cheese chip - Charley
        Cowley.</text>
  <closing>Warm Regards,</closing>
  <signature>Mrs. Charlie Cowley</signature>
</letter>
```

- **Dialogue from a movie**

```
        <dialogue>
  <rick>I'm saying it because it's true. Inside of us, we
        both know you belong with Victor. You're part of
        his work, the thing that keeps him going. If that
        plane leaves the ground and you're not with him,
        you'll regret it. Maybe not today. Maybe not
        tomorrow, but soon and for the rest of your
        life.</rick>
  <ilsa>But what about us?</ilsa>
  <rick>We'll always have Paris. We didn't have, we, we
        lost it until you came to Casablanca. We got it
        back last night.</rick>
```

```
  <ilsa>When I said I would never leave you.</ilsa>
  <rick> And you never will. But I've got a job to do,
         too. Where I'm going, you can't follow. What I've
         got to do, you can't be any part of. Ilsa, I'm no
         good at being noble, but it doesn't take much to
         see that the problems of three little people
         don't amount to a hill of beans in this crazy
         world. Someday you'll understand that. Now,
         now... Here's looking at you kid.</rick>
</dialogue>
```

- **Those customer records of Ray and Rick**

```
<customers>
  <customer>
    <id>100</id>
    <name>Ray Kinsella</name>
    <city>Anderson</city>
    <state>IN</state>
    <zip>46011</zip>
  </customer>
  <customer>
    <id>101</id>
    <name>Rick Blaine</name>
    <city>Manchester</city>
    <state>NH</state>
    <zip>02522</zip>
  </customer>
</customers>
```

- **A Web page**

```
<html>
  <head>
    <title>My Home Page</title>
  </head>
  <body>
    <h1>Heading</h1>
  </body>
</html>
```

From these examples, you can see that XML describes information in a very logical and straightforward manner. Put descriptive tags before and after the text values and you've just about got an XML document. XML isn't rocket science!

HTML is standardized with a fixed set of formatting tags or *elements* to define different parts of a document. An `<h1>` element identifies a Level 1 Header, and `<b>` denotes bolded text. In contrast, the only thing standardized about XML is its syntax rules, not its actual tags; this is what makes XML so flexible. For example, a bank can define a set of XML tags to describe its financial data:

```
<account id="10001010">
  <type>Checking</type>
  <rating level="-5"/>
  <customer preferred="no way, hosea">
    <firstname>John</firstname>
    <lastname>Charles</lastname>
    <address>123 Main Street</address>
    <city>Fremont</city>
    <state>CA</state>
    <zip>94425</zip>
  </customer>
</account>
```

Or, a pizza store chain can come up with its own set of XML elements that describes their pizzas.

```
<pizza>
  <size value="Mega"/>
  <crust type="Thick and Chewy"/>
  <toppings>Olives, Sausage, Pepperoni, Lima Beans</toppings>
  <cookingtime>30</cookingtime>
</pizza>
```

A set of defined XML tags used for a particular purpose is an *XML vocabulary*.

However, as great as it is at organizing information, XML by its very nature is a raw material. XML is of little use by itself and needs help from its X-Team teammates to actually make its information usable in the real world.

XSL: Presenting XML Data

Time to pass the baton to the second member of the X-Team: Extensible Stylesheet Language (XSL). XSL is charged with styling or laying out XML documents into a form that makes sense to its intended audience. As shown in Figure 1-2, you use XSL to define a set of formatting rules that are referred to when an XML document is processed.

For example, if I want to format the letter from the preceding "XML to the rescue" section, I use XSL to create a few rules, such as

- **When a `<date>` element is found, italicize the date's text.**
- **When a `<salutation>` element is found, indent salutation's text.**
- **When a `<closing>` element is found, add an extra line after it.**

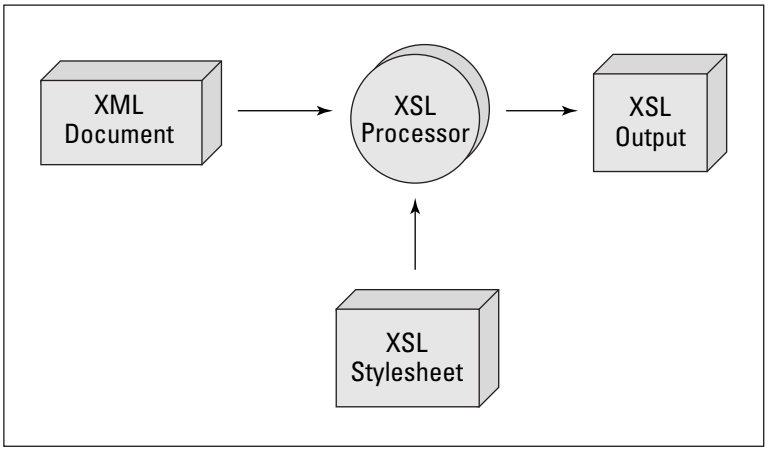


Figure 1-2: Using XSL to apply formatting to XML documents.

XSL rules like these are contained in an *XSL Stylesheet*, which is just a plain text file filled with formatting instructions that look like the following example.

