1
Essential XSLT

Welcome to the world of Extensible Stylesheet Language Transformations, XSLT. This book is your guided tour to that world, which is large and expanding in unpredictable ways every minute. In this book, we're going to make that world your world. There's a lot of territory to cover, because these days XSLT is getting into the most amazing places, and in the most amazing ways. And you're going to see all of it at work in this book.

XSLT is all about handling and formatting the contents of XML documents (the companion volume to this book is *Inside XML*, New Riders, 2000). XML has become a very hot topic, and now it's XSLT's turn. XML enables you to structure the data in documents, and XSLT enables you to work with the contents of XML documents—manipulating the content and creating other documents, such as when you sort an XML employee records database or store that data in an HTML document, as well as format that data in a detailed way.

You can work with the contents of XML documents by writing your own programs that interface to XML parser applications, but that involves writing your own code. With XSLT, on the other hand, you can perform the same kinds of tasks, and there's no programming required. Rather than write your own Java, Visual Basic, or C++ to handle the contents of XML documents, you just use XSLT to specify what you want to do, and an XSLT processor does the rest. That's what XSLT is all about, and it's become the next big thing in the XML world.

XSL = XSLT + XSL-FO

XSLT itself is actually part of a larger specification, Extensible Stylesheet Language, or XSL. XSL is all about specifying the exact format, down to the millimeter, of documents. The formatting part of XSL, which is a far larger specification than XSLT is based on special formatting objects, and this part of XSL is often called XSL-FO (or XSL:FO, or XSLFO). XSL-FO is an involved topic, because styling your documents with formatting objects can be an intricate process. In fact, XSLT was originally added to XSL to make it easier to transform XML documents into documents that are based on XSL-FO formatting objects.

This book is all about XSLT, but it also provides an introduction to XSL-FO, including how to use XSLT to transform documents to XSL-FO form; after all, XSLT was first introduced to make working with XSL-FO easier. To get started, this chapter examines both XSLT and XSL-FO in overview.

A Little Background

XSL itself is a creation of the World Wide Web Consortium (W3C, `www.w3.org`), a coalition of groups originally founded by Tim Berners-Lee. The W3C is the body that releases the specifications, such as those for XSL, that are used in this book. They make XML and XSL what they are.

W3C and Style Languages

You can read about the history of W3C's work with style languages at `www.w3.org/Style/History`. It's interesting to see how much work has gone on—and how much style languages have changed over the years.

The W3C originally developed the grandfather of XML, SGML (Standard Generalized Markup Language), in the 1980s, but it was too complex to find much use, and in fact, XML (like HTML) is a simplified version of SGML. The W3C also created a style language called DSSSL (for Document Style Semantics and Specification Language) for use with SGML, and in the same way that XML was derived from SGML, XSL is based on the original DSSSL. As the W3C says: "The model used by XSL for rendering documents on the screen builds upon many years of work on a complex ISO-standard style language called the Document Style Semantics and Specification Language (DSSSL)."

However, the original part of XSL—that is, XSL-FO—has not proven easy enough to find widespread use yet either, so XSLT was introduced to make it easier to convert XML documents to XSL-FO form. As it turns out,

XSLT is what has really taken off, because it provides a complete transformation language that enables you to work with the contents of XML documents without writing programming code, transforming those documents into another XML document, HTML, or other text-based formats. The big success story here, surprising even the W3C, is XSLT.

XSLT—XSL Transformations

XSLT lets you work with the contents of XML documents directly. For example, you might have a huge XML document that holds all baseball statistics for the most recent baseball season, but you might be interested only in the statistics for pitchers. To extract the data on pitchers, you can write your own program in Java, Visual Basic, or C++ that works with XML *parsers*. Parsers are special software packages that read XML documents and pass all the data in the document, piece by piece, to your own code. You can then write a new XML document, pitchers.xml, that contains only data about pitchers.

That way of doing things works, but it involves quite a bit of programming, as well as the investment of a lot of time and testing. XSLT was invented to solve problems such as this. XSLT can be read by XSLT processors, which work on XML documents for you—all you have to do is create an XSLT stylesheet that specifies the rules you want to apply to transform one document into another. No programming is needed—and that's what makes it attractive to many people, even experienced programmers. For the baseball example, all you'd have to do is write an XSLT stylesheet that specifies what you want to do, and let the XSLT processor do the rest.

Besides transforming one XML document into another XML document, you can also transform XML documents into other types of documents, such as HTML documents, rich text (RTF) documents, documents that use XSL-FO, and others. You can also transform XML documents into other XML-based languages, such as MathML, MusicML, VML, XHTML, and more—all without programming.

In many ways, XSLT can function like a database language such as SQL (Structured Query Language, the famous database-access language), because it enables you to extract the data you want from XML documents, much like applying an SQL statement to a database. Some people even think of XSLT as the SQL of the Web, and if you're familiar with SQL, that gives you some idea of the boundless horizons available to XSLT. For example, using an XSLT stylesheet, you can extract a subset of data from an XML document, create an entire table of contents for a long document, find all elements that match a specific test—such as customers in a particular zip code—and so on. And you can do it all in one step!

XSL-FO: XSL Formatting Objects

The other part of XSL is XSL-FO, which is the formatting language part of XSL, and you'll get a taste of XSL-FO in this book. You can use XSL-FO to specify how the data in XML documents is to be presented, down to the margin sizes, fonts, alignments, header and footer size, and page width. When you're formatting an XML document, there are hundreds of items to think about, and accordingly, XSL-FO is much bigger than XSLT.

On the other hand, because of its very complexity, XSL-FO is not very popular yet, certainly not compared to XSLT. There's not much software that supports XSL-FO at this point, and none that implements anywhere near the complete standard. Just as the most common use of XSLT is to transform XML to HTML, the most common use of XSL-FO is to convert XML to formatted PDF (Portable Data Format), the format used by the Adobe Acrobat. You'll see an example of that at the end of this chapter, as well as in Chapter 11.

The W3C Specifications

W3C releases the specifications for both XML and XSL, and those specifications are what we'll be working with in this book. W3C specifications are not called standards because by international agreement, standards are created only by government-approved bodies. Instead, the W3C starts by releasing the *requirements* for a new specification. The requirements are goals, and list a sort of preview of what the specification will be all about, but the specification isn't written at that point. Next, the W3C releases specifications first as *working drafts*, which anyone may comment on, then as *candidate recommendations*, which are still subject to review, and then finally as *recommendations*, which are final.

The following list includes the XSLT-related W3C specifications that we'll be using in this book and where you can find them:

- **The complete XSL candidate recommendation** `www.w3.org/TR/xsl/`. This is the big document that specifies all there is to XSL.
- **The XSL Transformations 1.0 recommendation** `www.w3.org/TR/xslt`. XSLT's function is to transform the contents of XML documents into other documents, and it's what's made XSL so popular.
- **The XSLT 1.1 working draft** `www.w3.org/TR/xslt11`. This is the XSLT 1.1 working draft, which will not be upgraded into a recommendation—the W3C plans to add the XSLT 1.1 functionality to XSLT 2.0.
- **The XSLT 2.0 requirements** `www.w3.org/TR/xslt20req`. W3C has released the set of goals for XSLT 2.0, including more support for XML schemas.

- **The XPath 1.0 specification** `www.w3.org/TR/xpath`. You use XPath to locate and point to specific sections and elements in XML documents so that you can work with them.
- **The XPath 2.0 requirements** `www.w3.org/TR/xpath20req`. XPath is being updated to offer more support for XSLT 2.0.

XSLT Versions

The specifications for XSLT have been considerably more active than the specifications for XSL as a whole. The XSLT 1.0 recommendation was made final November 16, 1999, and that's the version that forms the backbone of XSLT today.

Next came the XSLT 1.1 working draft, and although it was originally intended to go on and become a new recommendation, some people in the W3C started working on XSLT 2.0; after a while, the W3C decided to *cancel* the XSLT 1.1 recommendation. This means that the XSLT 1.1 working draft is not going to go any further—it'll always stay in working draft form and will not become a recommendation. In other words, there will be no official version 1.1 of XSLT.

However, the W3C also says that it plans to integrate much of what was done in the XSLT 1.1 working draft into XSLT 2.0, and for that reason, I'll take a look at the XSLT 1.1 working draft in this book. I'll be sure to label material as "XSLT 1.1 working draft only" when we're discussing something new that was introduced in the XSLT 1.1 working draft.

Here are the changes from XSLT 1.0 that were made in the XSLT 1.1 working draft; note that this list is included just for reference, because most of this material probably won't mean anything to you yet:

- The result tree fragment data type, supported in XSLT 1.0, was eliminated.
- The output method no longer has complete freedom to add namespace nodes, because a process of namespace fixup is applied automatically.
- Support for XML Base was added.
- Multiple output documents are now supported with the `<xsl:document>` element.
- The `<xsl:apply-imports>` element is now allowed to have parameters.
- Extension functions can now be defined using the `<xsl:script>` function.
- Extension functions are now allowed to return *external objects*, which do not correspond to any of the XPath data types.

This book covers the XSLT 1.0 recommendation, as well as the XSLT 1.1 working draft. In fact, the W3C has raced ahead and has released the requirements for XSLT 2.0, and we'll also cover what's known of XSLT 2.0 in this book. The following list gives you an overview of the goals for XSLT 2.0:

- Add more support for the use of XML Schema-typed content with XSLT.
- Simplify manipulation of string content.
- Make it easier to use XSLT.
- Improve internationalization support.
- Maintain backward compatibility with XSLT 1.0.
- Support improved processor efficiency.

Although XSLT 2.0 won't be out for quite a while yet, I'll cover all that's known about it so far when we discuss pertinent topics. For example, the W3C's successor for HTML is the XML-based XHTML. In XSLT 1.0 the XSLT 1.1 working draft, there is no special support for XML to XHTML transformations, so we'll have to create that transformation from scratch. However, that support is coming in XSLT 2.0, and I'll mention that fact when we discuss XHTML.

