

CHAPTER

14

EXTENSIBLE STYLESHEET LANGUAGE (XSL)

In this chapter, you will:

- ◆ Study Extensible Stylesheet Language (XSL)
- ◆ Transform Extensible Markup Language (XML) data using XSL Transformations (XSLT)
- ◆ Work with XSLT templates
- ◆ Use XSLT to manipulate transformed data

In Chapter 1, you learned the basics of Extensible Markup Language (XML). While XML is primarily a way of defining and organizing data, it does not include any of the display capabilities of Extensible Hypertext Markup Language (XHTML). However, because XML is fast becoming the standard method of transmitting data across the Internet, there will be times when you will want to display XML data as a formatted Web page. In this chapter, you will learn how to use Extensible Stylesheet Language (XSL) to format and display XML data as Web pages.



Before you begin working through this chapter, you may find it useful to review the XML information in Chapter 1.

628 Chapter 14 Extensible Stylesheet Language (XSL)

EXTENSIBLE STYLESHEET LANGUAGE (XSL)

You can create formatted Web pages using XML and **Extensible Stylesheet Language (XSL)**, which is a stylesheet language for XML. Think of XSL as being roughly equal to the Cascading Style Sheets (CSS) you use with XHTML documents, although XSL is much more complex than CSS. XSL does not just format XML data so it can be displayed in a Web browser; it also extracts and transforms specific data from an XML document. A **transformation** refers to the conversion of XML data into another type of document. To understand what this means, examine the following XML code, which organizes the 2002 Winter Olympics medal counts by country:

```
<?xml version="1.0" encoding="iso-8859-1"
standalone="yes"?>
<olympics>
  <year>2002 Winter Olympics</year>
  <medals>Olympic Medal Counts</medals>
  <country name="Germany">
    <gold>12</gold>
    <silver>16</silver>
    <bronze>7</bronze>
  </country>
  <country name="USA">
    <gold>10</gold>
    <silver>13</silver>
    <bronze>11</bronze>
  </country>
  <country name="Norway">
    <gold>11</gold>
    <silver>7</silver>
    <bronze>6</bronze>
  </country>
</olympics>
```

If you were to open the preceding document in a Web browser, the document would simply be displayed as XML data, as shown in Figure 14-1.

But what if you want to display the Olympic medals XML data as a Web page, not just as the XML data shown in Figure 14-1? Your choices are to re-create a new XHTML document from scratch or use XSL to transform the data into an XHTML document. The following XHTML document shows an example of how the Olympic medals XML data may appear after you transform it with XSL. The elements in the body of the new document are created using data extracted from the XML document. You will learn how to perform this type of transformation later in the chapter. The important thing to understand is that you can use XSL to convert the preceding XML document into the following XHTML document, which can then be displayed as the formatted Web page shown in Figure 14-2.

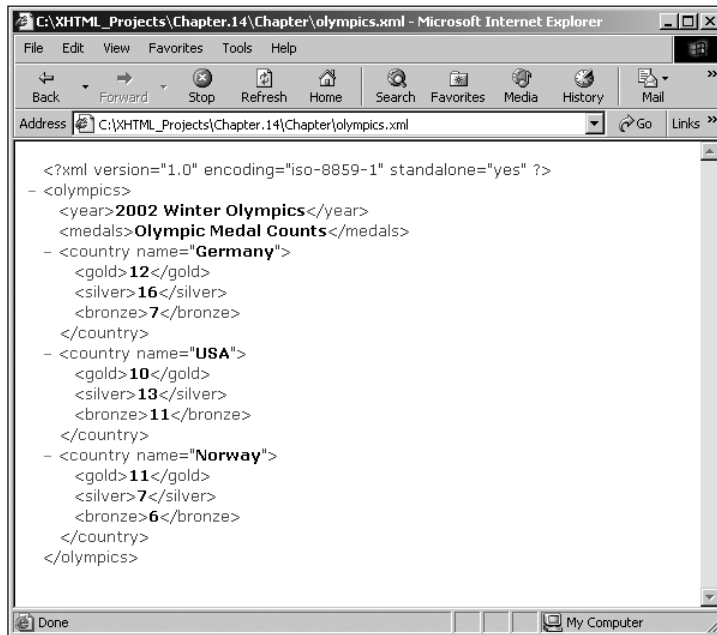


Figure 14-1 Olympic medals XML data in a Web browser

```

<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>2002 Winter Olympics</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
<style type="text/css">
body { font-family: Verdana, Arial, sans-serif }
h1 { font-size: 1.5em; color: navy; background-color:
transparent }
h2 { font-size: 1.2em; color: navy; background-color:
transparent }
p, td, th { font-size: .8em; color: olive;
background-color: transparent }
</style>
</head>
<body>
<h1>2002 Winter Olympics</h1>
<h2>Olympic Medal Counts</h2>
<table width="100%" border="1">
  <colgroup span="1" align="left" />
  <colgroup span="3" align="center" />
  <tr>
    <th>Country</th>
    <th>Gold</th>