```
<fo:page-sequence master-name="easy">
     <fo:flow flow-name="xsl-region-body">
          <fo:block font-family="Serif">Serif
      font</fo:block>
     </fo:flow>
</fo:page-sequence>
```

Note that this XSL is written in something that resembles XML. That is more than mere coincidence because, ironically, XSL is actually written in XML and is itself an XML vocabulary.

If your head is spinning around, hang on. XSL is used to format XML, which in turn is used as the language for defining XSL. The circular logic can be confusing, but fortunately, you don't need to worry about the particulars of how that all works — just know that it does. Actually, the fact that XSL uses XML to describe its instructions makes it far easier to learn than trying to grasp yet another language syntax.

When XSL was conceived by the W3C, the original intention of XSL was simply to provide a way to format XML data. However, after people began to use XML in the real world, it was discovered that something more was needed besides assigning font colors and margin indentions to the content. True, developers needed to be able to style XML in a way that was easily readable, but they also discovered a need to change an XML document from one XML structure to another, as well as to have the ability to easily convert

XML into HTML and other output options. Taking up this charge, the W3C expanded the scope of XSL to support transforming, and in doing so, gave birth to XSL Transformations (XSLT).

XSLT: Transforming Your XML Data

The third member of the X-Team is XSL Transformations (XSLT). XSLT is analogous to that high-priced rookie on a professional sports team that unseats the veteran player: XSL was supposed to be the killer technology to enable XML to achieve widespread adoption, but XSLT's ability to convert XML data into any wanted output has proven so compelling that it essentially swallowed up XSL. In fact, when people today talk about XSL, they're almost always referring to XSLT.

XSL is actually composed of two independent parts: XSLT for transforming XML from one structure to another; and XSL Formatting Objects and Formatting Properties for formatting XML documents.

The key reason for all this enthusiasm in support of XSLT is that XML documents often need to serve multiple audiences. The XML needs to be formatted so that it can be viewed in a Web browser, and the same XML may need to be tweaked to conform to a new trading partner's own XML structure. See Figure 1-3 for an illustration of this relationship.

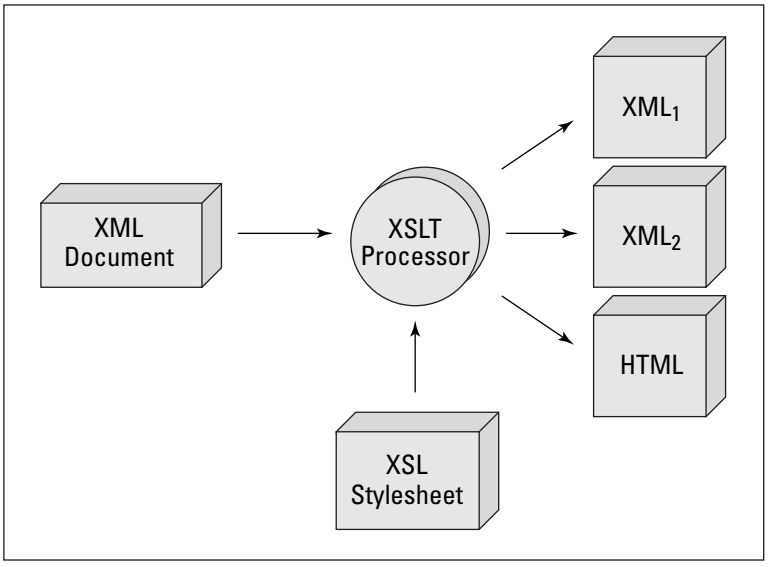


Figure 1-3: XSLT transforms XML into a variety of outputs.

To illustrate, suppose that I want to change the XML definition of a customer from the original format of

```
<customer>
<id>101</id>
<name>Rick Blaine</name>
<city>Manchester</city>
<state>NH</state>
<zip>02522</zip>
</customer>
```

into this:

```
<customer id="101">
<fullname>Rick Blaine</fullname>
<address city="Manchester" state="NH"
zipcode="02522"/>
</customer>
```

Before XSLT came along, I'd have to dust off my programming software, bury myself in a cave for a day, and write a program to do this migration process. However, with XSLT, I can transform the data from one XML format to another nearly instantly, with no true programming required.

XSLT is not a programming language as such. In fact, when written out, it doesn't even look anything like C++, Java, or Visual Basic. Like its XSL parent, XSLT rules and templates are defined by using XML.

Most programming languages transform data structures through blood, sweat, and tears. In contrast, XSLT does this work in what can best be described as *transforming by example* — you provide an example of what kind of information you'd like to see, and XSLT does the rest. For example, the following XSLT snippet changes the `name` element to `fullname` in the output document.