That provides us with an overview, and sets the stage. Now it's time to get to work. XSL is designed to work on XML documents, so I'm going to review the structure of XML documents first. You'll be working on XML documents, but XSL stylesheets themselves are actually XML documents as well, which is something you have to keep in mind as you write them. This book assumes that you have some knowledge of both XML and HTML. (As mentioned earlier, this book is the companion volume to New Rider's *Inside XML*.)

XML Documents

It's going to be important for you to know how XML documents work, so use this section to ensure that you're up to speed. Here's an example XML document that I'll take a look at:

```
<?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>
```

Here's how this document 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 UTF-8 character encoding, which means that I'm using an eight-bit condensed version of Unicode:

```
<?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>`. You can use any name, not just DOCUMENT, for a tag, as long as the name starts with a letter or underscore (_), and 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*, and you create XML elements with an opening tag, such as `<DOCUMENT>`, followed by any element content (if any), such as text or other elements, and ending with the matching closing tag that starts with </, such as `</DOCUMENT>`. You enclose the entire document, except for processing instructions, in one element, called the *root element*, and that's the `<DOCUMENT>` element here:

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

Now I'll add a new element, `<GREETING>`, that encloses text content (in this case, "Hello From XML") within this XML document as follows:

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

Next, I can add a new element as well, <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>. And each of the <GREETING> and <MESSAGE> elements themselves hold text. In this way, I've created a new XML document.

There's more to the story, however—XML documents can also be *well-formed* and *valid*.

Well-Formed XML Documents

To be well-formed, an XML document must follow the syntax rules set up for XML by the W3C in the XML 1.0 recommendation (which you can find at www.w3.org/TR/REC-xml). Informally, "well-formed" means mostly that the document must contain one or more elements, and one element, the *root element*, must contain all the other elements. Also, each element must nest inside any enclosing elements properly. For example, the following 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 can also check whether it's valid. An XML document is valid if a *Document Type Declaration* (DTD) or XML *schema* is associated with it, and if the document complies with that DTD or schema. That is, the

DTD or schema specifies a set of rules for the document's own internal consistency, and if the browser can confirm that the document follows those rules, the document is valid.

XML schemas are gaining popularity, and much more support for schemas is coming in XSLT 2.0 (in fact, supporting XML schemas is the motivating force behind XSLT 2.0), but DTDs are still the most commonly used tools for ensuring validity. DTDs can be stored in a separate file, or they can be stored in the document itself, in a `<!DOCTYPE>` element. This example adds a `<!DOCTYPE>` element to the example XML document we developed:

```
<?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>
```

This book does not cover DTDs (see *Inside XML* for all the details on DTDs), but what this DTD says is 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.

You can have all kinds of hierarchies in XML documents, where one element encloses another, down to many levels deep. You can also give elements *attributes*, like this: `<CIRCLE COLOR="blue">`, where the COLOR attribute holds the value "blue." You can use such attributes to store additional data about elements. You can also include *comments* in XML documents that explain more about specific elements by enclosing comment text inside `<!--` and `-->`.

Here's an example of an XML document, planets.xml, that puts these features to work by storing data about the planets Mercury, Venus, and Earth, such as their mass, length of their day, density, distance from the sun, and so on. This document is used throughout the book, because it includes most of the XML features you'll work with in a short, compact form:

Listing 1.1 ***planets.xml***

```
<?xml version="1.0"?>
<PLANETS>
    <PLANET>
        <NAME>Mercury</NAME>
        <MASS UNITS="(Earth = 1)">.0553</MASS>
        <DAY UNITS="days">58.65</DAY>
        <RADIUS UNITS="miles">1516</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.983</DENSITY>
        <DISTANCE UNITS="million miles">43.4</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Venus</NAME>
        <MASS UNITS="(Earth = 1)">.815</MASS>
        <DAY UNITS="days">116.75</DAY>
        <RADIUS UNITS="miles">3716</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.943</DENSITY>
        <DISTANCE UNITS="million miles">66.8</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Earth</NAME>
        <MASS UNITS="(Earth = 1)">1</MASS>
        <DAY UNITS="days">1</DAY>
        <RADIUS UNITS="miles">2107</RADIUS>
        <DENSITY UNITS="(Earth = 1)">1</DENSITY>
        <DISTANCE UNITS="million miles">128.4</DISTANCE><!--At perihelion-->
    </PLANET>
</PLANETS>
```

You also need to understand a few XML definitions in this book:

- **CDATA.** Simple character data (that is, text that does not include any markup).
- **ID.** A proper XML name, which must be unique (that is, not shared by any other attribute of the ID type).
- **IDREF.** Will hold the value of an ID attribute of some element, usually another element that the current element is related to.
- **IDREFS.** Multiple IDs of elements separated by whitespace.
- **NAME Character.** A letter, digit, period, hyphen, underscore, or colon.
- **NAME.** An XML name, which must start with a letter, an underscore, or a colon, optionally followed by additional name characters.
- **NAMES.** A list of names, separated by whitespace.

- **NMTOKEN.** A token made up of one or more letters, digits, hyphens, underscores, colons, and periods.
- **NMTOKENS.** Multiple proper XML names in a list, separated by whitespace.
- **NOTATION.** A notation name (which must be declared in the DTD).
- **PCDATA.** Parsed character data. PCDATA does not include any markup, and any entity references have been expanded already in PCDATA.

That gives us an overview of XML documents, including what a well-formed and valid document is. If you don't feel you're up to speed on XML documents, read another book on the subject, such as *Inside XML*. You might also look at some of the XML resources on the Web:

- `http://www.w3c.org/xml`. The World Wide Web Consortium's main XML site, the starting point for all things XML.
- `http://www.w3.org/XML/1999/XML-in-10-points`. "XML In 10 Points" (actually only seven); an XML overview.
- `http://www.w3.org/TR/REC-xml`. This is the official W3C recommendation for XML 1.0, the current (and only) version. Not terribly easy to read.
- `http://www.w3.org/TR/xml-stylesheet/`. All about using stylesheets and XML.
- `http://www.w3.org/TR/REC-xml-names/`. All about XML namespaces.
- `http://www.w3.org/XML/Activity.html`. An overview of current XML activity at W3C.
- `http://www.w3.org/TR/xmlschema-0/`, `http://www.w3.org/TR/xmlschema-1/`, and `http://www.w3.org/TR/xmlschema-2/`. XML schemas, the alternative to DTDs.
- `http://www.w3.org/TR/xlink/`. The XLinks specification.
- `http://www.w3.org/TR/xptr`. The XPointers specification.
- `http://www.w3.org/TR/xhtml1/`. The XHTML 1.0 specification.
- `http://www.w3.org/TR/xhtml11/`. The XHTML 1.1 specification.
- `http://www.w3.org/DOM/`. The W3C Document Object Model, DOM.

So, now you've created XML documents—how can you take a look at them?

What Does XML Look Like in a Browser?

You can use a browser such as the Microsoft Internet Explorer, version 5 or later, to display raw XML documents directly. For example, if I saved the XML document we just created in a document named greeting.xml, and opened that document in the Internet Explorer, you'd see something like Figure 1.1.

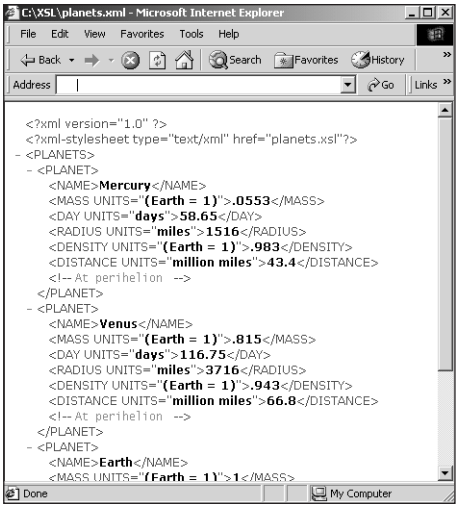

Figure 1.1 An XML document in the Internet Explorer.

You can see the complete XML document in Figure 1.1. There's no particular formatting at all; the XML document appears in the Internet Explorer just as it does when you might print it out on a printer. (In fact, the Internet Explorer default stylesheet for XML documents was used for the screen shown in Figure 1.1. The stylesheet converts XML into Dynamic HTML, which the Internet Explorer knows how to use.) But what if you want to present the data in a different way? For example, what if you want to present the data in planets.xml in an HTML document as an HTML table?

This is where XSLT transformations enter the picture. We'll take a look at them first in this chapter. At the end of this chapter, we'll take a look at the other side of XSL, XSL-FO.

XSLT Transformations

XSLT is a powerful language for manipulating the data in XML documents. For example, using an XSLT stylesheet, I'll be able to take the data in planets.xml and format that data into an HTML table. *Stylesheets* contain the rules you've set up to transform an XML document, and much of this book focuses on writing stylesheets and helping you understand how they work. Here's what the XSLT stylesheet planets.xsl, that transforms the data in planets.xml into an HTML table, looks like (we'll dissect it in Chapter 2):

Listing 1.2 ***planets.xsl***

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/PLANETS">
        <HTML>
            <HEAD>
                <TITLE>
                    The Planets Table
                </TITLE>
            </HEAD>
            <BODY>
                <H1>
                    The Planets Table
                </H1>
                <TABLE BORDER="2">
                    <TR>
                        <TD>Name</TD>
                        <TD>Mass</TD>
                        <TD>Radius</TD>
                        <TD>Day</TD>
                    </TR>
                    <xsl:apply-templates/>
                </TABLE>
            </BODY>
        </HTML>
    </xsl:template>

    <xsl:template match="PLANET">
       <TR>
          <TD><xsl:value-of select="NAME"/></TD>
          <TD><xsl:apply-templates select="MASS"/></TD>
          <TD><xsl:apply-templates select="RADIUS"/></TD>
          <TD><xsl:apply-templates select="DAY"/></TD>
       </TR>
   </xsl:template>

    <xsl:template match="MASS">
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

    <xsl:template match="RADIUS">
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>
```

continues ▸

Listing 1.2 **Continued**

```
    <xsl:template match="DAY">
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

</xsl:stylesheet>
```

You can see that this XSLT stylesheet has the look of an XML document—and for good reason, because that's exactly what it is. All XSLT stylesheets are also XML documents, and as such should be well-formed XML. You'll see these two documents—planets.xml (as given in Listing 1.1) and its associated stylesheet, planets.xsl (as given in Listing 1.2)—throughout the book as we perform XSLT transformations in many different ways.

