1

What Is XML?

This chapter provides a brief summary of what XML is. The abbreviation "XML" refers to eXtensible Markup Language, which means that XML is extensible or changeable. HTML (Hypertext Markup Language), on the contrary, is a non-extensible language and is the default language that sits behind many of the web pages in your web browser, along with numerous other languages.

HTML does not allow changes to web pages. HTML web pages are effectively frozen in time when they are built and cannot be changed when viewed in a browser.

Internet Explorer and Netscape are browsers used for viewing websites on the Internet.

XML, on the other hand, allows generation of web pages on the fly. XML allows storage of changeable data into web pages that can be altered at any time besides runtime. XML pages can also be tailored in look, feel, and content, and they can be tailored to any specific user looking at a web page at any point in time.

In this chapter you learn:

- What XML is
- What XSL is
- The differences between XML and HTML
- Basic XML syntax
- The basics of the XML DOM
- Details about different browsers and XML
- The basics of the DTD (Document Type Definition)
- How to construct an XML document
- Reserved characters in XML
- How to ignore the XML parser

- What XML namespaces are
- How to handle XML for multiple languages

Let's begin by comparing XML with HTML, the Hypertext Markup Language.

Comparing HTML and XML

XML can, in some respects, be considered an extensible form of HTML. This is because HTML is restrictive in terms of the tags it is allowed to use. In the following sample HTML document, all tags, such as `<HTML>`, are predefined:

```
<HTML>
<HEAD><TITLE>This is a simple HMTL page</TITLE></HEAD>
<BODY>

<P>
Once more unto the breach, dear friends, once more; or close the wall up with
our English dead. In peace there's nothing so becomes a man as modest stillness
and humility; but when th' blast of war blows in our ears, then imitate the action
of the tiger: stiffen the sinews, summon up the blood, disguise fair nature with
hard-favour'd rage; then lend the eye a terrible aspect.

<P>
Cry 'Havoc !' and let slip the dogs of war, that this foul deed shall smell above
the earth with carrion men, groaning for burial.
</P>

</BODY>
</HTML>
```

Figure 1-1 shows the execution of this script in a browser. You can see in the figure that none of the tags appear in the browser, only the text between the tags. In the preceding sample HTML page code, the tags are all predefined and enclosed within angle brackets (`< . . . >`). An HTML document will always begin with the tag `<HTML>` and end with the corresponding closing tag `</HTML>`. Other tags shown in the above script are `<HEAD>`, `<TITLE>`, `<BODY>`, and `<P>`. The `<P>` tag is used for paragraphs.

Unlike HTML, XML is extensible and thus is capable of being extended or modified by changing or adding features. XML can have tags of its own created (customized) that are unique to every XML document created. An XML document when embedded into an HTML page needs the predefined tag that an HTML page does, such as `<HTML>` and `<P>`, but XML can also make up its own tags as it goes along.

> *An important restriction with respect to the construction of XML documents that is not strictly applied in HTML code is that all tags must be contained within other tags. The root node tag is the only exception. Examine the previous HTML coding example and you will see that in the source code the first paragraph does not have a terminating `</P>` tag (using the / or forward slash character). HTML does not care about this. XML does!*

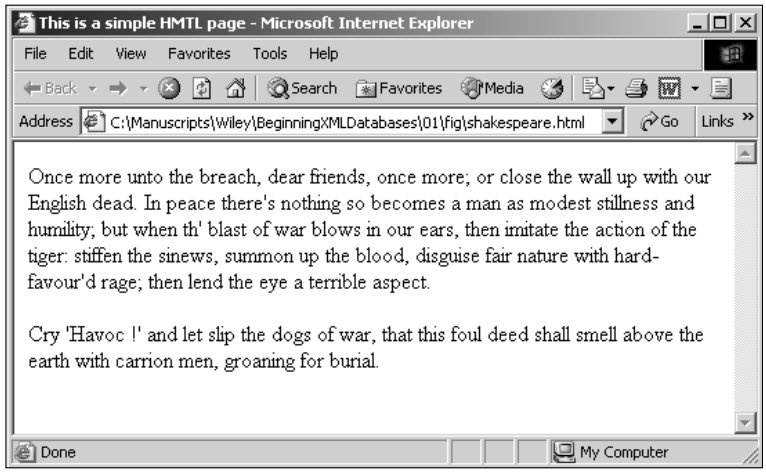


Figure 1-1: A simple sample HTML page

What Is XML Capable Of?

So, XML is not limited to a predefined set of tags as HTML is, but allows the creation of customized tags. The advantages of using XML could loosely be stated as follows:

- **Flexibility with data:** Any information can be placed into an XML page. The XML page becomes the data rather than the definitional container for data, as shown in Figure 1-1.
- **Web page integration:** This becomes easier because building those web pages becomes more generic. Web pages are data driven (based on content in the web page) rather than relying on the definition of the tags (programming language–driven) and where the tags are placed.
- **Open standards:** XML is completely flexible. No single software company can control and define what tags are created, what each tag means, and where in a document tags should appear. XML is a little like a completely generic programming language.
- **Enhanced scalability and compression:** When sending web pages over the Internet, XML pages can contain just data. All the coded programming tags required for HTML are not needed.
- **Irrelevant order of data:** The order in which data appears in an XML page is unimportant because it is data. Data can have things applied to it at the client site (in a browser) to change it, if you use something like eXtensible Style Sheets (XSL).

What Is XSL?

XSL is a formatting language that applies templating to consistent data repetitions inside XML documents. For example, an XML page containing a listing of clients and their addresses could be formatted into a nice looking table field using an XSL style sheet that showed each different client on a single row in the table, as shown in Figure 1-2.

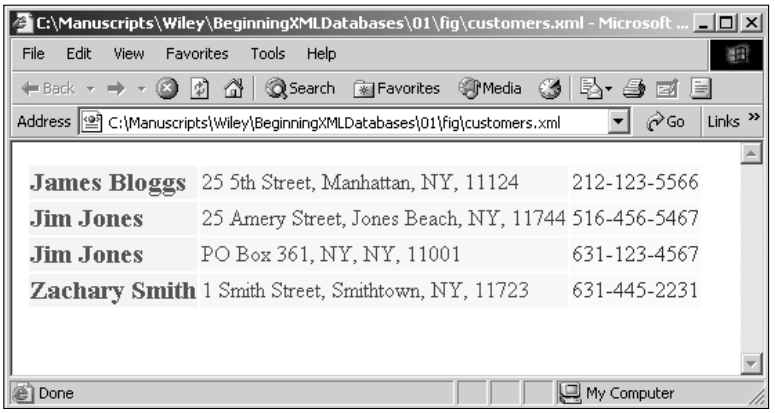


Figure 1-2: XSL can be used to apply templates to XML documents.