```
<xsl:template match="name">
  <fullname>
  <xsl:apply-templates/>
  </fullname>
</xsl:template>
```

(I get into the specifics of how XSLT template rules work in Chapter 4.)

However, as powerful as XSLT is, it needs help to do its transformational magic from our last X-Team member: XPath. XPath specializes in picking out the specific nuggets of information from one XML document in order for XSLT to fit it neatly into another one.

XPath: Seeking Out Your Data

XPath is the spy or seeker of the X-Team who is charged with going into an XML document and picking out the requested information for XSLT. Without the ability to precisely locate information in an XML document, the ability to transform or do anything special with XML is minimal.

Any XSLT transformation must be set up to answer two questions:

- **Input:** What information from the original XML document do you want?
- **Output:** How would you like that information structured in the output document?

XSLT relies on XPath to answer the first question, as shown in Figure 1-4.

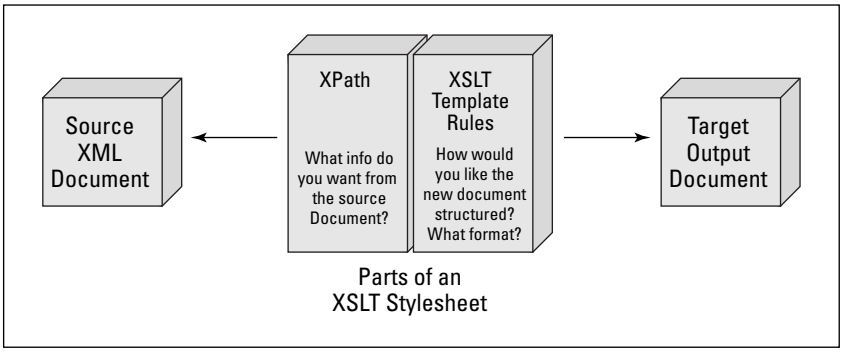


Figure 1-4: XSLT uses XPath to retrieve data from XML documents.

XSLT looks at an XML document element by element, so XPath expressions are used to tell what your XSLT stylesheet should look for as it goes through the XML document. Looking closer at the preceding XSLT example, the XPath expression `name` tells XSLT what information to look for, which in this case is to look for all `name` elements.

```
<xsl:template match="name">
  <fullname>
  <xsl:apply-templates/>
  </fullname>
</xsl:template>
```

This XPath expression is intuitive and easy to understand, but for more hearty needs, the syntax can be quite arcane and challenging. (I discuss XPath in detail in Chapter 5.)

Interestingly, much of the effort required to develop XSLT stylesheets is related to the input side of the equation, so throughout this book, I spend a sizeable amount of time on how to use XPath.

The X-Team through HTML Eyes

You may be approaching the X-Team after having already worked with HTML. If so, when you look at XML and XSLT, it's natural to view these new technologies through HTML eyes. Having a knowledge of HTML definitely gives you a head start in learning XML syntax; noting the similarities and differences between them is important.

Although I compare HTML and XML in this section, remember that XSL and XSLT stylesheets are both written using XML, so the same rules apply to them as to XML.

XML looks a lot like HTML . . .

If you can read HTML, you quickly see that XML looks an awful lot like HTML in terms of its syntax. For example, a document title in HTML is defined as

```
<title>My Document Title</title>
```

Like HTML, the element is the primary building block of XML. Therefore, a book title in XML might be defined to look something like this:

```
<book>War and Peace</book>
```

Additionally, XML follows HTML in using name-value pairs inside elements to provide additional descriptive information about an element.

```
<invoice id="110">
  <company>Polar Salsa Corporation</company>
</invoice>
```

In this XML snippet, the `id` attribute provides additional information related to the `invoice` element.

But XML isn't the same as HTML . . .

HTML and XML have a definite likeness, but you should watch out for some significant variations in syntax rules. The three most important are as follows.

XML is well-formed

HTML has always been lenient in some of its syntax requirements, not always forcing you to have closing tags on some of the elements, such as the paragraph (`<p>`) element. For example, both of the following lines are valid HTML:

```
<p>Hello. My name is Inigo Montoya. You killed my father.
           Prepare to die.
```

and

```
<p>Hello. My name is Inigo Montoya. You killed my father.
           Prepare to die.</p>
```

In contrast, XML is much more rigid: All XML documents must be *well-formed*, meaning that every begin tag needs to have a matching end tag:

```
<president>Andrew Jackson</president>
```

XML allows shortcuts

Although XML requires any element to have a start and end tag pair, it does allow you to combine the two tags if the element is *empty,* meaning that no text is provided between the start and end tags. For example, the following two lines are equivalent.

```
<device id="3838-2020"></device>
```

and

```
<device id="3838-2020"/>
```

XML is case sensitive

HTML is case insensitive, so long as you spell out the tag syntax correctly, the document is processed appropriately. Each of the following are valid HTML statements.

```
<body bgcolor="#FFFFFF"></body>
<BODY BGCOLOR="#FFFFFF"></BODY>
<Body Bgcolor="#FFFFFF"></Body>
```

On the other hand, XML is case sensitive, so the following statements aren't considered equal.

```
<quote>Get used to disappointment.</quote>
<QUOTE>Get used to disappointment.</QUOTE>
```

To avoid confusion, you should consistently use either all lower- or upper-case characters for the XML, XSL, and XSLT documents that you create. However, I recommend consistently using lowercase characters because this is the convention that nearly everyone follows.