How do you connect this stylesheet to the XML document planets.xml? As we'll see in the next chapter, one way to do that is with an `<?xml-stylesheet?>` XML processing instruction. This processing instruction uses two attributes. The first attribute is `type`, which you set to "text/xml" to indicate that you're using an XSLT stylesheet. (To use the other type of stylesheets, cascading stylesheets [CSS]—which are usually used with HTML—you'd use "text/css".) The second attribute is `href`, which you set to the URI (recall that XML uses Uniform Resource Identifiers, URIs, rather than URLs) of the stylesheet:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xml" href="planets.xsl"?>
<PLANETS>

    <PLANET>
        <NAME>Mercury</NAME>
        <MASS UNITS="(Earth = 1)">.0553</MASS>
        <DAY UNITS="days">58.65</DAY>
        <RADIUS UNITS="miles">1516</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.983</DENSITY>
        <DISTANCE UNITS="million miles">43.4</DISTANCE><!--At perihelion-->
    </PLANET>
        .
        .
        .
```

Now I can use an XSLT *processor* to apply planets.xsl to planets.xml and create a new document, planets.html. The XSLT processor creates planets.html, and you can see that new HTML document in Figure 1.2.

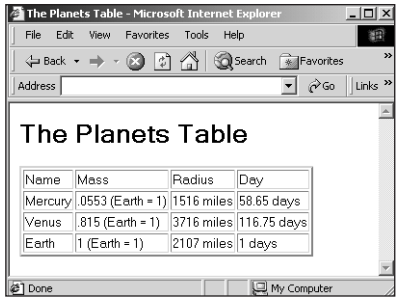

Figure 1.2 An HTML document created by an XSLT processor.

As you see in Figure 1.2, the XSLT processor read the data in planets.xml, applied the rules put into planets.xsl, and created an HTML table in planets.html. That's the first example of an XSLT transformation.

What actually happened here? You've seen the XML document, planets.xml, and the XSLT stylesheet, planets.xsl. But how did they combine to create planets.html?

Making an XSLT Transformation Happen

You use an XSLT processor to bring about an XSLT transformation, such as transforming planets.xml into planets.html. You can use XSLT in three ways to transform XML documents:

- **With standalone programs called XSLT Processors.** There are several programs, usually based on Java, that will perform XSLT transformations; we'll see a number of them in this chapter.
- **In the client.** A client program, such as a browser, can perform the transformation, reading in the stylesheet you specify with the `<?xml-stylesheet?>` processing instruction. The Internet Explorer can handle transformations this way to some extent.
- **In the server.** A server program, such as a Java servlet, can use a stylesheet to transform a document automatically and send it to the client.

We'll see all three ways of performing XSLT transformation in this book. In fact, you're going to see an overview of all these different ways of doing things right here in this chapter.

Using Standalone XSLT Processors

One of the most common ways of making XSLT transformations happen is to use standalone XSLT processors. There are plenty of such processors around, although not all can handle all possible XSLT stylesheets. To use an XSLT processor, you just run it from the command line (which means in a DOS window in Windows), giving it the name of the XML source document, the XSLT stylesheet to use, and the name of the document you want to create.

Here's a starter list of some of the available standalone XSLT processors available online, in alphabetical order—most (but not all) are free:

- **4XSLT** `http://Fourthought.com/4Suite/4XSLT`. A Python XSLT processor.
- **EZ/X** `http://www.activated.com/products/products.html`. A Java package for both XML parsing and XSLT processing.
- **iXSLT** `http://www.infoteria.com/en/contents/download/index.html`. A command-line XSLT processor.
- **Koala XSL Engine** `http://www.inria.fr/koala/XML/xslProcessor`. A Java XSLT processor using the Simple API for XML (SAX 1.0) and the Document Object Model (DOM 1.0) API.
- **LotusXSL** `http://www.alphaworks.ibm.com/tech/LotusXSL`. IBM's LotusXSL implements an XSLT processor in Java, and can interface to APIs that conform to the Document Object Model (DOM) Level 1 Specification. A famous XSLT processor, but it now appears to be superceded by Xalan 2.0.
- **MDC-XSL** `http://mdc-xsl.sourceforge.net`. An XSLT processor in C++ that can be used as a standalone program.
- **Microsoft XML Parser** `http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp`. This is the Microsoft XML parser, a high-performance parser that is available as a COM component, and can be used to implement XSLT support in applications.
- **Sablotron** `http://www.gingerall.com/charlie-bin/get/webGA/act/sablotron.act`. Sablotron is a fast, compact and portable XSLT processor. Currently supports a subset of the XSLT recommendation. You can use it with C or Perl.
- **SAXON** `http://users.iclway.co.uk/mhkay/saxon/index.html`. An XSLT processor that fully implements XSLT 1.0 and XPath 1.0, as well as a number of extensions to these specifications. Note that this release has some support for the XSLT 1.1 working draft as well.
- **Transformiix** `http://www.mozilla.org`. Transformiix is Mozilla's XSLT component, now implemented to some extent in Netscape 6.0.

- **Unicorn XSLT processor (UXT)** `http://www.unicorn-enterprises.com`. This XSLT processor supports XSLT Transformations, and is written in C++.
- **Xalan C++** `http://xml.apache.org/xalan-c/index.html`. Implementation of the W3C Recommendations for XSLT and the XML Path Language (XPath). The C++ version of the famous Apache Xalan processor.
- **Xalan Java** `http://xml.apache.org/xalan-j/index.html`. Java Implementation of the W3C Recommendations for XSLT and the XML Path Language (XPath). The Java version of the famous Apache Xalan processor. Also includes extension functions for SQL access to databases via JDBC, and much more.
- **xesalt** `http://www.inlogix.de/products.html`. This XSLT processor is available as a Web server module (for both Apache and IIS Web servers), Netscape 4.x plug-in, and command line processor.
- **XML parser for C** `http://technet.oracle.com/tech/xml/parser_c2`. Oracle's XSLT processor. Supports the XSLT 1.0 Recommendation, created for use with C.
- **XML parser for Java** `http://technet.oracle.com/tech/xml/parser_java2`. Oracle's XSLT processor. Supports the XSLT 1.0 Recommendation, created for use with Java.
- **XML parser for PL/SQL** `http://technet.oracle.com/tech/xml/parser_plsql`. Oracle's XSLT processor. Supports the XSLT 1.0 Recommendation, created for use with PL/SQL.
- **XML::XSLT** `http://xmlxslt.sourceforge.net`. This is an XSLT parser written in Perl. It implements parts of the XSLT Recommendation.
- **Xport** `http://www.timelux.lu`. An XSLT transformation processor, available as a COM object.
- **XSL:P** http://`www.clc-marketing.com/xslp/download.html`. An up-to-date XSLT processor.
- **XT** `http://www.jclark.com/xml/xt.html`. XT is a well-known implementation in Java of the XSLT Recommendation.

The following sections examine four of these XSLT processors in more detail: XT, Saxon, Oracle's XSLT processor, and Xalan. All these programs are available for free online, and can implement the XSLT examples shown in this book. If you want to follow the examples in this book, it will be useful to pick up one or more of these standalone XSLT processors (probably the best known and most widely used is Xalan). To make XSLT transformations happen, I'll use these XSLT processors throughout the book.

These processors are all Java-based, so you'll need Java installed on your system. If you don't already have Java, you can get it for free at Sun's Java site. The most recent edition, Java 2 version 1.3, is available at `http://java.sun.com/j2se/1.3`, as of this writing. All you have to do is download Java for your operating system and follow the installation instructions on the download pages.

Although you need Java to run these XSLT processors, don't panic if you're not a programmer—no programming is required. Although Chapter 10 does go into some Java programming to show you how to create XSLT transformations in code, all these processors—XT, Saxon, Oracle's XSLT processor, and Xalan—can be run from the command line.

If you are running Windows, there's an even easier way to use XT and Saxon—they both come packaged as an .exe file (xt.exe and saxon.exe) that you can run directly in Windows, and you won't need Java at all. This way of doing things is covered as well.

Using a Java XSLT Processor

To use a Java-based XSLT processor, you download it and unzip it, and it's ready to go. You should read the posted directions, of course, but typically there are just two steps to take.

First, you must let Java know how to find the XSLT processor, which is stored in a Java Archive, or JAR, file. To tell Java to search the JAR file, you set the `classpath` environment variable to the path of the JAR file. For example, in any version of Windows, you start by opening a DOS window. Then you can execute a line such as the following, which sets the `classpath` variable to the Oracle XSLT processor's JAR file, xmlparserv2.jar, which in this case is stored in the directory c:\oraclexml\lib:

```
C:\>set classpath=c:\oraclexml\lib\xmlparserv2.jar
```

Now you're ready to take the second step, which is to run the XSLT processor. This involves executing the Java *class* that supports the XSLT processor. For the Oracle XSLT processor, this is oracle.xml.parser.v2.oraxsl. In Windows, for example, you could change to the directory that held the planets.xml and planets.xsl files, and execute oracle.xml.parser.v2.oraxsl using Java this way:

```
C:\planets>java oracle.xml.parser.v2.oraxsl planets.xml planets.xsl planets.html
```

This will transform planets.xml to planets.html using planets.xsl. Note that this example assumes that java.exe, which is what runs Java, is in your Windows path. If java.exe is not in your path, you can specifically give its location, which is the Java bin directory, such as c:\jdk1.3\bin (JDK stands

for Java Development Kit, and Java 2 version 1.3 installs itself in the c:\jdk1.3 directory by default) as follows:

```
C:\planets>c:\jdk1.3\bin\java oracle.xml.parser.v2.oraxsl
➥planets.xml planets.xsl planets.html
```

In fact, you can combine the two steps (setting the `classpath` and running the XSLT processor) into one if you use `-cp` with Java to indicate what `classpath` to use:

```
C:\planets>c:\jdk1.3\bin\java -cp c:\oraclexml\lib\xmlparserv2.jar
➥oracle.xml.parser.v2.oraxsl planets.xml planets.xsl planets.html
```

These are all fairly long command lines, and at first you might feel that this is a complex way of doing things. However, there's a reason that most XSLT processors are written in Java: Java is supported on many platforms, from the Macintosh to UNIX, which means that the XSLT processor can run on all those platforms as well.

Of course, this is all a lot easier if you're running Windows and use the precompiled version of either XT (which is xt.exe) or Saxon (saxon.exe). For example, here's how to use xt.exe in Windows to perform the same transformation (this example assumes that xt.exe is in your path):

```
C:\planets>xt planets.xml planets.xsl planets.html
```

That's the process in overview; now I'll take a look at each of the four XSLT processors (XT, Saxon, Oracle's XSLT processor, and Xalan) in depth, showing exactly how to use each one. First, note two things: XML and XSL software changes very quickly, so by the time you read this, some of it might already be out of date; and although all these XSLT processors are supposed to support all standard XSLT, they give different results on some occasions.

James Clark's XT

You can get James Clark's XT at `www.jclark.com/xml/xt.html`. Besides XT itself, you'll also need an XML *parser*, which XT will use to read your XML document. The XT download also comes with sax.jar, which holds James Clark's XML parser, or you can use James Clark's XP parser, which you can get at `www.jclark.com/xml/xp/index.html`, for this purpose.

My own preference is to use the Apache Project's Xerces XML parser, which is available at `http://xml.apache.org`. (As of this writing, the current version, Xerces 1.3.0, is available at `http://xml.apache.org/dist/xerces-j/` in zipped format for UNIX as Xerces-J-bin.1.3.0.tar.gz and Windows as Xerces-J-bin.1.3.0.zip.)

XT itself is a Java application, and included in the XT download is the JAR file you'll need, xt.jar. To use xerces.jar and xt.jar, you must include

them both in your `classpath`, as shown in the following example for Windows (modify the locations of these files as needed):

```
C:\>set classpath=C:\xerces-1_3_0\xerces.jar;C:\xt\xt.jar
```

Then you can use the XT transformation class, com.jclark.xsl.sax.Driver.class. You supply the name of the parser you want to use, which in this case is org.apache.xerces.parsers.SAXParser in xerces.jar, by setting the com.jclark.xsl.sax.parser variable to that name on the command line. For example, here's how I use XT to transform planets.xml, using planets.xsl, into planets.html in Windows (assuming that c:\planets is the directory that holds planets.xml and planets.xsl, and that java.exe is in your path):

```
C:\planets>java -Dcom.jclark.xsl.sax.parser=org.apache.xerces.parsers.SAXParser
➥com.jclark.xsl.sax.Driver planets.xml planets.xsl planets.html
```

That line is quite a mouthful, so it might provide some relief to know that XT is also packaged as a Win32 executable program, xt.exe. To use xt.exe, however, you need the Microsoft Java Virtual Machine (VM) installed (which is included with the Internet Explorer). Here's an example in Windows that performs the same transformation as the preceding command, assuming xt.exe is in your path:

```
C:\planets>xt planets.xml planets.xsl planets.html
```

If xt.exe is not in your path, you can specify its location directly, like this if xt.exe is in c:\xt:

```
C:\planets>c:\xt\xt planets.xml planets.xsl planets.html
```

Saxon

Saxon by Michael Kay is one of the earliest XSLT processors, and you can get it for free at `http://users.iclway.co.uk/mhkay/saxon/`. All you have to do is download saxon.zip and unzip it, which creates the Java JAR file you need, saxon.jar.

To perform XSLT transformations, you first make sure that saxon.jar is in your `classpath`. For example, in Windows, assuming that saxon.jar is in c:\saxon, you can set the `classpath` variable this way:

```
C:\>set classpath=c:\saxon\saxon.jar
```

Now you can use com.icl.saxon.StyleSheet.class, the Saxon XSLT class, like this to perform an XSLT transformation:

```
C:\planets>java com.icl.saxon.StyleSheet planets.xml planets.xsl
```

By default, Saxon sends the resulting output to the screen, which is not what you want if you want to create the file planets.html. To create planets.html, you can use the UNIX or DOS > pipe symbol like this, which sends Saxon's output to that file:

```
C:\planets>java com.icl.saxon.StyleSheet planets.xml planets.xsl > planets.html
```

If you're running Windows, you can also use instant Saxon, which is a Win32 executable program named saxon.exe. You can download saxon.exe from `http://users.iclway.co.uk/mhkay/saxon/`, and run it in Windows like this (the `-o planets.html` part specifies the name of the output file here):

```
C:\planets>saxon -o planets.html planets.xml planets.xsl
```

Oracle XSLT

Oracle corporation also has a free XSLT processor, which you can get from `http://technet.oracle.com/tech/xml/`. You have to go through a lengthy registration process to get it, though. As of this writing, you click the XDK for Java link at `http://technet.oracle.com/tech/xml/` to get the XSLT processor.

When you unzip the download from Oracle, the JAR file you need (as of this writing) is named xmlparserv2.jar. You can put it in your `classpath` in Windows as follows:

```
C:\>set classpath=c:\oraclexml\lib\xmlparserv2.jar
```

The actual Java class you need is oracle.xml.parser.v2.oraxsl, and you can use it like this to transform planets.xml into planets.html using planets.xsl:

```
C:\planets>java oracle.xml.parser.v2.oraxsl planets.xml planets.xsl planets.html
```

Xalan

Probably the most widely used standalone XSLT processor is Xalan, from the Apache Project (Apache is a type of Web server in widespread use). You can get the Java version of Xalan at `http://xml.apache.org/xalan-j/index.html`. Just click the zipped file you want, currently xalan-j_2_0_0.zip for Windows or xalan-j_2_0_0.tar.gz for UNIX.

When you unzip the downloaded file, you get both xalan.jar, the XSLT processor, and xerces.jar, the XML parser you need. You can include both these JAR files in your classpath like this in Windows (modify the paths here as appropriate for your system):

```
C:\>set classpath=c:\xalan-j_2_0_0\bin\xalan.jar;c:\xalan-j_2_0_0\bin\xerces.jar
```

To then use planets.xsl to transform planets.xml into planets.html, execute the Java class you need, org.apache.xalan.xslt.Process, as follows:

```
C:\planets>java org.apache.xalan.xslt.Process
➥-IN planets.xml -XSL planets.xsl -OUT planets.html
```

Note that you use `-IN` to specify the name of the input file, `-OUT` to specify the name of the output file, and `-XSL` to specify the name of the XSLT stylesheet. Xalan is the XSLT processor we'll use most frequently, so here are some more details. The following list includes all the tokens you can use with the org.apache.xalan.xslt.Process class, as printed out by Xalan itself:

- `-CR` (Use carriage returns only on output—default is CR/LF)
- `-DIAG` (Output timing diagnostics)
- `-EDUMP [optional]FileName` (Do stackdump on error)
- `-HTML` (Use HTML formatter)
- `-IN inputXMLURL`
- `-INDENT` (Number of spaces to indent each level in output tree—default is 0)
- `-LF` (Use linefeeds only on output—default is CR/LF)
- `-OUT outputFileName`
- `-PARAM name value` (Set a stylesheet parameter)
- `-Q` (Quiet mode)
- `-QC` (Quiet Pattern Conflicts Warnings)
- `-TEXT` (Use simple text formatter)
- `-TG` (Trace each result tree generation event)
- `-TS` (Trace each selection event)
- `-TT` (Trace the templates as they are being called)
- `-TTC` (Trace the template children as they are being processed)
- `-V` (Version info)
- `-VALIDATE` (Validate the XML and XSL input—validation is off by default)
- `-XML` (Use XML formatter and add XML header)
- `-XSL XSLTransformationURL`

You'll see all these processors in this book, but as mentioned, probably the one I'll use most is Xalan. (The reason I use Xalan most often is because it has become the most popular XSTL processor and is the most widespread use). Of course, you can use any XSLT processor, as long as it conforms to the W3C XSLT specification.

That completes your look at standalone XSLT processors. There's another way to transform XML documents without a standalone program—you can use a client program, such as a browser, to transform documents.

Using Browsers to Transform XML Documents

Both the Microsoft Internet Explorer and Netscape Navigator include some support for XSLT. Of the two, the Internet Explorer's support is far more developed, and I'll use version 5.5 of that browser here. You can read about the Internet Explorer XSLT support at `http://msdn.microsoft.com/xml/XSLGuide/`.

Internet Explorer 5.5 and earlier does not support exact XSLT syntax by default, so we'll have to make a few modifications to planets.xml and planets.xsl. (You'll learn more about this in the next chapter. There are downloads you can install for updated XSLT support.) In fact, just as this book goes to print, Internet Explorer 6.0 has become available. When I installed and tested it, it does appear to support standard XSLT syntax (except you still must use the type "text/xsl" for stylesheets like this: `<?xml-stylesheet type="text/xsl" href="planets.xsl"?>` instead of "text/xml". If you still use IE 5.5 or earlier, you'll have to make the changes outlined here and in the next chapter. If you want to avoid all that, I suggest you upgrade to IE 6.0—it looks like that browser supports full XSLT syntax.

To use planets.xml with IE (including version 6.0), I have to convert the `type` attribute in the `<?xml-stylesheet?>` processing instruction from "text/xml" to "text/xsl" (this assumes that planets.xsl is in the same directory as planets.xml, as specified by the `href` attribute):

Listing 1.3 **Microsoft Internet Explorer Version of planets.xml**

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="planets.xsl"?>
<PLANETS>

    <PLANET>
        <NAME>Mercury</NAME>
        <MASS UNITS="(Earth = 1)">.0553</MASS>
        <DAY UNITS="days">58.65</DAY>
        <RADIUS UNITS="miles">1516</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.983</DENSITY>
        <DISTANCE UNITS="million miles">43.4</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Venus</NAME>
        <MASS UNITS="(Earth = 1)">.815</MASS>
        <DAY UNITS="days">116.75</DAY>

        <RADIUS UNITS="miles">3716</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.943</DENSITY>
```

continues ▸

Listing 1.3 **Continued**

```
        <DISTANCE UNITS="million miles">66.8</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Earth</NAME>
        <MASS UNITS="(Earth = 1)">1</MASS>
        <DAY UNITS="days">1</DAY>
        <RADIUS UNITS="miles">2107</RADIUS>
        <DENSITY UNITS="(Earth = 1)">1</DENSITY>
        <DISTANCE UNITS="million miles">128.4</DISTANCE><!--At perihelion-->
    </PLANET>

</PLANETS>
```