```

630 Chapter 14 Extensible Stylesheet Language (XSL)

```

        <th>Silver</th>
        <th>Bronze</th>
    </tr>
    <tr>
        <td>Germany</td>
        <td>12</td>
        <td>16</td>
        <td>7</td>
    </tr>
    <tr>
        <td>USA</td>
        <td>10</td>
        <td>13</td>
        <td>11</td>
    </tr>
    <tr>
        <td>Norway</td>
        <td>11</td>
        <td>7</td>
        <td>6</td>
    </tr>
</table>
</body>
</html>

```

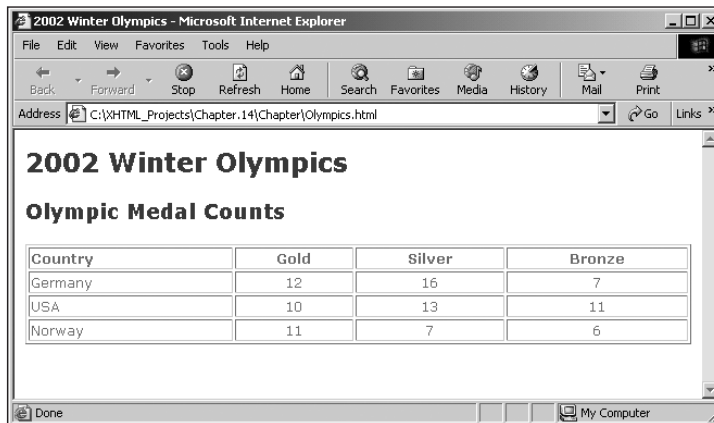


Figure 14-2 XHTML document created using XML and XSL



XHTML documents are really just specialized XML documents; unlike standard XML documents, XHTML documents contain code that allows them to be displayed in a Web browser. As you have seen, to transform a standard XML document into an XHTML document, you use XSL. In fact, this type of transformation is one of the more common uses of XSL.

XSL can be much more complicated than the previous example suggests. With XSL you can extract only the elements or attributes that you want from an XML document using conditional expressions, math operations, and other types of operations that you would normally see in a programming language such as JavaScript.

Although XSL is much more powerful than CSS, XML and XSL will not replace XHTML or CSS anytime in the near future. This is because XSL is primarily designed for adding complex formatting to XML so it can be used with a variety of user agents, including Personal Digital Assistants (PDAs) and mobile phones, as well as Web browsers. If you have an XML document from which you want to extract, display, and format data, then you should use XSL. However, if your goal is simply to design Web pages, then you should stick with XHTML and CSS.

The Parts of XSL

XSL is actually a combination of the following parts:

- XSLT
- XML Path Language (XPath)
- XSL Formatting Objects (XSL-FO)

XSL Transformations (XSLT) is a language that transforms one XML document into another XML document. For example, a Web site for an online merchant, such as a travel company, may frequently receive XML documents that list travel specials. Instead of manually creating new XHTML to display this information, the company could use XSLT to format the XML data so it can be displayed as a Web page. XSLT is considered the most important part of XSL because it is within an XSLT style sheet that you specify the rules for transforming XML data into a new XML document.

XML Path Language (XPath) is a language this is used in XSLT to access or refer to the parts of an XML document. Essentially, XPath allows you to select the elements and attributes that you want XSLT to include in a transformed document. XPath also allows you to manipulate the values that will be added to a transformed XML document. For example, XPath includes a **sum()** function that you can use to add the values of numbers stored within designated elements and attributes, and then include the result in the transformed XML document. For example, with the travel Web site you might use XML documents to store reservation information. You could use the XPath **sum()** function to add the total cost of a travel reservation such as airfare, car rental, and hotel information. The transformed document with the total trip cost could then be displayed to the traveler as a formatted Web page.



XPath is also used with XPointers, which identify locations within XML documents.

632 Chapter 14 Extensible Stylesheet Language (XSL)

XSL Formatting Objects (XSL-FO) is a language that determines how an XML document should be displayed. XSL-FO uses XSLT to transform an XML document into an XSL-FO document, which is essentially an XHTML document that is formatted using XSL-FO instead of CSS. XSL-FO was created as a way for formatting long and complex documents, such as those found in the publishing or technical writing industries. At the time of this writing, XSL-FO is not supported by any major browsers. Therefore, there is little reason to study the language at this point. Even if XSL-FO is widely supported, you will probably have little reason to work with it, unless you work on long and complex documentation, such as engineering specifications or books that will be commercially published. Instead, you should continue using CSS to format your documents. CSS is often used with XSL to format XML documents that have been transformed using XSLT.



This chapter only covers the basics of XSL, primarily the use of XSLT. For more information, see the World Wide Web Consortium's (W3C's) XSL page at <http://www.w3.org/Style/XSL/>.

Extensible Markup Language (XML) Processors

You need to use an XML processor to transform a document. An **XML processor** is an application that builds a new XML document by reading a source XML document and applying the rules in an associated XSLT style sheet. There are various standalone XML processing programs, both commercial and free, that you can use to transform an XML document. Two of the more popular free XML processors are the Apache XML Project's Xalan and SAXON, written by Michael Kay. You can download Xalan at <http://xml.apache.org/xalan-j/> and SAXON at <http://saxon.sourceforge.net/>.