The HTML equivalent of XSL is cascading style sheets (CSS).

Creating and Displaying a Simple XML Document

Following is a sample XML document. The only required predefined tag is on the first line, which describes that the version of the XML parser used is version 1.0:

```
<?xmlversion="1.0"?>
<WeatherForecast date="2/1/2004">
   <city>
      <name>Frankfurt</name>
      <temperature>
      <min>43</min>
         <max>52</max>
      </temperature>
   </city>
   <city>
      <name>London</name>
      <temperature>
         <min>31</min>
         <max>45</max>
      </temperature>
   </city>
   <city>
      <name>Paris</name>
      <temperature>
         <min>620</min>
         <max>74</max>
      </temperature>
   </city>
</WeatherForecast>
```

A parser is a program that analyzes and verifies the syntax of the coding of a programming language. An XML- capable browser parses XML code to ensure that is syntactically correct. As already mentioned, one parser function is to ensure that all starting and ending tags exist, and that there is no interlocking of XML tags within the document. Interlocking implies that a new tag of the same type, such as `<city>`*, cannot be started, until the ending tag of the previous city (*`</city>`*), has been found.*

In a browser, the XML document looks as shown in Figure 1-3. The callouts in Figure 1-3 show that in addition to being flexible for a web pages programmer, XML is even flexible to the end user. End users are unlikely to see an XML document in this raw state, but Figure 1-3 helps to demonstrate the flexibility of XML.

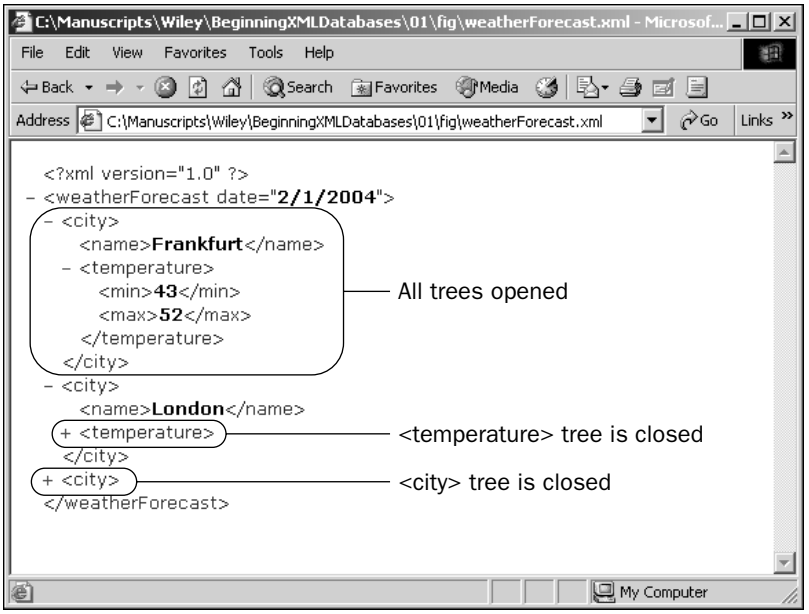


Figure 1-3: A simple sample XML page

The primary purpose of HTML is for display of data. XML is intended to describe data. XML is the data and thus describes itself. When HTML pages contain data, they must be explicitly generated. For every web page weather report written in HTML, a new HTML page must be created. This includes both the weather report data and all HTML tags. When regenerating an XML-based weather report, only the data is regenerated. Any templates using something like XSL remain the same. And those templates are probably only downloaded once. The result is that XML occupies less network bandwidth and involves less processing power.

XML is also a very capable medium for bulk data transfers that are platform and database independent. This is because XML is a universal standard. In short, XML does not do as much processing as HTML does. XML is structure applied to data. Effectively XML complements HTML rather than replaces it. XML was built to store and exchange data; HTML is designed to display data. XSL, on the other hand, is designed to format data.

Try It Out Creating a Simple XML Document

The data shown below represents three regions containing six countries:

```
Africa          Zambia
Africa          Zimbabwe
Asia            Burma
Australasia     Australia
Caribbean       Bahamas
Caribbean       Barbados
```

Here, you are going to create a single hierarchy XML document. The example shown in Figure 1-3, and its preceding matching XML data, gives you an example to base this task on.

Create the XML document as follows:

1. Use an appropriate editor to create the XML document text file (Notepad in Windows).
2. Create the XML tag:

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

3. Create the root tag first. The data is divided up as countries listed within continents (regions). Countries are contained within regions. There are multiple regions so there has to be a tag, which is a parent tag of the multiple regions. If there was a single region there could be a single `<region>` tag as the root node. So create a root node such as `<regions>`, indicating multiple regions. The XML document now looks something like this:

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

4. Now add each region in as a child of the `<regions>` tag. It should look something like this:

```
<?xml version="1.0"?>
<regions>
   <region>Africa</region>
   <region>Asia</region>
   <region>Australasia</region>
   <region>Caribbean</region>
</regions>
```

5. Next you can add the individual countries into their respective regions by creating individual `<country>` tags:

```
<?xml version="1.0"?>
<regions>
   <region>Africa</region>
      <country>Zambia</country>
      <country>Zimbabwe</country>
```

```
    <region>Asia</region>
       <country>Burma</country>
    <region>Australasia</region>
       <country>Australia</country>
    <region>Caribbean</region>
       <country>Bahamas</country>
       <country>Barbados</country>
</regions>
```

6. When executed in a browser, the result will look as shown in Figure 1-4.

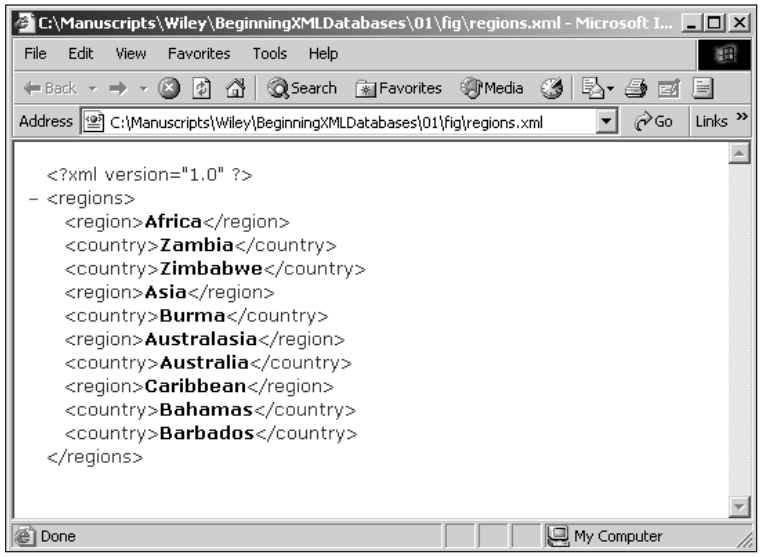


Figure 1-4: Creating a simple XML document

How It Works

You opened a text editor and created an XML document file. The XML document begins with the XML tag, identifying the version of XML is use. Next you added the root node called `<regions>`. All XML documents must have a single root node. Next you added four `<region>` nodes representing four regions into the root node. Next you added countries into the four different regions. Last, you viewed the XML document in your browser.