Now you must also convert the stylesheet planets.xsl for use in IE if you're using version 5.5 or earlier (but not version 6.0 or later—the only change you have to make is setting the `type` attribute in the `<?xml-stylesheet?>` processing instruction from "text/xml" to "text/xsl"). You'll see how to make this conversion in the next chapter; here's the new version of planets.xsl that you use:

Listing 1.4 **Microsoft Internet Explorer Version of planets.xsl**

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>
                    The Planets Table
                </TITLE>
            </HEAD>
            <BODY>
                <H1>
                    The Planets Table
                </H1>
                <TABLE BORDER="2">
                    <TR>
                        <TD>Name</TD>
                        <TD>Mass</TD>
                        <TD>Radius</TD>
                        <TD>Day</TD>
                    </TR>
                    <xsl:apply-templates/>
                </TABLE>
            </BODY>
```

```
        </HTML>
    </xsl:template>

    <xsl:template match="PLANETS">
        <xsl:apply-templates/>
    </xsl:template>

    <xsl:template match="PLANET">
       <TR>
          <TD><xsl:value-of select="NAME"/></TD>
          <TD><xsl:apply-templates select="MASS"/></TD>
          <TD><xsl:apply-templates select="RADIUS"/></TD>
          <TD><xsl:apply-templates select="DAY"/></TD>
       </TR>
   </xsl:template>

    <xsl:template match="MASS">
        <xsl:value-of select="."/>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

    <xsl:template match="RADIUS">
        <xsl:value-of select="."/>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

    <xsl:template match="DAY">
        <xsl:value-of select="."/>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

</xsl:stylesheet>
```

Now you can open planets.xml in the Internet Explorer directly, as you see in Figure 1.3.

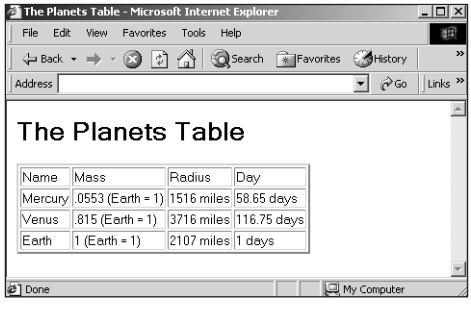

Figure 1.3 Performing an XSLT transformation in the Internet Explorer.

Although you can use XSLT with the Internet Explorer in this way, you need to modify your stylesheet to match what the Internet Explorer requires. Because the Internet Explorer does not currently support true XSLT when you open XML documents by navigating to them, I won't be using that browser to perform XSLT transformations in this book unless specifically noted. I'll use XSLT processors like Saxon and Xalan to perform transformations, and when the result is HTML, take a look at that result in the Internet Explorer.

Interestingly, there *is* a way to perform true XSLT transformations in the Internet Explorer without making any special modifications to XML or XSL documents, even if you don't download and install the latest MSXML parser (as discussed in Chapter 2)—rather than navigate to an XML document, however, you must access the XSLT processor in the Internet Explorer, MSXML3, directly, using JavaScript.

Using XSLT and JavaScript in the Internet Explorer

The XSLT processor in the Internet Explorer 5.5 is part of the MSXML3 XML parser, and if you access MSXML3 directly, using JavaScript, you don't have to modify the original planets.xml and planets.xsl (Listings 1.1 and 1.2) as you saw in the previous section. You'll see how this works in Chapter 10, but here's a Web page, xslt.html, that uses JavaScript and MSXML3 to transform planets.xml using planets.xsl and displays the results (note that you can adapt this document to use your own XML and XSLT documents without writing any JavaScript; just replace the names planets.xml and planets.xsl with the names of your XML and XSL documents):

Listing 1.5 **Microsoft Internet Explorer JavaScript Transformation**

```
<HTML>
    <HEAD>
        <TITLE>XSLT Using JavaScript</TITLE>

        <SCRIPT LANGUAGE="JavaScript">
        <!--

        function xslt()
        {
            var XMLDocument = new ActiveXObject('MSXML2.DOMDocument.3.0');
            var XSLDocument = new ActiveXObject('MSXML2.DOMDocument.3.0');
            var HTMLtarget = document.all['targetDIV'];

            XMLDocument.validateOnParse = true;
            XMLDocument.load('planets.xml');
```

```
            if (XMLDocument.parseError.errorCode != 0) {
                 HTMLtarget.innerHTML = "Error!"
                 return false;
            }

            XSLDocument.validateOnParse = true;
            XSLDocument.load('planets.xsl');
            if (XSLDocument.parseError.errorCode != 0) {
                 HTMLtarget.innerHTML = "Error!"
                 return false;
            }

            HTMLtarget.innerHTML = XMLDocument.transformNode(XSLDocument);
            return true;
        }

        //-->
        </SCRIPT>
    </HEAD>

    <BODY onload="xslt()">
        <DIV ID="targetDIV">
        </DIV>
    </BODY>
</HTML>
```

This Web page produces the same result you see in Figure 1.3, and it does so by loading planets.xml and planets.xsl directly and applying the MSXML3 parser to them. These files, planets.xml and planets.xsl, are the same as we've seen throughout this chapter, without the modifications necessary in the previous topic, where we navigated to planets.xml directly using the Internet Explorer. See Chapter 10 for more information.

Using VBScript

You can also use the Internet Explorer's other scripting language, VBScript, to achieve the same results if you're more comfortable with VBScript.

XSLT Transformations on Web Servers

You can also perform XSLT transformations on a Web server so that an XML document is transformed before the Web server sends it to a browser. The most common transformation here is to transform an XML document to HTML, but XML-to-XML transformations on the server are becoming more and more common.

Unlike the other XSLT transformations we've seen so far in this chapter, if you want to perform XSLT transformations on a Web server, you'll usually need to do some programming. There are three common ways to perform XSLT transformations on Web servers: using Java servlets, Java Server Pages (JSP), and Active Server Pages (ASP). Chapter 10 explores all three in greater detail. Some XSLT processors can be set up to be used on Web servers—here's a starter list:

- **AXSL** `www.javalobby.org/axsl.html`. AXSL is a server-side tool that converts XML to HTML using XSLT.
- **Microsoft XML Parser** `http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp`. MSXML3 provides server-safe HTTP access for use with ASP.
- **mod_xslt** `http://modxslt.userworld.com`. A simple Apache Web server module that uses XSLT to deliver XML-based content. Uses the Sablotron processor to do the XSLT processing.
- **PXSLServlet** `www.pault.com/Pxsl`. This servlet can be used to convert XML to HTML with XSLT. It also enables you to read from and write to a SQL database (JDBC).
- **xesalt** `www.inlogix.de/products.html`. This XSLT processor is available as a module for both Apache and IIS Web servers.
- **XML Enabler** `www.alphaworks.ibm.com/tech/xmlenabler`. The XML Enabler enables you to send requests to a servlet and when the servlet responds, the XML Enabler can format the data using different XSLT stylesheets.
- **XT** can be used as a Java servlet. It requires a servlet engine that implements at least version 2.1 of the Java Servlet API. The Java servlet class is com.jclark.xsl.sax.XSLServlet.

The following example shows JSP used to invoke Xalan on the Web server. Xalan converts planets.xml to planets.html, using the planets.xsl stylesheet. The code then reads in planets.html and sends it back to the browser from the Web server:

```
<%@ page errorPage="error.jsp" language="java"
    contentType="text/html" import="org.apache.xalan.xslt.*;java.io.*" %>

<%
    try
    {
        XSLTProcessor processor = XSLTProcessorFactory.getProcessor();
        processor.process(new XSLTInputSource("planets.xml"),
            new XSLTInputSource("planets.xsl"),
            new XSLTResultTarget("planets.html"));
```

```
    }
    catch(Exception e) {}

    FileReader filereader = new FileReader("planets.html");
    BufferedReader bufferedreader = new BufferedReader(filereader);
    String instring;

    while((instring = bufferedreader.readLine()) != null) { %>
        <%= instring %>
<%  }
    filereader.close();
%>
```

You can see the results in Figure 1.4, which shows planets.html as sent to the Internet Explorer from a Web server running JSP. Chapter 10 provides more information about using Java servlets, JSP, and ASP for server-side XSLT transformations.

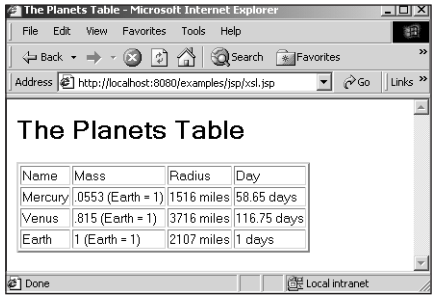

Figure 1.4 Transforming XML on the Web server.

Up to this point, you've seen how to perform XSLT transformations using standalone XSLT processors in the Internet Explorer browser and on Web servers. However, the only transformation we've done so far is to transform XML into HTML. Although that's the most popular transformation, XML to XML transformations are becoming increasingly popular.

XML-to-XML Transformations

XML-to-XML transformations are sometimes thought of as SQL for the Internet, because they enable you to use what amount to database queries on XML documents. Here's an example of what I mean. The planets.xml file we've been using has a lot of data about each planet, as you see here:

```
<?xml version="1.0"?>
<PLANETS>
    <PLANET>
        <NAME>Mercury</NAME>
```