Recent Web browsers also include built-in XML processors. When you open an XML document in a Web browser that includes a built-in XML processor, the XML processor automatically builds a new XML document by reading the source XML document and applying the rules in an associated XSLT style sheet. The XML processor for Internet Explorer is called the MSXML Parser. Microsoft claims that MSXML Parser version 3.0, which shipped with Internet Explorer version 6, is 100% compatible with the W3C's XSL recommendation. Other browsers, including Netscape version 6, partially support the W3C's XML recommendation. For this reason, you should use Internet Explorer version 6 or higher for the exercises you create in this chapter.



For more information on MSXML Parser, including information on where you can download the most recent version, visit <http://msdn.microsoft.com/xml/general/xmlparser.asp>.

XSL TRANSFORMATIONS (XSLT)

The structure of an XML document is arranged in a hierarchical tree. Each element and attribute in an XML document tree is referred to as a **node**. The document's root element is referred to as the root node. Individual nodes in an XML document tree can contain other nodes; similar to the way a folder on a hard drive can contain subfolders. An XML document tree is really just another way of looking at nested elements. However, the tree structure is important to understand because it is critical to how XSLT transforms one XML document into another. Figure 14-3 shows a conceptual example of the document tree for the Olympic Medals XML document you saw earlier.

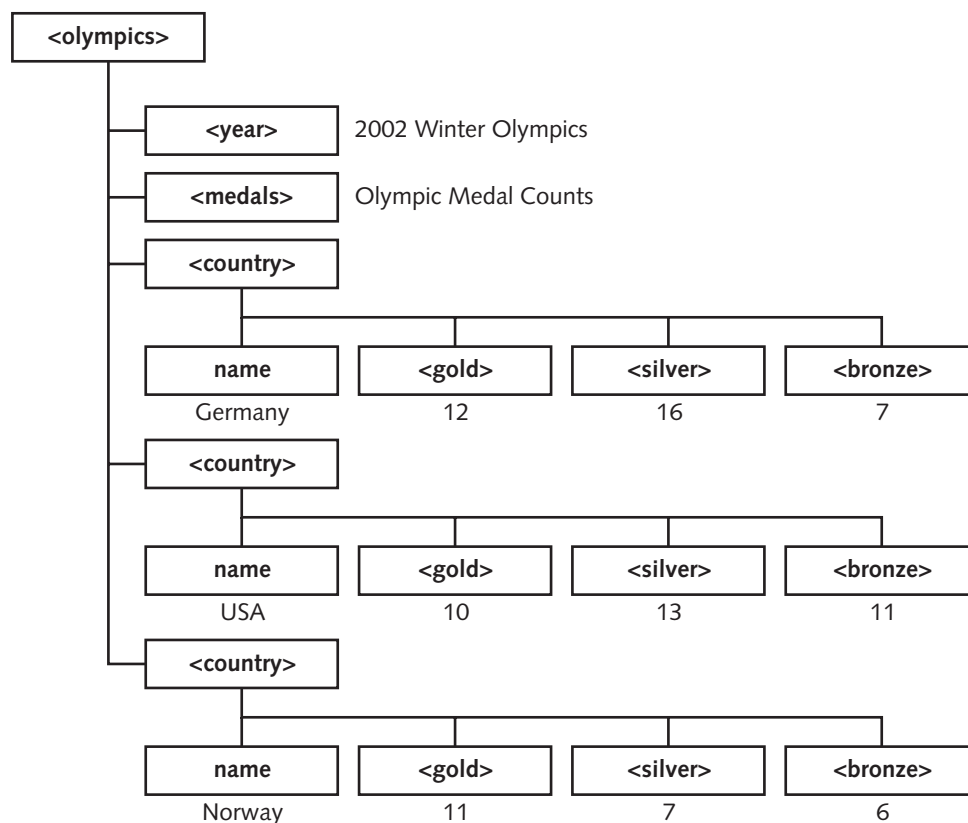


Figure 14-3 Olympic Medals XML document tree

During the transformation process, the XML processor looks for patterns in the source tree that match the patterns contained in an XSLT style sheet. A **pattern** is a sequence of nested elements that represent a branch in an XML document tree. A **source tree** is the document tree of the XML document that is being transformed. If a branch in a source tree matches a pattern in an XSLT style sheet, then its nodes will be included with the data that is transformed into the result tree. A **result tree** is the document tree of the

634 Chapter 14 Extensible Stylesheet Language (XSL)

transformed XML document. For example, an XSL style sheet may specify that the nodes of any pattern that matches `olympics\country\gold` should be added to the result tree. This pattern specifies that an XML transformation should include the contents of any `<gold>` element that are nested within `<country>` element, which in turn are nested within the `<olympics>` root element. In the case of XSLT, the `<olympics>` element is referred to as the `olympics` root node, and the `<gold>` and `<country>` elements are referred to as the `gold` and `country` nodes.

Next, you will study the predefined elements that you can use with XSLT.