Embedding XML in HTML Pages (Data Islands)

XML documents can also be displayed in a browser using an XML data island. An *XML data island* is an XML document (with its data) directly or indirectly embedded inside an HTML page. An XML document can be embedded inline inside an HTML page using the HTML `<XML>` tag. It can also be referenced with an HTML `SRC` attribute.

This first example uses the XML tag to embed XML document data within an HTML page:

```
<HTML><BODY>

<XML ID="xmlParts">
   <?xml version="1.0" ?>
   <parts>
      <part>
         <partnumber>X12334-125</partnumber>
         <description>Oil Filter</description>
         <quantity>$24.99</quantity>
      </part>
      <part>
         <partnumber>X44562-001</partnumber>
         <description>Brake Hose</description>
         <quantity>$22.45</quantity>
      </part>
      <part>
         <partnumber>Y00023-12A</partnumber>
         <description>Transmission</description>
         <quantity>$8000.00</quantity>
      </part>
   </parts>
</XML>

<TABLE DATASRC=#xmlParts>
<TR>
   <TD><DIV DATAFLD="partnumber"></DIV></TD>
   <TD><DIV DATAFLD="$text"></DIV></TD>
</TR>
</TABLE>

</BODY></HTML>
```

HTML and XML tags can have attributes or descriptive values. In the HTML code `<IMG SRC= "image.jpg" BORDER="1">` *the tag is an* `<IMG>` *or image tag for referencing an image. The* `SRC` *attribute tells the HTML* `<IMG>` *tag where to find the image, and the* `BORDER` *tag tells HTML to put a "1" pixel wide border around the image.*

The second example allows a reference to a separate XML file using the `SRC` attribute of the XML tag.

The XML source file is stored externally to the HTML page. In this case, the parts.xml file is stored in the operating system and not stored within the HTML file as in the previous example:

```
<HTML><BODY>

<XML ID="xmlParts" SRC="parts.xml"></XML>

<TABLE DATASRC=#xmlParts>
   <TR><TD><DIV DATAFLD="partnumber"></DIV></TD>
      <TD><DIV DATAFLD="$text"></DIV></TD>
   </TR>
</TABLE>

</BODY></HTML>
```

Both of these examples look as the screen does in Figure 1-5.

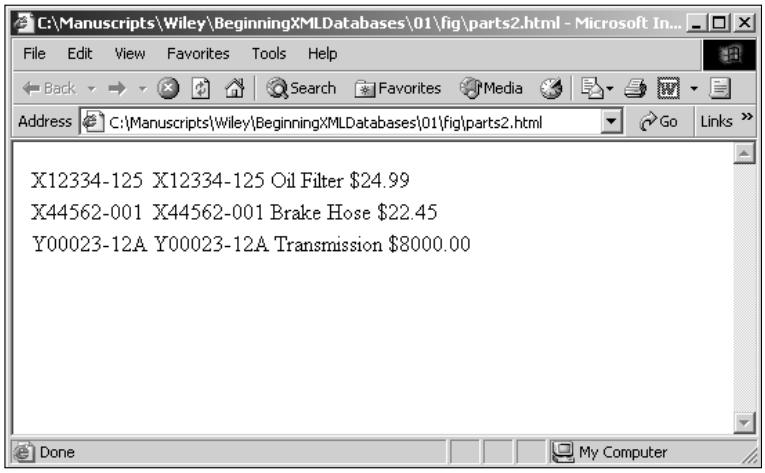


Figure 1-5: Using the XML tag to embed XML data islands into an HTML page

There are always different ways to do things.

Try It Out XML Data Islands

The XML document that follows represents the three regions and six countries created in the Try It Out exercise presented earlier in this chapter:

```
<?xml version="1.0"?>
<regions>
   <region>Africa</region>
      <country>Zambia</country>
      <country>Zimbabwe</country>
   <region>Asia</region>
      <country>Burma</country>
   <region>Australasia</region>
      <country>Australia</country>
   <region>Caribbean</region>
      <country>Bahamas</country>
      <country>Barbados</country>
</regions>
```

Here we will create a simple HTML page, containing the preceding XML document as a data island. Assume that the XML document is called countries.xml. Don't worry about a full path name. The example shown in Figure 1-5 and its preceding matching XML data island HTML pages give you an example to base this task on.

Create the HTML page as follows:

1. Use an appropriate editor to create a text file.

2. Begin by creating the <HTML> tags for the start and end of the HTML page:

```
<HTML>
</HTML>
```

3. You could add a <HEAD> tag, allowing inclusion of a title into the browser. Begin by creating the <HTML> tags for the start and end of the HTML page:

```
<HTML>
<HEAD><TITLE>Regions and Countries</TITLE></HEAD>
</HTML>
```

4. Add the body section for the HTML page by enclosing it between the <BODY> tags:

```
<HTML>
<HEAD><TITLE>Regions and Countries</TITLE></HEAD>
<BODY>
</BODY>
</HTML>
```

5. Now add the <XML> tag into the body of the HTML page, which references the externally stored XML document:

```
<HTML>
<HEAD><TITLE>Regions and Countries</TITLE></HEAD>
<BODY>
<XML ID="xmlCountries" SRC="countries.xml"></XML>
</BODY>
</HTML>
```