```
        <MASS UNITS="(Earth = 1)">.0553</MASS>
        <DAY UNITS="days">58.65</DAY>
        <RADIUS UNITS="miles">1516</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.983</DENSITY>
        <DISTANCE UNITS="million miles">43.4</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Venus</NAME>
        <MASS UNITS="(Earth = 1)">.815</MASS>
        <DAY UNITS="days">116.75</DAY>
        <RADIUS UNITS="miles">3716</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.943</DENSITY>
        <DISTANCE UNITS="million miles">66.8</DISTANCE><!--At perihelion-->
    </PLANET>
        .
        .
        .
```

What if you just want a subset of that data, such as the name and mass of each planet? In database terms, planets.xml represents a table of data, and you want to create a new table holding just a subset of that data. That's what SQL can do in databases, and that's what XSLT can do with XML documents.

Here's a new version of planets.xsl that will perform the transformation we want, selecting only the name and mass of each planet, and sending that data to the output document. Note in particular that we're performing an XML-to-XML transformation, so I'm using the `<xsl:output>` element with the `method` attribute set to "xml" (in fact, the default output type is usually XML, but if an XSLT processor sees a `<html>` tag, it usually defaults to HTML):

Listing 1.6 **Selecting Name and Mass Only**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:strip-space elements="*"/>
<xsl:output method="xml" indent="yes"/>

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

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

```
    <xsl:template match="PLANET">
       <xsl:copy>
          <xsl:apply-templates/>
       </xsl:copy>
    </xsl:template>

    <xsl:template match="MASS">
        <xsl:copy>
        <xsl:value-of select="."/>
        <xsl:value-of select="@UNITS"/>
        </xsl:copy>
    </xsl:template>

    <xsl:template match="RADIUS">
    </xsl:template>

    <xsl:template match="DAY">
    </xsl:template>

    <xsl:template match="DENSITY">
    </xsl:template>

    <xsl:template match="DISTANCE">
    </xsl:template>

</xsl:stylesheet>
```

I'll apply this new version of planets.xsl to planets.xml using Xalan to create a new XML document, new.xml:

```
C:\planets>java org.apache.xalan.xslt.Process -IN planets.xml -XSL planets.xsl -OUT new.xml
```

Here's what the resulting XML document, new.xml, looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<PLANET>
    <NAME>Mercury</NAME>
    <MASS>.0553(Earth = 1)</MASS>
</PLANET>
<PLANET>
    <NAME>Venus</NAME>
    <MASS>.815(Earth = 1)</MASS>
</PLANET>
<PLANET>
    <NAME>Earth</NAME>
    <MASS>1(Earth = 1)</MASS>
</PLANET>
```

Note that this looks much like the original planets.xml, except that each <PLANET> element contains only <NAME> and <MASS> elements. In this way, we've been able to get a subset of the data in the original XML document.

You can make any number of other types of XML-to-XML transformations, of course. You can process the data in an XML document to create entirely new XML documents. For example, you can take an XML document full of student names and scores and create a new document that shows average scores. XSLT supports many built-in functions that enable you to work with data in this way, and you'll see those functions in Chapter 8.

In addition, many programs use XML to exchange data online, and they usually format their XML documents differently, so another popular use of XML-to-XML transformations on the Internet is to transform XML from the format used by one program to that used by another.

XML-to-XHTML Transformations

Although many books concentrate on XML-to-HTML transformations, the truth is that the W3C isn't overwhelmingly happy about that. They've been trying to phase out HTML (and they're the ones who standardized it originally) in favor of their new specification, XHTML, which is an XML-compliant revision of HTML. XHTML documents are also well-formed valid XML documents, so transforming from XML to XHTML is really transforming from XML to a special kind of XML.

Although the W3C is really pushing XHTML, it's not in widespread use yet. For that reason, I'll stick to HTML in this book, but because the W3C says you should use XHTML, I'll take a brief look at that here and in Chapter 6. If you want to learn more about XHTML, take a look at the W3C XHTML 1.0 recommendation at `www.w3.org/TR/xhtml1/`, as well as the XHTML 1.1 recommendation at `www.w3.org/TR/xhtml11/`.

Although the W3C says you should be converting from XML to XHTML rather than HTML, I've never seen a working example on the W3C site. The examples that they do present do not, in fact, produce valid XHTML documents. However, support for XML-to-XHTML transformations is supposed to be built into XSLT 2.0, so presumably that's coming.

I'll take a closer look at this type of transformation in Chapter 6, but here's a working version of planets.xsl that will create a valid XHTML version of planets.html. Note that this time you need to use the `doctype-public` attribute in the <xsl:output> element, and although that is correct XSLT, not all XSLT processors can handle it:

Listing 1.7 **XML-to-XHTML Transformation**

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" doctype-system
="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
    doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" indent="yes"/>

    <xsl:template match="/PLANETS">
        <html>
            <head>
                <title>
                    The Planets Table
                </title>
            </head>
            <body>
                <h1>
                    The Planets Table
                </h1>
                <table>
                    <tr>
                        <td>Name</td>
                        <td>Mass</td>
                        <td>Radius</td>
                        <td>Day</td>
                    </tr>
                    <xsl:apply-templates/>
                </table>
            </body>
        </html>
    </xsl:template>

    <xsl:template match="PLANET">
       <tr>
          <td><xsl:value-of select="NAME"/></td>
          <td><xsl:apply-templates select="MASS"/></td>
          <td><xsl:apply-templates select="RADIUS"/></td>
          <td><xsl:apply-templates select="DAY"/></td>
       </tr>
   </xsl:template>

    <xsl:template match="MASS">
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

    <xsl:template match="RADIUS">
        <xsl:value-of select="."/>
```

continues ▸

Listing 1.7 **Continued**

```
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

    <xsl:template match="DAY">
        <xsl:value-of select="."/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@UNITS"/>
    </xsl:template>

</xsl:stylesheet>
```

I'll convert planets.xml into a valid XHTML document, planets.html, using this new version of planets.xsl and the XT XSLT processor. First, I set the classpath as needed:

```
C:\>set classpath=c:xerces\xerces-1_3_0\xerces.jar;c:\xt\xt.jar;
```

Then I perform the transformation:

```
C:\planets>java - Dcom.jclark.xsl.sax.parser=org.apache.xerces.parsers.SAXParser
➥com.jclark.xsl.sax.Driver planets.xml planets.xsl planets.html
```

Here's the resulting XHTML file, planets.html:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title>
            The Planets Table
        </title>
    </head>

    <body>
        <h1>
            The Planets Table
        </h1>

        <table>
            <tr>
                <td>Name</td>
                <td>Mass</td>
                <td>Radius</td>
                <td>Day</td>
```

```
            </tr>

            <tr>
                <td>Mercury</td>
                <td>.0553 (Earth = 1)</td>
                <td>1516 miles</td>
                <td>58.65 days</td>
            </tr>

            <tr>
                <td>Venus</td>
                <td>.815 (Earth = 1)</td>
                <td>3716 miles</td>
                <td>116.75 days</td>
            </tr>

            <tr>
                <td>Earth</td>
                <td>1 (Earth = 1)</td>
                <td>2107 miles</td>
                <td>1 days</td>
            </tr>
        </table>
    </body>
</html>
```

This document, planets.html, does indeed validate as well-formed and valid transitional XHTML 1.0 (the kind of XHTML in most popular use) according to the W3C HTML and XHTML validation program. The HTML/XHTML validator tool can be found online at `http://validator.w3.org/file-upload.html`. Chapter 6 provides more information on XML-to-XHTML transformations.

So far, you've gotten a good overview of how XSLT works at this point, performing XML-to-HTML, XML, and XHTML transformations. You'll also see XML-to-RTF (Rich Text Format text), to plain text, to XSL-FO, to JavaScript, to SQL-based databases, as well as other types of XSLT transformations in this book. In addition, there's a lot more material available to you on XSLT that you should know about, and we'll now take a look at what kinds of XSLT resources you can find online.

XSLT Resources

You can find a great deal of material on XSLT online, and it's worth knowing what's out there. Note that all the following URLs are subject to change without notice—these lists are only as up to date as the people that maintain these sites allow them to be, and things can change frequently.

XSLT Specifications, Tutorials, and Examples

The starting place for XSLT resources, of course, is W3C itself. Here are the URLs for the W3C specifications that are used in this book:

- `www.w3.org/Style/XSL/`. The main W3C XSL page.
- `www.w3.org/TR/xslt`. The XSLT 1.0 specification.
- `www.w3.org/TR/xslt11`. The XSLT 1.1 working draft, which makes it easier to extend XSLT, and adds support for the W3C XBase recommendation.
- `www.w3.org/TR/xslt20req`. The XSLT 2.0 requirements, which offer a preview of XSLT 2.0, including more support for XML schemas.
- `www.w3.org/TR/xsl/`. XSL Formatting objects.
- `www.w3.org/Style/2000/xsl-charter.html`. Goals of the XSL committee.
- `www.w3.org/TR/xpath`. The XPath 1.0 recommendation.
- `www.w3.org/TR/xpath20req`. The XPath 2.0 requirements, which offer a preview of XPath 2.0, which includes more support for XSLT 2.0.
- `http://lists.w3.org/Archives/Public/www-xml-stylesheet-comments/`. The W3C list on XML stylesheets.

Many XSLT tutorials and examples are available from other sources as well; here's a starter list:

- `http://http.cs.berkeley.edu/~wilensky/CS294/xsl-examples.html`. A number of XSLT examples.
- `http://msdn.microsoft.com/xml/reference/xsl/Examples.asp`. XSLT pattern examples used in matching elements.
- `http://msdn.microsoft.com/xml/XSLGuide/xsl-overview.asp`. Getting started with XSLT.
- `www.lists.ic.ac.uk/hypermail/xml-dev/xml-dev-Nov-1999/0371.html`. PowerPoint XSLT tutorial.
- `www.mulberrytech.com/xsl/xsl-list/`. An open list dedicated to discussing XSL.
- `www.nwalsh.com/docs/tutorials/xsl/xsl/slides.html`. XSLT tutorial.
- `www.oasis-open.org/cover/xsl.html`. Coverage of what's going on in XSLT.
- `www.w3.org/Style/Activity`. Good page listing what's going on at W3C on stylesheets.
- `www.xml101.com/xsl/`. Good set of tutorials on XSLT.