XSLT Elements

You add XSLT to XSLT style sheets using a set of predefined elements that begin with the `xsl` namespace. For example, the XSLT style sheet root element is `<xsl:stylesheet>`. Table 14-1 lists the top-level XSLT elements that you can nest within an XSLT style sheet's root element.

Table 14-1 Top-level XSLT elements

Element	Description
<code><xsl:import></code>	Imports another XSLT style sheet
<code><xsl:include></code>	Includes another XSLT style sheet
<code><xsl:strip-space></code>	Identifies elements in the source XML document that should be stripped of white space before they are transformed
<code><xsl:preserve-space></code>	Identifies elements in the source XML document that should not be stripped of white space before they are transformed
<code><xsl:output></code>	Specifies the output format of the result tree
<code><xsl:key></code>	Defines a key for a node that can be referenced elsewhere in the document using the XSLT <code>key()</code> function
<code><xsl:decimal-format></code>	Defines the decimal format to be used when converting numbers into strings with the XSLT <code>format-number()</code> function
<code><xsl:namespace-alias></code>	Declares one namespace Uniform Resource Identifier (URI) as an alias for another namespace Uniform Resource Locator (URL)
<code><xsl:attribute-set></code>	Defines a named set of attributes
<code><xsl:variable></code>	Declares a variable
<code><xsl:param></code>	Declares a parameter
<code><xsl:template></code>	Defines a template rule which contains a pattern for identifying nodes that should be transformed

XSLT includes additional elements, called instruction elements that you can nest within the elements listed in Table 14-1. For example, the `<xsl:sort>` element is used to sort the nodes that appear in the result tree and can be nested within the `<xsl:template>` element. You will work with several of the instruction elements in this chapter.



You can find a complete listing of XSLT elements and functions in Appendix F.

XSLT Style Sheets

You create XSLT style sheets using a text editor, just like when you create XHTML and CSS files. However, you need to use a filename extension of **.xsl** for your XSLT style sheets. Because an XSLT style sheet is also an XML document, it needs to contain a root element. An XSLT style sheet's root element can be either `<xsl:stylesheet>` or `<xsl:transform>`. Both of these elements perform identical functions in that they declare the document to be an XSLT style sheet. You can use either one, but `<xsl:stylesheet>` is more commonly used.

The W3C recommends that you declare your XSLT style sheet root elements using the following syntax. The following code also includes an XML declaration (or processing instruction) in the first line to specify the version of XML being used.

```
<?xml version="1.0" encoding="iso-8859-1"
standalone="yes"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  style declarations
</xsl:stylesheet>
```



You studied processing instructions in Chapter 1. Recall that a processing instruction is a special statement that passes information to the user agent or application that is processing the XML document. You can easily recognize processing instructions because they begin with `<?` and end with `?>`.

To link an XSLT style sheet to an XML document, you add to the XML document an `<xsl-stylesheet>` processing instruction similar to the following:

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

The `<xsl-stylesheet>` processing instruction should include the two properties shown in the preceding code: **type**, which is assigned a value of "text/xsl" and **href**, which is assigned the name of the XSLT style sheet. You add the `<xsl-stylesheet>` processing instruction after the XML declaration. The following code shows the Olympic Medals XML document with an `<xsl-stylesheet>` processing instruction that links the document to an XSLT style sheet named **olympics.xsl**:

```
<?xml version="1.0" encoding="iso-8859-1"
standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="olympics.xsl"?>
<olympics>
  <year>2002 Winter Olympics</year>
  <medals>Olympic Medal Counts</medals>
...
```

636 Chapter 14 Extensible Stylesheet Language (XSL)

To format an XML document to display as a Web page, you add the usual XHTML elements to an XSLT style sheet document, as the content of the `<xsl:stylesheet>` element. For example, the following code shows the basis of an XSLT style sheet that will format the Olympic Medals XML document as a Web page. Notice that instead of including a `<!DOCTYPE>` declaration to identify the resulting Web page as XHTML Strict, the document simply includes another `xmlns` attribute in the `<xsl:stylesheet>` element that is assigned a value of `"http://www.w3.org/TR/xhtml1/strict"`. Because the document is an XSLT style sheet, you cannot declare it as an XHTML Strict document. Using the `xmlns="http://www.w3.org/TR/xhtml1/strict"` attribute, however, identifies the output document as XHTML Strict.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<html>
<head xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<title>2002 Winter Olympics</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
<style type="text/css">
body { font-family: Verdana, Arial, sans-serif }
h1 { font-size: 1.5em; color: navy; background-color:
transparent }
h2 { font-size: 1.2em; color: navy; background-color:
transparent }
p, td, th { font-size: .8em; color: olive;
background-color: transparent }
</style>
</head>
<body>
</body>
</html>
</xsl:stylesheet>
```

The preceding document is only the basis for an XSLT style sheet that will transform an XML document into a Web page. To complete it, you still need to add additional XSLT elements that identify which nodes from the source tree will be added to the transformed result tree.

Next, you will start working on an XSLT style sheet that formats the contents of an XML file named `Forecast.xml` that contains weather forecast data for selected American cities. You can find a copy of the `Forecast.xml` file in your `Chapter.14\Chapter` folder. The file contains a root element named `<weather>` that contains a single `<data>` element, along with numerous `<forecast>` elements for various cities. City names are assigned to a `city` attribute in each `<forecast>` element. Each forecast element also contains three nested elements: `<high_temp>`, `<low_temp>`, and `<conditions>`.

To create an XSLT style sheet:

1. Create a new file in your text editor.
2. Type the opening XML declaration, as follows:


```
<?xml version="1.0" encoding="iso-8859-1"
standalone="yes"?>
```
3. Next, type the following `<xsl:stylesheet>` element:


```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
</xsl:stylesheet>
```
4. Now add the following elements within the `<xsl:stylesheet>` element. These elements will form the basis of how the `Forecast.xml` file will display in a Web browser.