6. Add a table field (<TABLE> tag) to the HTML page. The table field references the <XML> tag, by the ID attribute, as shown in the code that follows. The SRC in the <XML> tag allows direct access from the HTML page to XML tags as stored in the countries.xml file. In other words, the countries.xml file is referenced from the HTML page as a referenced data island:

```
<HTML>
<HEAD><TITLE>Regions and Countries</TITLE></HEAD>
<BODY>

<XML ID="xmlCountries" SRC="countries.xml"></XML>

<TABLE DATASRC=#xmlCountries>
<TR>
   <TD><DIV DATAFLD="region"></DIV></TD>
   <TD><DIV DATAFLD="$text"></DIV></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

7. The result will look as shown in Figure 1-6, when executed in a browser.

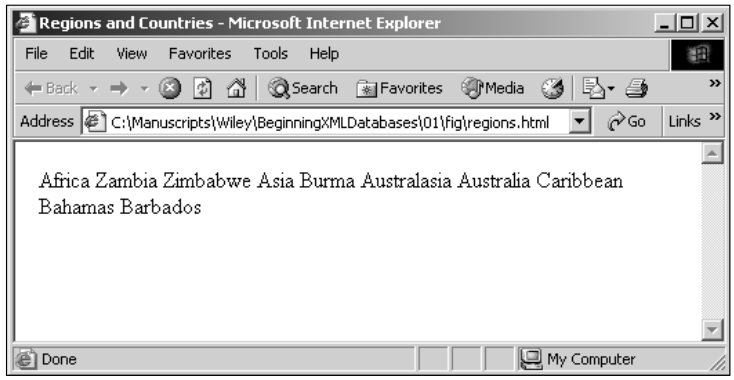


Figure 1-6: Creating a simple HTML page containing an XML data island

How It Works

You created an HTML page that referenced an XML document from the HTML page as a data island. The data island is referenced from the HTML page, to the XML document, using the XML tag as defined in the HTML page. Data is scrolled through in the HTML page using an HTML table field, using the `DATASRC` attribute of the HTML `<TABLE>` tag.

Introducing the XML Document Object Model

Another factor when using XML is that built into the browser used to display XML data, is a structure behind the XML data set. Look again at Figure 1-3 and you should see that everything is very neatly structured into a hierarchy. This entire structure can be accessed programmatically using something called the Document Object Model, or XML DOM. Using the XML DOM a programmer can find, read, and even change anything within an XML document. Those changes can also be made in two fundamental ways:

- **Explicit data access:** A program can access an XML document explicitly. For example, one can find a particular city by using the <city> tag and the name of the city.
- **Dynamic or generic access:** A program can access an XML document regardless of its data content by using the structure of the document. In other words, a program can scroll through all the tags and the data no matter what it is. That is what the XML DOM allows. An XML page can be a list of cities, weather reports, or even part numbers for an automobile manufacturer. The data set is somewhat irrelevant because the XML DOM allows direct access to the program within the browser, which displays the XML data on the screen, as shown in Figure 1-3. In other words, a program can find all the tags by passing up and down the tree of the XML DOM.

A browser uses the XML DOM to build a picture of an XML document, as shown in Figure 1-3. The browser contains a parser that does not care what the data is, but rather how data is constructed. In other words, the DOM contains a multiple dimensional (hierarchical) array structure. That array structure allows access to all tags and all data, without the programmer having to know the contents of the tags within it and even the names of the tags. An XML document is just data and so any data can be contained within it.

When creating weather reports for people in different parts of the world, the underlying templates that make the web pages look nice are all exactly the same; only the data is different. This is where this book comes into being. Data stored in databases as traditional relation tables can be used to create XML documents that can also be stored in a database. The XML DOM allows programmatic access into XML documents stored in a database. In other words, you can create XML documents, stuff them in a database, and then use database software to access the documents either as a whole or in part using the XML DOM.

That is really what this book is about. It is, however, necessary to explain certain facets of XML before we get to the meat of databases and XML. You need to have a basic picture of things such as XML and XSL first.

XML Browsers and Different Internet Browsers

There are varying degrees of support for XML in different Internet browsers. In general the latest versions of Internet Explorer or Netscape will do nicely. Using an older version of a software tool can sometimes be asking for trouble.

Using a non-mainstream browser might also be limited in scope but this is unlikely if you use the latest version. There are, however, some very specific technologies used by specific vendors. Microsoft's Internet Explorer falls into this category. Then again, Internet Explorer is probably now the most widely used browser. So, for browser-based examples, I've used Microsoft technology.

Database technology being used in this book will primarily be Oracle Database from Oracle Corporation and SQL-Server Database from Microsoft. Once again, bear in mind that the focus of this book is on using XML as a database, or in other databases.

The Document Type Definition

The Document Type Definition (DTD) is a method of defining consistent structure across all XML documents within a company, an installation, and so on. In other words, it allows validation of XML documents, ensuring that standards are adhered to even for XML data where the source of the XML data is external to the company.

From an XML in databases perspective, DTD could provide a method of structural validation, which is of course very important to any kind of database structure. However, it could also be superfluous and simply get in the way. It may depend on how XML documents are created or generated as being sources of both metadata and data. If XML documents are manually created then something like DTD could be very useful. Of course, once data is created, it is possible that only one round of validation is required for at least static data.

Static data in a database is data that does not change very often, if at all. In a database containing customers and invoices, your customers are relatively static (their names don't change — at least not very often). Transactional or dynamic data such as invoices is likely to change frequently. However, it is extremely likely that any creation of XML documents would be automatically generated by application programs. Why validate with the DTD when applications generating data (XML documents) will do that validation for you?

The DTD will be covered in a later chapter in detail, where you will deal with schemas and XML Schemas. XML Schemas are a more advanced form of the DTD. XML Schemas can be used to define what and how everything is to be created in an XML document.

XML Syntax

The basic syntax rules of XML are simple but also very strict. This section goes through those basic syntax rules one by one:

- **The XML tag:** The first line in an XML document declares the XML version in use:

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

- **Including style sheets:** The optional second line contains a style sheet reference, if a style sheet is in use:

```
<?xml:stylesheet type="text/xsl" href="cities.xsl"?>
```

- **The root node:** The next line will contain the root node of the XML document tree structure. The root node contains all other nodes in the XML document, either directly or indirectly (through child nodes):

```
<root>
```

- **A single root node:** An XML document must have a single root tag, such that all other tags are contained within that root tag. All subsequent elements must be contained within the root tag, each nested within its parent tag.

An XML tag is usually called an element.

- **The ending root tag:** The last line will contain the ending element for the root element. All ending elements have exactly the same name as their corresponding starting elements, except that the name of the node is preceded by a forward slash (/):

```
</root>
```

- **Opening and closing elements:** All XML elements must have a closing element. Omitting a closing element will cause an error. Exceptions to this rule is the XML definitional element at the beginning of the document, declaring the version of XML in exceptions, and an optional style sheet:

```
<root>
   <branch_1>
      <leaf_1>
      </leaf_1>
   </branch_1>
   <branch_2>
   </branch_2>
</root>
```

HTML tags do not always require a closing tag. Examine the first HTML code example in this chapter in the section "Comparing HTML and XML." The first paragraph does not have a `</P>` *paragraph end tag. The second paragraph does have a* `</P>` *paragraph eng tag. Some closing tags in HTML are optional, meaning that a closing tag can be included or not.*

- **Case sensitive:** XML elements are case sensitive. HTML tags are not case sensitive. The XML element <root> in the previous example is completely different than the XML element <Root> in the next example. The following example is completely different than the previous XML document shown in the previous point. Even though all the elements are the same, their case is different for the `<Root>` and `<BRANCH_1>` elements:

```
<Root>
   <BRANCH_1>
      <leaf_1>
      </leaf_1>
   </BRANCH_1>
   <branch_2>
   </branch_2>
</Root>
```

HTML does not require proper nesting of elements, such as in this example:

```
<FONT COLOR="red"><B><I>This is bold italic text in red</FONT></B></I>
```

XML on the other hand, produces an error using the preceding code. For example, in XML the following code is invalid because `</tag2>` should appear before `</tag1>`:

```
<tag1><tag2>some tags</tag1></tag2>
```

- **Element attributes:** Like HTML tags, XML elements can have attributes. An element attribute refines the aspects of an element. Attributes and their values are called name-value pairs. An XML element can have one or more name-value pairs, and the value must always be quoted. HTML attribute values do not always have to be quoted, although it is advisable. In the following XML document sample (the complete document is not shown here), populations for continents (including the name of the continent) are contained as attributes of the `<continent>` element. In other words, the continent of Africa had a population of 748,927,000 people in 1998 (748 million people where the population in thousands is the total divided by 1,000, or 748,927).

It follows that projected populations for the African continent are 1.3 billion (1,298,311) for the year 2025, and 1.8 billion (1,766,082) for the year 2050. Also in this example, the name of the country is stored in the XML document as an attribute of the `<country>` element:

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="791202 fig0105.xsl" ?>
<populationInThousands>
   <world>
      <continents>
         <continent name="Africa" year1998="748,927" year2025="1,298,311"
year2050="1,766,082">
            <countries>
               <country name="Burundi">
                  <year1998>6457</year1998>
                  <year2025>11569</year2025>
                  <year2050>15571</year2050>
               </country>
               <country name="Comoros">
                  <year1998>658</year1998>
                  <year2025>1176</year2025>
                  <year2050>1577</year2050>
               </country>
               ...
            </country>
            ...
         </continent>
         ...
      </continents>
      ...
   </world>
   ...
</populationInThousands>
```

XML element and attribute names can have space characters included in those names, as in the `<continent>` *element shown in the preceding sample XML document.*

- As shown in Figure 1-7, the previous sample XML document does include a style sheet, making the XML document display with only the names of continents and countries.
- And here is an HTML equivalent of the XML document for the previous example — as shown in Figure 1-7. Notice how much more raw code there is for each population region and country:

```
<HTML><BODY>
<TABLE CELLPADDING="2" CELLSPACING="0" BORDER="1">
   <TR>
      <TH BGCOLOR="silver">Continent
      <TH BGCOLOR="silver">Country
      <TH BGCOLOR="silver">1998
      <TH BGCOLOR="silver">2025
      <TH BGCOLOR="silver">2050
   </TR>
```

```
      <TR ALIGN="right">
         <TD BGCOLOR="#D0FFFF" ALIGN=left>Africa</TD>
         <TD BGCOLOR=#FFFFD0> </TD>
         <TD>748,927</TD>
         <TD>1,298,311</TD>
         <TD>1,766,082</TD>
      </TR>
      <TR ALIGN="right">
         <TD BGCOLOR="#D0FFFF"> </TD>
         <TD BGCOLOR=#FFFFD0 ALIGN=left>Burundi</TD>
         <TD>6,457</TD>
         <TD>11,569</TD>
         <TD>11,571</TD>
      </TR>
      <TR ALIGN="right">
         <TD BGCOLOR="#D0FFFF"> </TD>
         <TD BGCOLOR=#FFFFD0 ALIGN=left>Comoros</TD>
         <TD>658</TD>
         <TD>1,176</TD>
         <TD>1,577</TD>
      </TR>
      ...
   </TABLE>
   </BODY></HTML>
```

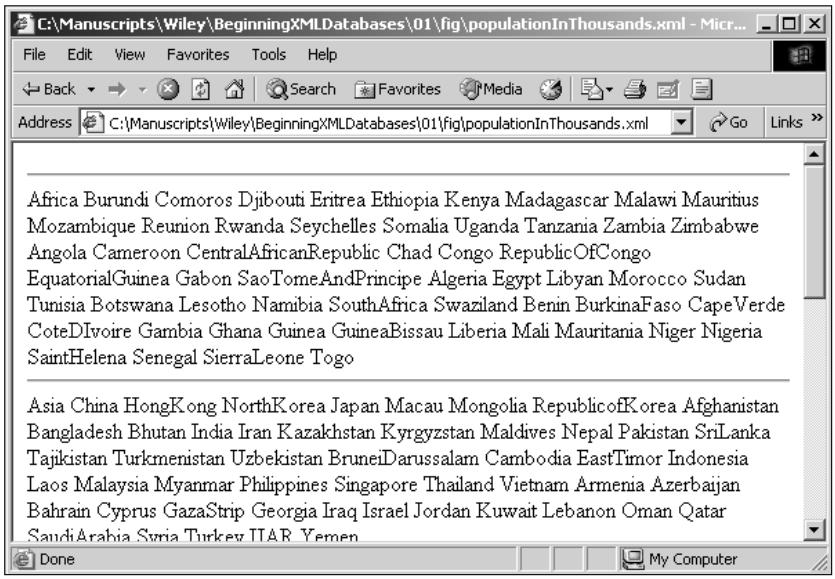
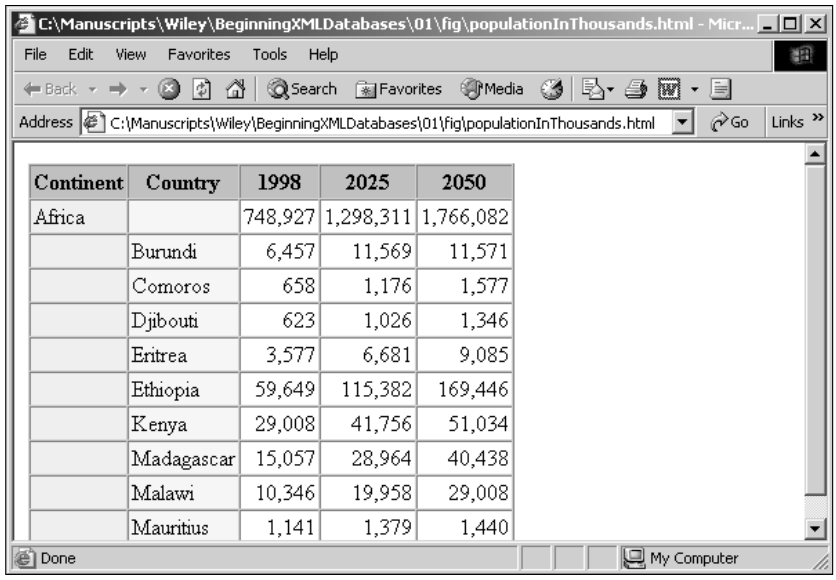


Figure 1-7: Using XML element attributes to change the display of an XML document

Figure 1-8 shows the HTML display of the preceding HTML coded page, and the XML displayed document in Figure 1-7 (the previous example).