- www.xslinfo.com. Good collection of XSLT resources, collected by James Tauber.
- www.zvon.org/xxl/XSLTutorial/Books/Book1/bookInOne.html. Tutorials on XSLT, XPath, XML, WML, and others.

I know of only one Usenet group on XSLT, however, and it's run by Microsoft—microsoft.public.xsl. Others will appear in time. You might also want to check out an XSL mailing list—it's at www.mulberrytech.com/xsl/xsl-list.

Besides W3C specifications, tutorials, and examples, you'll also find plenty of editors that you can use to create XSLT stylesheets online.

XSLT Editors

To create the XML and XSL documents used 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 and XSL 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 a document, you save it in your editor's plain text format.

Using WordPad

Windows text editors such as WordPad 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 actually a problem with .xml and .xsl files, because WordPad understands the extensions .xml and .xsl, but if you try to save documents that you create while working with this book with extensions that WordPad doesn't recognize, it'll add the extension .txt at the end. To avoid that, place the name of the file in quotation marks when you save it, as in "file.abc".

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

- **Adobe FrameMaker** www.adobe.com. Adobe includes great, but expensive, XML support in FrameMaker.
- **XML Pro** www.vervet.com/. Costly but powerful XML editor.
- **XML Writer**, on disk, XMLWriter http://xmlwriter.net/. Color syntax highlighting, nice interface.
- **XML Notepad** msdn.microsoft.com/xml/notepad/intro.asp. Microsoft's free XML editor—a little obscure 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** www.`xmetal.com`. An expensive but very powerful XML editor, and many authors' editor of choice.
- **XML Spy** `www.xmlspy.com/`. A good user interface and easy to use.
- **Arbortext's Epic** `www.arbortext.com/`. A powerful editor, expensive, and customizable.

You can see XML Spy at work in Figure 1.5, XML Writer in Figure 1.6, and XML Notepad in Figure 1.7.

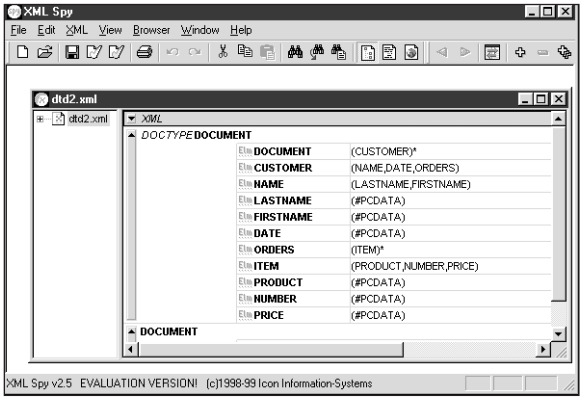

Figure 1.5 XML Spy editing XML.

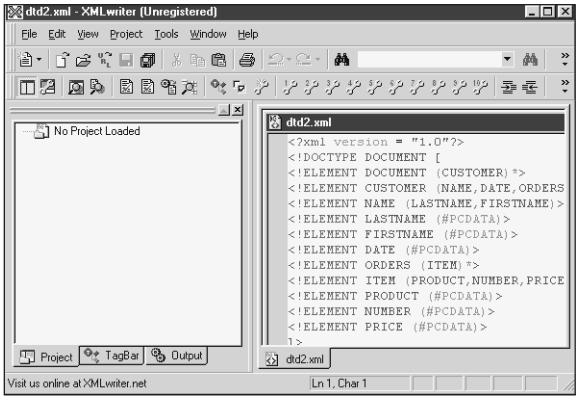

Figure 1.6 XML Writer editing XML.

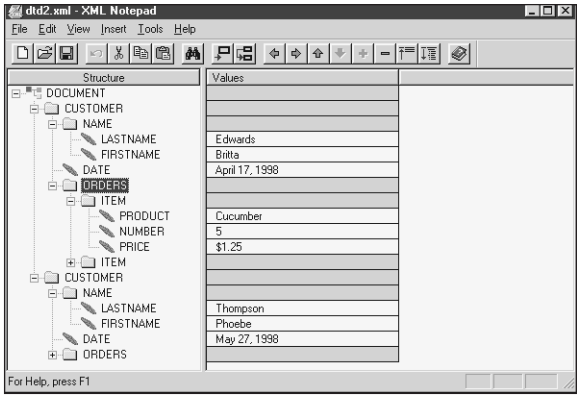

Figure 1.7 XML Notepad editing XML.

In fact, some dedicated XSLT editors are available. Here's a starter list:

- `http://lists.w3.org/Archives/Public/xsl-editors/`. A W3C list discussing XSL editors.
- **IBM XSL Editor** `www.alphaworks.ibm.com/tech/xsleditor`. Java XSLT stylesheet editor that provides a visual interface for writing stylesheets and writing select-and-match expressions. Currently, you must have Java 2 version 1.1 (not 1.2 or 1.3) installed, however.
- **Stylus** `www.exceloncorp.com/products/excelon_stylus.html`. Stylus includes an XSLT stylesheet editor.
- **Visual XML Transformation Tool** `www.alphaworks.ibm.com/aw.nsf/techmain/visualxmltools`. Visual XML Transformation Tool generates XSLT for transforming source documents into target documents for you.
- **Whitehill Composer** `www.whitehill.com/products/prod4.html`. A drag-and-drop, WYSIWYG XSLT generator of XSLT stylesheets.
- **XL-Styler** `www.seeburger.de/xml`. Includes syntax highlighting, tag completion, HTML preview, and more.
- **XML Cooktop** `http://xmleverywhere.com/cooktop/`. This one is just out, and it looks like a good one, it lets you develop and test XSLT stylesheets.
- **XML Spy** `www.xmlspy.com/`. XML Spy is an XML editor you can also use to edit XSLT.
- **XML Style Wizard** `www.infoteria.com/en/contents/download`. A tool for generating XSLT files. The wizard creates an XSLT file by examining XML data and asking the user questions.

- **xslide** `www.mulberrytech.com/xsl/xslide`. Supports an XSLT editing mode for Emacs.
- **XSpLit** `www.percussion.com/xmlzone/technology.htm`. Enables you to split HTML documents into XML DTDs and XSLT stylesheets.

XSLT Utilities

There are also many XSLT utilities available on the Web, and the following list includes some favorites:

- **Microsoft XSL ISAPI Extension** `http://msdn.microsoft.com/downloads/webtechnology/xml/xslisapi.asp`. The Microsoft XSL ISAPI Extension simplifies the task of performing server-side XSLT transformations.
- **Microsoft XSL-to-XSLT Converter** `http://msdn.microsoft.com/downloads/webtechnology/xml/xsltconv.asp`. Converts XSL into XSLT.
- **XSL Lint** `www.nwalsh.com/xsl/xslint`. XSL Lint is a syntax checker for XSLT that detects many types of errors.
- **XSL Trace** `www.alphaworks.ibm.com/tech/xsltrace`. This product enables a user to visually step through XSLT.
- **XSLT Compiler** `www.sun.com/xml/developers/xsltc`. Converts XSLT files into Java classes for transforming XML files.
- **XSLT test tool** `www.netcrucible.com/xslt/xslt-tool.htm`. This tool enables you to run XSLT with various popular processors so that you can make sure your transforms work well on all systems. It also enables you to call Microsoft's MSXML3 from the command-line like any other XSLT processor.
- **XSLTC** `www3.cybercities.com/x/xsltc`. Compiles XSLT stylesheets into C++ code. It's based on Transformiix, Mozilla's XSLT processor.
- **XSLTracer** `www.zvon.org/xxl/XSLTracer/Output/introduction.html`. XSLTracer is a Perl tool that shows how the processing of XML files with XSLT stylesheet works.

That completes your overview of XSLT in this chapter, the foundation chapter. As you can see, there's a tremendous amount of material here, waiting to be put to work in this book. The rest of this chapter provides an overview of XSL-FO.

XSL Formatting Objects: XSL-FO

The most popular part of XSL is the XSLT transformation part that you've already seen in this chapter. The other, and far larger, part is the XSL Formatting Objects part, XSL-FO.

Using XSL-FO, you can specify down to the millimeter how an XML document should be formatted and displayed. You specify everything for your documents: the text font, position, alignment, color, flow, indexing, margin size, and more. It's sort of like writing a word processor by hand, and the complexity of XSL-FO makes some people reluctant to use it. You'll learn more about what XSL-FO has to offer and how to use it in Chapters 11 and 12.

XSL-FO Resources

Some XSL-FO resources are available to you on the Web, but far fewer than those for XSLT. Here are the main ones:

- `www.w3.org/TR/xsl`. The main XSL candidate recommendation, which also includes XSL-FO.
- `http://lists.w3.org/Archives/Public/www-xsl-fo/`. A W3C list for comments on XSL-FO.

Just as there are XSLT processors out there for you to use, there are also XSL-FO processors. None comes close to implementing the whole standard, however. Here's a starter list of XSL-FO processors:

- **FOP** `http://xml.apache.org/fop`. A Java application that reads an XSL formatting object tree (which you create with an XML parser) and creates a PDF document.
- **PassiveTeX** `http://users.ox.ac.uk/~rahtz/passivetex`. A TeX package that formats XSL-FO output to PDF. Makes use of David Carlisle's xmltex XML parser.
- **SAXESS Wave** `www.saxess.com/wave/index.html`. An XML-to-Shockwave/Flash converter.
- **TeXML** `www.alphaworks.ibm.com/tech/texml`. Converts XML documents into TeX.
- **Unicorn Formatting Objects (UFO)** `www.unicorn-enterprises.com`. XSL Formatting Objects processor written in C++. It can generate output in PostScript, PDF, and other formats supported by TeX DVI drivers.
- **XEP** `http://www.renderx.com/FO2PDF.html`. A Java XSL-FO processor that converts XSL formatting objects to PDF or PostScript.

In this book, I'll use fop (formatting objects processor), which is probably the most widely used XSL-FO processor. This Java-based XSL-FO processor takes an XML document that is written to use the XSL-FO formatting objects and translates it to PDF format, which you can examine with Adobe Acrobat. Although XSLT transformations are often made to HTML, that won't work for XSL-FO, because in that case, you specify every aspect of the presentation format down to the last detail, which means that PDF format is much more appropriate.