```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>Weather Forecast</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
<style type="text/css">
body { background-color: silver; color: navy; font-family:
'Trebuchet MS', Arial, Helvetica }
h1 { font-size: 2em; font-weight: normal }
h2 { font-size: 1.2em; font-weight: normal }
</style></head>
<body>
<h1>Weather Forecast</h1>
</body>
</html>
```
5. Save the file as **WeatherForecast.xsl** in your Chapter.14\Chapter folder.
6. Open the **Forecast.xml** file in your text editor and add the following `<xsl-stylesheet>` processing instruction immediately after the XML declaration:


```
<?xml-stylesheet type="text/xsl"
href="WeatherForecast.xsl"?>
```
7. Save the **Forecast.xml** document and open it in Explorer. Although the style sheet does not transform any elements yet, you should see the heading element.
8. Close your Web browser window.

638 Chapter 14 Extensible Stylesheet Language (XSL)

WORKING WITH TEMPLATES

A **template** is created with the `<xsl:template>` element and defines the transformation procedures for a node or group of nodes that match a given pattern. The `<xsl:template>` element is arguably the most important XSLT element because it selects and applies rules to the nodes that will be added to the result tree.

The `<xsl:template>` element has several attributes, the most important of which is the **match** attribute, which specifies the pattern to which the template will apply. Assigning a value of `"/"` to the **match** attribute specifies that the template will apply to the root node. This means that the entire source tree is available for transformation. However, this does not mean that the entire source tree will automatically be included in the result tree. You need to specify which nodes you want included in the transformation. Before you can specify which nodes to include in a transformation, you need to understand how to use patterns in XSLT.

Patterns

Assigning a value of `"/"` to the `<xsl:template>` element's **match** attribute essentially gives you access to all of the nodes in the document. One way to specify which nodes you want included in the transformation is to use the `<xsl:value-of>` element to access a node's value. The only required value of the `<xsl:value-of>` element is the **select** attribute, which you use to specify the node you want included in the transformation. If you use a **match** attribute value of `"/"` with the Olympic Medals XML document, you can access the value of each of the document's top-level nodes by assigning the name of the node, preceded by a slash (`/`) and the name of the root node to the **select** attribute. For example, the following code shows the XSLT style sheet for the Olympic Medals XML document. An `<xsl:template>` element now contains the XHTML elements. Two `<xsl:value-of>` element's in the `<body>` element select the content of the **year** and **medals** nodes for transformation. Notice that the first `<xsl:value-of>` element accesses the value of the **year** node by assigning the value `"olympics/year"` to the **select** attribute. If you opened the Olympic Medals XML document in Internet Explorer, you would see the output shown in Figure 14-4.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>2002 Winter Olympics</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
<style type="text/css">
```

```

body { font-family: Verdana, Arial, sans-serif }
h1 { font-size: 1.5em; color: navy; background-color:
transparent }
h2 { font-size: 1.2em; color: navy; background-color:
transparent }
p, td, th { font-size: .8em; color: olive;
background-color: transparent }
</style>
</head>
<body>
<h1><xsl:value-of select="olympics/year" /></h1>
<h2><xsl:value-of select="olympics/medals" /></h2>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

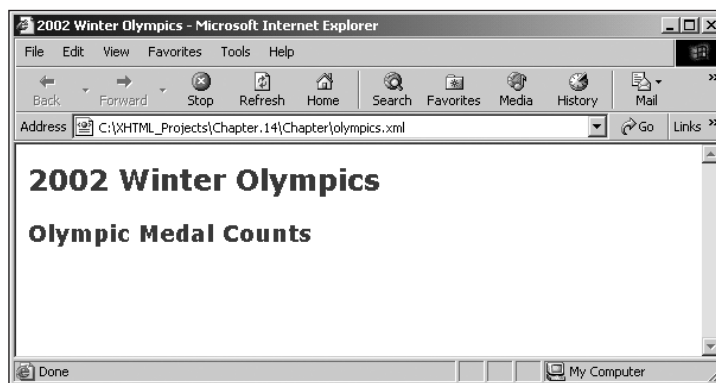


Figure 14-4 Olympic Medals XML document after adding two `<xsl:value-of>` elements to the style sheet

14

If you are familiar with path statements used in file systems, then you probably recognize the syntax for assigning values to the `<xsl:value-of>` element's `select` attribute. The values you assign to the `select` attribute are actually XPath instructions. XPath gets its name because it uses a path syntax to access the nodes in an XML document in much the same way you access folders and files using path statements. Following this syntax, you can access nodes that are nested within an XML document's top-level nodes by appending them with slashes to the value you assign to the `select` attribute. For example, you may have an XML document with a root node named `banking` that contains a top-level node named `checking`. The `checking` node in turn may contain a node named `balance`. You can use the `<xsl:value-of>` element to access the value of the `balance` node using the following statement:

```
<xsl:value-of select="banking/checking/balance" />
```

640 Chapter 14 Extensible Stylesheet Language (XSL)

Assigning a value of “1/” to the `<xsl:template>` element’s `match` attribute sets the root node as the current node. The term **current node** refers to the node that is assigned to an `<xsl:template>` element’s `match` attribute. You can access the nodes within the current node without specifying the node path. For example, the following code assigns a value of “banking/checking” to the `<xsl:template>` element’s `match` attribute, making the **checking** the current node. Because the **checking** node is the current node, the value assigned to the `<xsl:value-of>` element’s `select` attribute does not need to specify the node path to access the value of the **balance** node. In fact, if you attempt to specify the node path, you will receive an error because the XML processor would attempt to look for the **banking/checking** path beneath the current **balance** node.

```
<xsl:template match="banking/checking">
<xsl:value-of select="balance" />
</xsl:template>
```

To access the value of an attribute node, you precede the node name with an ampersand (@). Returning to the banking XML example, suppose the **checking** node includes an attribute node named **interest**. If the attribute node is an attribute of the current element node, then you can access its value by assigning the value “@interest” to the `<xsl:value-of>` element’s `select` attribute. The following code shows how you can access the value of the **interest** attribute node if the current node for the banking XML document is the root node:

```
<xsl:template match="/">
<xsl:value-of select="banking/checking/@interest" />
</xsl:template>
```

The following code shows another example of the body section of the Olympic Medals XSLT style sheet. This time, the code includes a table that displays the medal count information for the first country in the XML document, Germany. Figure 14-5 shows how the Olympic Medals XML document appears in Internet Explorer.

```
...
<body>
<h1><xsl:value-of select="olympics/year" /></h1>
<h2><xsl:value-of select="olympics/medals" /></h2>
<table width="100%" border="1">
<colgroup span="1" align="left" />
<colgroup span="3" align="center" />
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
<tr>
<td><xsl:value-of select="olympics/country/@name" /></td>
<td><xsl:value-of select="olympics/country/gold" /></td>
<td><xsl:value-of select="olympics/country/silver" /></td>
```

```
 <xsl:value-of select="olympics/country/bronze" /></td> </tr> </table> </body> </html> </xsl:template> </xsl:stylesheet> |
```

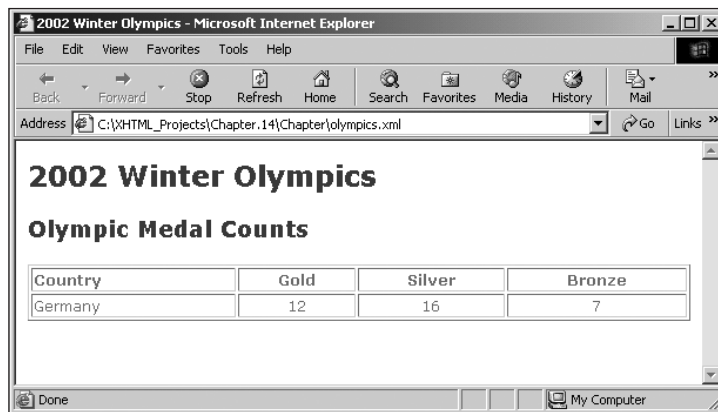


Figure 14-5 Olympic Medals XML document after adding a table to the style sheet

One limitation of the `<xsl:value-of>` element is that it only applies to the first matching node in the XML document. Because Germany is the first country node in the Olympic Medals XML document, it is the only node included in the result tree. In the next section, you will learn how to transform all of the country nodes to the result tree, not just nodes for Germany.

Next, you will add an `<xsl:template>` element and `<xsl:value-of>` elements to the WeatherForecast.xsl file.

To add an `<xsl:template>` element and `<xsl:value-of>` elements to the WeatherForecast.xsl file:

1. Return to the WeatherForecast.xsl file in your text editor.
2. Add an opening `<xsl:template match="/">` tag immediately above the opening `<html>` tag.
3. Add a closing `</xsl:template>` tag immediately above the closing `</xsl:stylesheet>` tag.
4. Now add the following `<h2>` element immediately after the `<h1>` element. The element contains a nested `<xsl:value-of>` element that adds the `<date>` element to the result tree:

```

<h2>for selected American cities on <xsl:value-of
select="weather/date" /></h2>

```

642 Chapter 14 Extensible Stylesheet Language (XSL)

5. Next, add the following table after the `<h2>` element to display the transformed data for each city forecast from the `Forecast.xml` file. The table also uses `<xsl:value-of>` elements to add data to the result tree.