Continent	Country	1998	2025	2050
Africa		748,927	1,298,311	1,766,082
	Burundi	6,457	11,569	11,571
	Comoros	658	1,176	1,577
	Djibouti	623	1,026	1,346
	Eritrea	3,577	6,681	9,085
	Ethiopia	59,649	115,382	169,446
	Kenya	29,008	41,756	51,034
	Madagascar	15,057	28,964	40,438
	Malawi	10,346	19,958	29,008
	Mauritius	1,141	1,379	1,440

Figure 1-8: HTML embeds the code and is less flexible than XML.

- **Comments**: Both XML and HTML use the same character strings to indicate commented out code:

```
<!-- This is a comment and will not be processed by the HTML or XML parser -->
```

Elements

As you have already seen in the previous section, an XML element is the equivalent of an HTML tag. A few rules apply explicitly to elements:

- **Element naming rules:** The names of elements (XML tags) can contain all alphanumeric characters as long as the name of the element does not begin with a number or a punctuation character. Also, names cannot contain any spaces. XML delimits between element names and attributes using a space character. Do not begin an element name with any combination of the letters XML, in any combination of uppercase or lowercase characters. In other words, `XML_1`, `xml_1`, `xML_1`, and so on, are all not allowed. It will not produce an error to use multiple operative characters, such as + (addition) and – (subtraction), but their use is inadvisable. Elements least likely to cause any problems are those containing only letters and numbers. Stay away from odd characters.

- **Relationships between elements:** The root node has only children. All other nodes have one parent node, as well as zero or more child nodes. Nodes can have elements that are related on the same hierarchical level. In the code example that follows, the following apply:
  - The root node element is called `<root>`.
  - The root node has two child node elements: `<branch_1>` and `<branch_2>`.
  - The node `<branch_1>` has one child element called `<leaf_1_1>`.
  - The node `<branch_2>` has three child elements called `<leaf_2_1>`, `<leaf_2_2>`, and `<leaf_2_3>`.
  - The nodes `<leaf_2_1>`, `<leaf_2_2>`, and `<leaf_2_3>` are all siblings, having the same parent node element in common (node `<branch_2>`):

```
<root>
   <branch_1>
      <leaf_1_1>
      </leaf_1_1>
   </branch_1>
   <branch_2 name="branch two">
      <leaf_2_1>
      </leaf_2_1>
      <leaf_2_2>
      </leaf_2_2>
      <leaf_2_3>This is a leaf</leaf_2_3>
   </branch_2>
</root>
```

- **The content of elements:** XML elements can have simple content (text only), attributes for the element concerned, and can contain other child elements. Node `<branch_2>` in the preceding example has an attribute called name (with a value of *branch two*). The node `<leaf_1>` contains nothing. The node `<leaf_2_3>` contains the text string *This is a leaf*.
- **Extensible elements:** XML documents can be altered without necessarily altering what is delivered by an application. Examine Figure 1-7. The following is the XSL code used to apply the reduced template for get the result shown in Figure 1-7:

```
   <xsl:template>
      <xsl:apply-templates select="@*"/>
         <xsl:if test="@name[.='Africa']"><HR/></xsl:if>
         <xsl:if test="@name[.='Asia']"><HR/></xsl:if>
         <xsl:if test="@name[.='Europe']"><HR/></xsl:if>
         <xsl:if test="@name[.='Latin America and the Caribbean']"><HR/></xsl:if>
         <xsl:if test="@name[.='North America']"><HR/></xsl:if>
         <xsl:if test="@name[.='Oceania']"><HR/></xsl:if>
         <xsl:value-of select="@name"/>
      <xsl:apply-templates/>
   </xsl:template>
</xsl:stylesheet>
```

Looking at the preceding XSL script, yes, we have not as yet covered anything about eXtensible Style Sheets (XSL). The point to note is that the boldface text in the preceding code finds only the name attribute values, from all elements, ignoring everything else. Therefore all population

numbers are discarded and only the names of continents and countries are returned. It is almost as if the XML document might as well look like that shown next, with all population numbers removed. The result in Figure 1-7 will still be exactly the same:

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="791202 fig0107.xsl" ?>

<populationInThousands>
   <world>
      <continents>
         <continent name="Africa">
            <countries>
               <country name="Burundi"></country>
               <country name="Comoros"></country>
            </countries>
            ...
         </continent>
      </continents>
   </world>
</populationInThousands>
```

Attributes

Elements can have attributes. An element is allowed to have zero or more attributes that describe it. Attributes are often used when the attribute is not part of the textual data set of an XML document, or when not using attributes is simply awkward. Store data as individual elements and metadata as attributes.

> *Metadata is the data about the data. In a database environment the data is the names of your customers and the invoices you send them. The metadata is the tables you define which are used to store records in customer and invoice tables. In the case of XML and HTML metadata is the tags or elements (< . . . > . . . < . . . >) contained within a web page. The values between the tags is the actual data.*

Once again, the now familiar population example:

```
<populationInThousands>
   <world>
      <continents>
         <continent name="Africa" year1998="748,927" year2025="1,298,311"
year2050="1,766,082">
            <countries>
               <country name="Burundi">
                  <year1998>6457</year1998>
                  <year2025>11569</year2025>
                  <year2050>15571</year2050>
               </country>
```

```
                <country name="Comoros">
                   <year1998>658</year1998>
                   <year2025>1176</year2025>
                   <year2050>1577</year2050>
                </country>
                ...
             </countries>
             ...
          </continent>
          ...
       </continents>
       ...
    </world>
    ...
</populationInThousands>
```

Attributes can also be contained within an element as child elements. The example you just saw can be altered as in the next script, removing all element attributes. The following script just looks busier and perhaps a little more complex for the naked eye to decipher. The more important point to note is that the physical size of the XML document is larger because additional termination elements are introduced. In very large XML documents this can be a significant performance factor:

```
<populationInThousands>
   <world>
      <continents>
         <continent>
            <name>Africa</name>
            <year1998>748,927</year1998>
            <year2025>1,298,311</year2025>
            <year2050>1,766,082</year2050>
            <countries>
               <country>
                  <name>Burundi</name>
                  <year1998>6457</year1998>
                  <year2025>11569</year2025>
                  <year2050>15571</year2050>
               </country>
               <country>
                  <name>Comoros</name>
                  <year1998>658</year1998>
                  <year2025>1176</year2025>
                  <year2050>1577</year2050>
               </country>
               ...
            </countries>
            ...
         </continent>
         ...
      </continents>
      ...
```