Formatting an XML Document

To format planets.xml into planets.pdf, we can use the XSL-FO formatting objects that are introduced in Chapter 12. For example, here's how we might display the name of the first planet, Mercury, using XSL-FO formatting objects such as `flow` and `block`:

```
<fo:page-sequence master-name="page">
    <fo:flow flow-name="xsl-region-body">
<fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt" font-weight="bold">
            Mercury
        </fo:block>
        .
        .
        .
```

However, writing an entire document using the XSL formatting objects is not an easy task for any but short documents. W3C foresaw that difficulty, and that's one of the main reasons they introduced the transformation language, XSLT. In particular, you can write a stylesheet and use XSLT to transform an XML document so that it uses the XSL formatting objects.

In practice, using stylesheets is almost invariably the way such transformations are done, and it's the way we'll do things in Chapters 11 and 12. All you have to do is supply an XSLT stylesheet that can be used to convert your document to use formatting objects. In this way, an XSLT processor can do all the work for you, transforming a document from a form you're comfortable working with to formatting object form, which you can then feed to a program that can handle formatting objects and display the formatted result.

To make all this self-evident, here's an example using the XML document we've already seen in this chapter, planets.xml:

```
<?xml version="1.0"?>
<PLANETS>
    <PLANET>
        <NAME>Mercury</NAME>
```

```
        <MASS UNITS="(Earth = 1)">.0553</MASS>
        <DAY UNITS="days">58.65</DAY>
        <RADIUS UNITS="miles">1516</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.983</DENSITY>
        <DISTANCE UNITS="million miles">43.4</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Venus</NAME>
        <MASS UNITS="(Earth = 1)">.815</MASS>
        <DAY UNITS="days">116.75</DAY>
        <RADIUS UNITS="miles">3716</RADIUS>
        <DENSITY UNITS="(Earth = 1)">.943</DENSITY>
        <DISTANCE UNITS="million miles">66.8</DISTANCE><!--At perihelion-->
    </PLANET>

    <PLANET>
        <NAME>Earth</NAME>
        <MASS UNITS="(Earth = 1)">1</MASS>
        <DAY UNITS="days">1</DAY>
        <RADIUS UNITS="miles">2107</RADIUS>
        <DENSITY UNITS="(Earth = 1)">1</DENSITY>
        <DISTANCE UNITS="million miles">128.4</DISTANCE><!--At perihelion-->
    </PLANET>
</PLANETS>
```

In this example, I'll use an XSLT stylesheet—which you'll see how to create in Chapter 11—to transform planets.xml so that it uses formatting objects. Then I'll use the FOP processor to turn the new document into a PDF file. I'll also take a look at the formatted document as it appears in Adobe Acrobat.

The XSLT Stylesheet

Here's what that stylesheet, planetsPDF.xsl, looks like. This stylesheet takes the data in planets.xml and formats it in a PDF file, planets.pdf. In this case, I'll use a large font for text—36 point:

Listing 1.8 **XML to XSL-FO Transformation**

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    version='1.0'>

    <xsl:template match="PLANETS">
        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
            <fo:layout-master-set>
                 <fo:simple-page-master master-name="page"
```

continues ▸

Listing 1.8 **Continued**

```
                page-height="400mm" page-width="300mm"
                margin-top="10mm" margin-bottom="10mm"
                margin-left="20mm" margin-right="20mm">

                <fo:region-body
                    margin-top="0mm" margin-bottom="10mm"
                    margin-left="0mm" margin-right="0mm"/>

                <fo:region-after extent="10mm"/>
            </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:page-sequence master-name="page">
            <fo:flow flow-name="xsl-region-body">
                <xsl:apply-templates/>
            </fo:flow>
        </fo:page-sequence>
    </fo:root>
</xsl:template>

<xsl:template match="PLANET/NAME">
    <fo:block font-weight="bold" font-size="36pt"
        line-height="48pt" font-family="sans-serif">
        Name:
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="PLANET/MASS">
    <fo:block font-size="36pt" line-height="48pt"
        font-family="sans-serif">
        Mass (Earth = 1):
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="PLANET/DAY">
    <fo:block font-size="36pt" line-height="48pt" font-family="sans-serif">
        Day (Earth = 1):
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="PLANET/RADIUS">
    <fo:block font-size="36pt" line-height="48pt" font-family="sans-serif">
        Radius (in miles):
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

```
    <xsl:template match="PLANET/DENSITY">
        <fo:block font-size="36pt" line-height="48pt" font-family="sans-serif">
            Density (Earth = 1):
            <xsl:apply-templates/>
        </fo:block>
    </xsl:template>

    <xsl:template match="PLANET/DISTANCE">
        <fo:block font-size="36pt" line-height="48pt" font-family="sans-serif">
            Distance (million miles):
            <xsl:apply-templates/>
        </fo:block>
    </xsl:template>
</xsl:stylesheet>
```

Transforming a Document into Formatting Object Form

To transform planets.xml into a document that uses formatting objects, which I'll call planets.fo, all I have to do is apply the stylesheet planetsPDF.xsl. You can do that using the XSLT techniques you already saw in this chapter.

For example, to use Xalan to create planets.fo, you first set the `classpath` something like this in Windows:

```
C:\>set classpath=c:\xalan\xalan-j_2_0_0\bin\xalan.jar;
c:\xalan\xalan-j_2_0_0\bin\xerces.jar
```

Then you apply planetsPDF.xsl to planets.xml to produce planets.fo:

```
C:\planets>java org.apache.xalan.xslt.Process
➥-IN planets.xml -XSL planetsPDF.xsl -OUT planets.fo
```

The document planets.fo uses the XSL formatting objects to specify how the document should be formatted. Here's what planets.fo looks like:

Listing 1.9 **planets.fo**

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

<fo:layout-master-set>
<fo:simple-page-master margin-right="20mm" margin-left="20mm"
        margin-bottom="10mm" margin-top="10mm"
        page-width="300mm" page-height="400mm" master-name="page">
<fo:region-body margin-right="0mm" margin-left="0mm"
            margin-bottom="10mm" margin-top="0mm"/>
```

continues ▸

Listing 1.9 **Continued**

```
        <fo:region-after extent="10mm"/>
    </fo:simple-page-master>
</fo:layout-master-set>

<fo:page-sequence master-name="page">
    <fo:flow flow-name="xsl-region-body">
<fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt" font-weight="bold">
            Name:
            Mercury
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Mass (Earth = 1):
            .0553
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Day (Earth = 1):
            58.65
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Radius (in miles):
            1516
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Density (Earth = 1):
            .983
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Distance (million miles):
            43.4
        </fo:block>

<fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt" font-weight="bold">
            Name:
            Venus
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Mass (Earth = 1):
            .815
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Day (Earth = 1):
            116.75
```

```
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Radius (in miles):
            3716
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Density (Earth = 1):
            .943
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Distance (million miles):
            66.8
        </fo:block>

<fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt" font-weight="bold">
            Name:
            Earth
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Mass (Earth = 1):
            1
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Day (Earth = 1):
            1</fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Radius (in miles):
            2107
        </fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Density (Earth = 1):
            1</fo:block>
        <fo:block font-family="sans-serif" line-height="48pt"
            font-size="36pt">
            Distance (million miles):
            128.4
        </fo:block>
    </fo:flow>
</fo:page-sequence>

</fo:root>
```

OK, now we've created planets.fo. How can we use it to create a formatted PDF file?

Creating a Formatted Document

To process planets.fo and create a formatted document, I'll use James Tauber's fop, which has now been donated to the Apache XML Project.

The main fop page is `http://xml.apache.org/fop`, and currently, you can download fop from `http://xml.apache.org/fop/download.html`. The fop package, including documentation, comes zipped, so you have to unzip it. It's implemented as a Java JAR file, fop.jar, and I'll use fop version 0.15 here.

You can use fop from the command line with a Java class that at this writing is org.apache.fop.apps.CommandLine. You need to provide the XML parser you want to use, and I'll use the Xerces Java parser in xerces.jar (which comes with Xalan). Here's how I use fop to convert planets.fo to planets.pdf with Java in Windows; in this case, I'm specifying the `classpath` with the `-cp` switch to include xerces.jar, as well as two necessary JAR files that come with the fop download—fop.jar and w3c.jar. (This example assumes that fop.jar, xerces.jar, and w3c.jar are all in C:\planets—if not, you can specify their full paths.)

```
C:\planets>java -cp fop.jar;xerces.jar;w3c.jar
➥org.apache.fop.apps.CommandLine planets.fo planets.pdf
```

You can use the Adobe Acrobat PDF reader to see the resulting file, planets.pdf, as shown in Figure 1.8. (You can get Acrobat PDF Reader for free at `www.adobe.com/products/acrobat/readermain.html`.) The planets.xml document appears in that figure formatted as specified in the planetsPDF.xsl stylesheet.

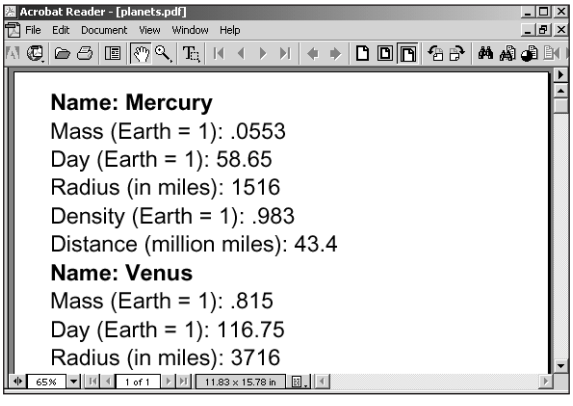

Figure 1.8 A PDF document created with formatting objects.

The PDF format is a good one for formatting object output, although it has some limitations—for example, it can't handle dynamic tables that can expand or collapse at the click of a mouse, or interactive multiple-target links, both of which are part of the formatting objects specification. Although there is little support in any major browser for XSL-FO today, it will most likely be supported in browsers one day.

That completes your overview. In this book, you're going to see all there is to XSLT, and you'll also get an introduction to XSL-FO. Now it's time to dig into XSLT, starting with the next chapter.