```
<table width="100%" border="1">
<colgroup span="1" align="left" />
<colgroup span="3" align="center" />
<tr>
<th>City</th>
<th>High Temperature</th>
<th>Low Temperature</th>
<th>Conditions</th>
</tr>
<tr>
<td><xsl:value-of select="weather/forecast/@city" /></td>
<td><xsl:value-of select="weather/forecast/
    high_temp" /></td>
<td><xsl:value-of select="weather/forecast/
    low_temp" /></td>
<td><xsl:value-of select="weather/forecast/
    conditions" /></td>
</tr>
</table>
```

6. Save the **WeatherForecast.xsl** file and then open the **Forecast.xml** file in Internet Explorer. The data from the first city, Albuquerque, should appear in the table as shown in Figure 14-6. You will learn how to add the data for the rest of the cities in the next section.

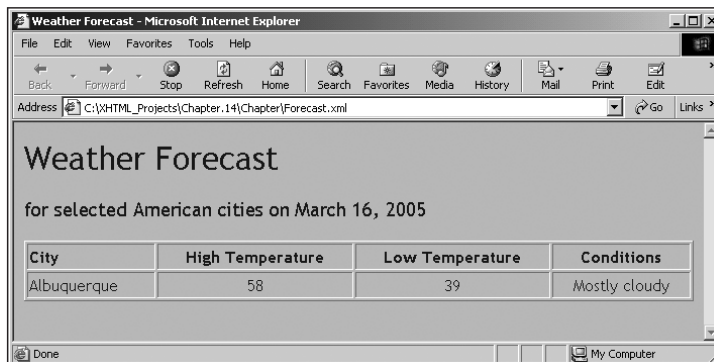


Figure 14-6 Forecast.xml after adding `<xsl:value-of>` elements

7. Close your Web browser window.

In addition to directly assigning a path as a select attribute's pattern, you can also use the references listed in Table 14-2.

Table 14-2 Pattern references

Reference	Description
.	Current node
/	Root node
..	Parent node
//	All child nodes
*	All nodes

Combining one of the pattern references listed in Table 14-2 with the `<xsl:value-of>` element transforms the contents of the referenced node (or nodes) into the result tree. For example, if you use a period reference (.) as a pattern, then the contents of the current node are transformed. However, if you use two slashes (//), then the contents of all child nodes of the current node will be transformed. For example, the `<xsl:template>` element in the following code looks for the “banking/checking/balance” pattern. To access the contents of `balance` node, the `select` attribute of the `<xsl:value-of>` element is assigned a value “.”.

```
<xsl:template match="banking/checking/balance">
  <xsl:value-of select="." />
</xsl:template>
```



This section only presents the tip of the iceberg when it comes to patterns. If you would like to learn more about XSLT patterns, then see the “5.2 Patterns” topic in the W3C’s XSL Transformations (XSLT) Recommendation at <http://www.w3.org/TR/xslt#patterns>.

The `<xsl:apply-templates>` Element

14

You will often want to apply the same transformation rules to all of the nodes in an XML document that match a given name. For example, with the Olympic Medals XML document, you would want the data for all of the `country` nodes added to the result tree, not just the data for Germany’s `country` node. To specify XSLT rules that will transform all matching nodes in a source tree to the result tree, you need to create an additional `<xsl:template>` element for the node’s pattern. For example, in the Olympic Medals style sheet, you would add the following new `<xsl:template>` element after the closing `</xsl:template>` tag that sets up the basic transformation structure:

```
<xsl:template match="olympics/country">
  <tr>
    <td><xsl:value-of select="@name" /></td>
    <td><xsl:value-of select="gold" /></td>
    <td><xsl:value-of select="silver" /></td>
    <td><xsl:value-of select="bronze" /></td>
  </tr>
</xsl:template>
```

644 Chapter 14 Extensible Stylesheet Language (XSL)



Nesting one `<xsl:template>` element inside another will generate an error.

The preceding template looks for all nodes that match the “olympics/country” path and transforms each node’s data into a table row in the result tree. In order to use the new template, you need to use the **`<xsl:apply-templates>` element** to specify where the nodes should be placed in the result tree. You include a **`select`** attribute in the **`<xsl:apply-templates>` element** to specify the node whose template should be applied. With the Olympic Medals style sheet, you place the **`<xsl:apply-templates>` element** after the closing `</tr>` tag for the table header row and above the closing `</table>` tag. The following code shows the Olympic Medals style sheet that calls the new template. Keep in mind that the code includes two templates. The first template assigns a value of “/” to the **`match`** attribute, which gives the style sheet access to all of the nodes in the document. This template also determines the XHTML document structure that will be added to the result tree. The second template only specifies transformation rules for the **`country`** node. The **`<xsl:apply-templates>` element** in the first template applies the transformation rules in the template for the **`country`** node. Figure 14-7 shows the Olympic Medals document as it appears in Internet Explorer.