```
    </world>
    ...
</populationInThousands>
```

From a purely programming perspective, it could be stated that attributes should not be used because of the following reasons:

- Elements help to define structure and attributes do not.
- Attributes are not allowed to have multiple values whereas elements can.
- Programming is more complex using attributes.
- Attributes are more difficult to alter in XML documents at a later stage.

As already stated, the preceding reasons are all sensible from a purely programming perspective. From a database perspective, and XML in databases, the preceding points need some refinement and perhaps even some contradiction:

- **Elements define structure and attributes do not.** I prefer not to put too much structure into data, particularly in a database environment because the overall architecture of data can become too complex to manage and maintain, both for administrators and the database software engine. Performance can become completely disastrous if a database gets large because there is simply too much structure to deal with.
- **Attributes are not allowed multiple values.** If attributes need to have multiple values then those attributes should probably become child elements anyway. This book is after all about XML databases (and XML in databases). Therefore it makes sense to say that an attribute with multiple values is effectively a one-to-many relationship.

  You send many invoices to your customers. There is a one-to-many relationship between each customer and all of their respective invoices. A one-to-many relationship is also known as a master-detail relationship. In this case the customer is the master, and the invoices are the detail structural element. The many sides of this relationship are also known as a collection, or even an array, in object methodology parlance.

- **Attributes make programming more complex.** Programming is more complex when accessing attributes because code has to select specific values. Converting attributes to multiple contained elements allows programming to scan through array or collection structures. Once again, performance should always be considered as a factor. Scrolling through a multitude of elements contained within an array or collection is much less efficient than searching for exact attributes, which are within exact elements. It is much faster to find a single piece of data, rather than searching through lots of elements, when you do not even know if the element exists or not. An XML document can contain an element, which can be empty, or the element can simply not exist at all. From a database performance perspective, avoiding use of attributes in favor of contained, unreferenced collections (which are what a multitude of same named elements is)/is suicidal for your applications if your database gets even to a reasonable size. It will just be too slow.

- **Attributes are not expansion friendly.** It is more difficult to change metadata than it is to change data. It should be. If you have to change metadata then there might be data structural design issues anyway. In a purely database environment (not using XML), changing the database model is the equivalent of changing metadata. In commercial environments metadata is usually not altered because it is too difficult and too expensive. All application code depends on database structure not being changed. Changing database metadata requires application changes as well. That's why it can get expensive. From a perspective of XML and XML in databases, you do not want to change attributes because attributes represent metadata, and that is a database modeling design issue — not a programming issue. Changing the data is much, much easier.

Try It Out Using XML Syntax

The following data represents three regions, containing six countries, as in the previous Try It Out sections in this chapter. In this example, currencies are now added:

```
Africa          Zambia          Kwacha
Africa          Zimbabwe        Zimbabwe Dollars
Asia            Burma
Australasia     Australia       Dollars
Caribbean       Bahamas         Dollars
Caribbean       Barbados        Dollars
```

In this example, you use what you have learned about the difference between XML document elements and attributes.

The following script is the XML document created in the first Try It Out section in this chapter:

```
<?xml version="1.0"?>
<regions>
   <region>Africa</region>
      <country>Zambia</country>
      <country>Zimbabwe</country>
   <region>Asia</region>
      <country>Burma</country>
   <region>Australasia</region>
      <country>Australia</country>
   <region>Caribbean</region>
      <country>Bahamas</country>
      <country>Barbados</country>
</regions>
```

You will use the preceding XML document and add the currencies for each country. Do not create any new elements in this XML document.

Change the XML document as follows:

1. Open the XML document. You can copy the existing XML text into a new text file if you want.

2. All you do is add an attribute name-value pair to each opening <country> tag:

```
<country currency="Kwacha">Zambia</country>
```

3. The final XML document looks something like this:

```
<?xml version="1.0"?>
<regions>
   <region>Africa</region>
      <country currency="Kwacha">Zambia</country>
      <country currency="Zimbabwe Dollars">Zimbabwe</country>
   <region>Asia</region>
      <country>Burma</country>
   <region>Australasia</region>
      <country currency="Dollars">Australia</country>
   <region>Caribbean</region>
      <country currency="Dollars">Bahamas</country>
      <country currency="Dollars">Barbados</country>
</regions>
```

4. Figure 1-9 shows the result when executed in a browser.

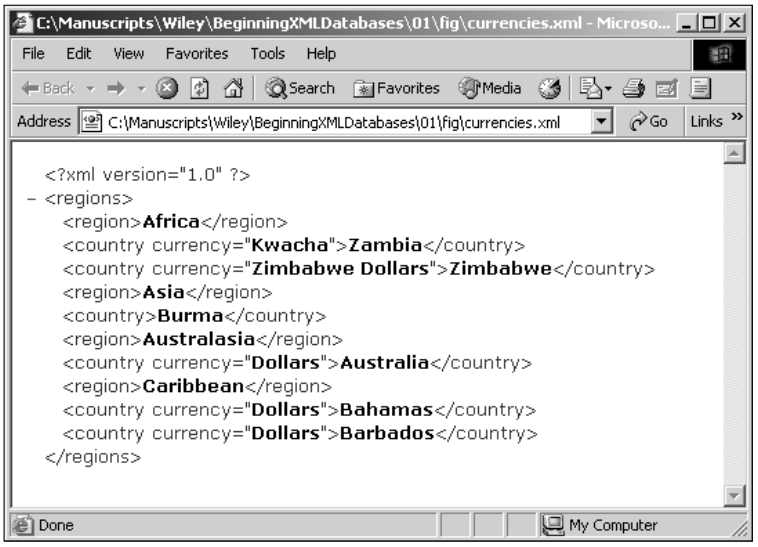


Figure 1-9: Adding attributes to elements in an XML document

How It Works

All you did was to edit an XML document containing the XML tag, a single root node, and various regions of the world that contained some of their respective countries. You then proceeded to add currency attributes into some of the countries.

Reserved Characters in XML

Escape characters are characters preventing execution in a programming language or parser. Thus the < and > characters must be escaped (using an escape sequence) if they are used in an XML document anywhere other than delimiting tags (elements). In XML, an escape sequence is a sequence of characters known to the XML parser to represent special characters. This escape sequence is exactly the same as that used by HTML. The following XML code is invalid:

```
<country name="Germany">West < East</country>
```

The preceding code can be resolved into XML by replacing the < character with the escape sequence string `<` as follows:

```
<country name="Germany">West &lt; East</country>
```

The <, >, and & characters are illegal in XML and will be interpreted. Quotation characters of all forms are best avoided and best replaced with an escape sequence.

Ignoring the XML Parser with CDATA

There is a special section in an XML document called the `CDATA` section. The XML parser ignores anything within the `CDATA` section. So no errors or syntax checking will be performed in the `CDATA` section. The `CDATA` section can be used to include scripts written in other languages such as JavaScript. The `CDATA` section is the equivalent of a `<SCRIPT> . . . </SCRIPT>` tag enclosed section in an HTML page. The `CDATA` section begins and ends with the strings, as highlighted in the following script example:

```
<SCRIPT>
<![CDATA[
function F_To_C
{
   return ((F - 32) * (5 / 9))
}
]]>
</SCRIPT>
```

What Are XML Namespaces?

Two different XML documents containing elements with the same name, where those names have different meanings, could cause conflict. This XML document contains weather forecasts for three different cities. The `<name>` element represents the name of each city:

```
<?xml version="1.0"?>
<WeatherForecast date="2/1/2004">
   <city>
      <name>Frankfurt</name>
      <temperature><min>43</min><max>52</max></temperature>
   </city>
```

```
   <city>
      <name>London</name>
      <temperature><min>31</min><max>45</max></temperature>
   </city>
   <city>
      <name>Paris</name>
      <temperature><min>20</min><max>74</max></temperature>
   </city>
</WeatherForecast>
```

This next XML document also contains `<name>` elements but those names are of countries and not of cities. Adding these two XML documents together could cause a semantic (meaning) conflict between the `<name>` elements in the two separate XML documents:

```
<?xml version="1.0"?>
<WeatherForecast date="2/1/2004">
   <country>
      <name>Germany</name>
      <temperature><min>22</min><max>45</max></temperature>
   </country>
   <country>
      <name>England</name>
      <temperature><min>24</min><max>39</max></temperature>
   </country>
   <country>
      <name>France</name>
      <temperature><min>22</min><max>85</max></temperature>
   </country>
</WeatherForecast>
```

Namespaces can be used to resolve this type of conflict by assigning a separate prefix to each XML document, adding the prefix to tags in each XML document as follows for the XML document containing cities:

```
<?xml version="1.0"?>
<i:WeatherForecastxmlns:i="http://www.mywebsite.com/nsforcities" date="2/1/2004">
   <i:city>
      <i:name>Frankfurt</i:name>
      <i:temperature><i:min>43</i:min><i:max>52</i:max></i:temperature>
   </i:city>
   <i:city>
      <i:name>London</i:name>
      <i:temperature><i:min>31</i:min><i:max>45</i:max></i:temperature>
   </i:city>
   <i:city>
      <i:name>Paris</i:name>
      <i:temperature><i:min>20</i:min><i:max>74</i:max></i:temperature>
   </i:city>
</i:WeatherForecast>
```

And for the XML document containing countries, you use a different prefix:

```
<?xml version="1.0"?>
<o:WeatherForecastxmlns:o="http://www.mywebsite.com/nsforcities" date="2/1/2004">
   <o:city>
      <o:name>Frankfurt</o:name>
      <o:temperature><o:min>43</o:min><o:max>52</o:max></o:temperature>
   </o:city>
   <o:city>
      <o:name>London</o:name>
      <o:temperature><o:min>31</o:min><o:max>45</o:max></o:temperature>
   </o:city>
   <o:city>
      <o:name>Paris</o:name>
      <o:temperature><o:min>20</o:min><o:max>74</o:max></o:temperature>
   </o:city>
</o:WeatherForecast>
```

Creating the preceding XML documents using prefixes has actually created separate elements in separate documents. This is done by using an attribute and a URL. Also when using a namespace, you don't have to assign the prefix to every child element, only the parent node concerned. So with the first XML document previously listed you can do this:

```
<?xml version="1.0"?>
<WeatherForecast xmlns:i="http://www.mywebsite.com/nsforcities" date="2/1/2004">
 <city>
    <name>Frankfurt</name>
    <temperature><min>43</min><max>52</max></temperature>
</city>
 <city>
    <name>London</name>
    <temperature><min>31</min><max>45</max></temperature>
 </city>
 <city>
   <name>Paris</name>
   <temperature><min>20</min><max>74</max></temperature>
 </city>
</WeatherForecast>
```

You could also use a namespace for the weather forecast for the countries.

XML in Many Languages

Storing XML documents in a language other than English requires some characters not used in the English language. These characters are encoded if not stored in Unicode. Notepad allows you to store text files, in this case XML documents, in Unicode. In Notepad on Win2K, select the Encoding option under the Save As menu option.

When reloading the XML document in a browser you simply have to alter the XML tag at the beginning of the script, to indicate that an encoding other than the default is used. Win2K (SP3) Notepad will allow storage as ANSI (the default), Unicode, Unicode big endian, and UTF-8. To allow the XML parser in a browser to interpret the contents of an XML document stored as UTF-8 change the XML tag as follows:

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

Summary

In this chapter you learned that:

- HTML is the Hypertext Markup Language and its set of tags is predetermined.
- XML is the eXtensible Markup Language.
- XML is extensible because its metadata (set of tags) is completely dynamic and can be extended.
- XSL stands for eXtensible Style Sheets.
- XSL allows for consistent formatting to be applied to repeated groups stored in XML documents.
- XML namespaces allow for the making of distinctions between different XML documents that have the same elements.
- XML can utilize character sets of different languages by using Unicode character sets.
- The XML DOM (Dynamic Object Model) allows run-time (dynamic) access to XML web pages.
- Different browsers and browser versions will behave differently with XML.
- For examples in this book, I've used Microsoft Internet Explorer version 6.0, running in Win2K (Windows 2000).
- The DTD (Document Type Definition) allows enforcement of structure across XML documents.

This chapter has given you a brief picture of what XML is, including a comparison with HTML and a brief summary of XSL. HTML creates web pages with fixed data and metadata. XML allows creation of web pages with adaptable data and metadata content.

The next chapter examines the XML DOM or the Document Object Model for XML. The XML DOM, like the HTML DOM, allows dynamic (run-time) access to both the data and metadata in a web page.

Exercise

1. Which line in this HTML script contains an error?

```
1. <HTML>
2. <HEAD><TITLE>Title</TITLE></HEAD>
3. <BODY>
4. <P>This is a paragraph.
5. <P>This another paragraph.</P>
6. </BODY></HTML>
```

a. 1

b. 3

c. 4

d. 5

e. None of the above

2. How many errors are present in this XML script?

```
<?xml>
<customers>
   <customer>
      <name>Zachary Smith</name>
      <address>
         <street>1 Smith Street</street>
         <town>Smithtown</town>
         <state>NY</state>
         <zip>11723</zip>
      </address>
      <phone>631-445-2231</phone>
   </customer>
   <customer>
</customers>
```

3. What kind of a web page is this?

```
<HTML><BODY>
<XML ID="xmlParts" SRC="cookies.xml"></XML>
<TABLE DATASRC=#xmlCookies>
   <TR>
      <TD><DIV DATAFLD="cookietype"></DIV></TD>
      <TD><DIV DATAFLD="$cookiedescription"></DIV></TD>
   </TR>
</TABLE>
</BODY></HTML>
```

4. What does XSL do for XML?

a. Allows changes to data in XML pages at run-time

b. Allows changes to metadata in XML pages at run-time

c. Allows regeneration of entire XML pages at run-time

d. All of the above

e. None of the above