```
<xsl:template match="/">
...
<body>
<h1><xsl:value-of select="olympics/year" /></h1>
<h2><xsl:value-of select="olympics/medals" /></h2>
<table width="100%" border="1">
<colgroup span="1" align="left" />
<colgroup span="3" align="center" />
<tr>
<th>Country</th>
<th>Gold</th>
<th>Silver</th>
<th>Bronze</th>
</tr>
<xsl:apply-templates select="olympics/country" />
</table>
</body>
</html>
</xsl:template>
<xsl:template match="olympics/country">
<tr>
<td><xsl:value-of select="@name" /></td>
<td><xsl:value-of select="gold" /></td>
<td><xsl:value-of select="silver" /></td>
<td><xsl:value-of select="bronze" /></td>
</tr>
</xsl:template>
```



Figure 14-7 Olympic Medals XML document after adding a template for the country node



The order in which `<xsl:template>` elements appear in an XSLT style sheet makes no difference.

Next, you will add another template to the `WeatherForecast.xsl` file that adds the data in all of the `<forecast>` elements to the result tree.

To add another template to the `WeatherForecast.xsl` file that adds the data in all of the `<forecast>` elements to the result tree:

1. Return to the **WeatherForecast.xsl** file in your text editor window.
2. Add the following new `<xsl:template>` element and table row elements above the closing `<xsl:stylesheet>` element:

```
<xsl:template match="weather/forecast">
  <tr>
    <td><xsl:value-of select="@city" /></td>
    <td><xsl:value-of select="high_temp" /></td>
    <td><xsl:value-of select="low_temp" /></td>
    <td><xsl:value-of select="conditions" /></td>
  </tr>
</xsl:template>
```

3. Replace the table row elements in the document body with the following `<xsl:apply-templates>` element:

```
...
<th>Conditions</th>
</tr>
<xsl:apply-templates select="weather/forecast" />
</table>
</body>
```

646 Chapter 14 Extensible Stylesheet Language (XSL)

4. Save the **WeatherForecast.xsl** file and open the **Forecast.xml** file in Internet Explorer. Your Web browser should appear similar to Figure 14-8.

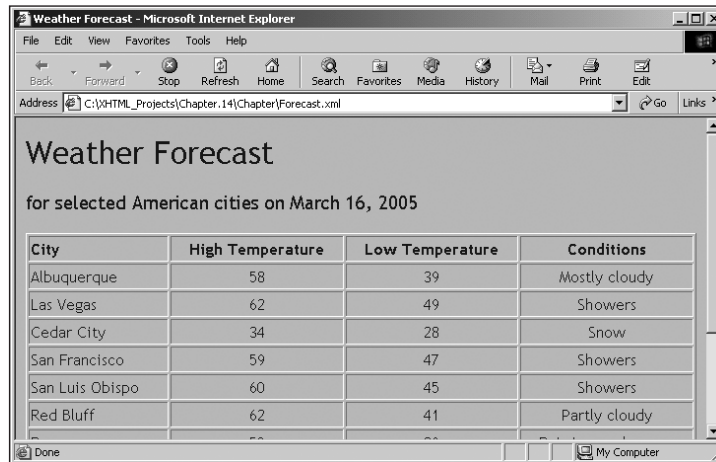


Figure 14-8 Forecast.xml after adding a new template

5. Close your Web browser window.

MANIPULATING TRANSFORMED DATA

In this section, you will learn how to use additional XSLT instruction elements to perform repetitions, make decisions, and sort XML data.

Repetition

Most programming languages include **repetition**, or **loop**, statements that repeatedly execute a statement or a series of statements while a specific condition is true or until a specific condition becomes true. The **<xsl:for-each>** element loops through the nodes in a source tree that match a given pattern, applying the same transformation rules to each node. On the surface, this is not much different than creating an additional **<xsl:template>** element for the node's pattern and applying it with the **<xsl:apply-templates>** element. However, the **<xsl:for-each>** element can be nested within **<xsl:template>** element and does not need to be applied with the **<xsl:apply-templates>** element. For example, instead of creating an additional **<xsl:template>** element for the Olympic Medals XML document, you can simply add the following **<xsl:for-each>** element in place of the **<xsl:apply-templates>** element:

```
<xsl:for-each select="olympics/country">
  <tr>
    <td><xsl:value-of select="@name" /></td>
```

```

<td><xsl:value-of select="gold" /></td>
<td><xsl:value-of select="silver" /></td>
<td><xsl:value-of select="bronze" /></td>
</tr>
</xsl:for-each>

```

Whether you use an `<xsl:template>` element or use an `<xsl:for-each>` element will depend on the situation. In general, you should use an `<xsl:template>` element when you need to use the same transformation rules in multiple places within an XSLT style sheet. However, you should use an `<xsl:for-each>` element if you anticipate the transformation rules will only be used once within the template.

Next, you will replace the `<xsl:template>` element in `WeatherForecast.xml` with an `<xsl:for-each>` element.

To replace the `<xsl:template>` element in `WeatherForecast.xml` with an `<xsl:for-each>` element:

1. Return to the **WeatherForecast.xml** document in your text editor.
2. Replace the `<xsl:apply-templates>` element in the document body with the following `<xsl:for-each>` element:

```

<xsl:for-each select="weather/forecast">
<tr>
<td><xsl:value-of select="@city" /></td>
<td><xsl:value-of select="high_temp" /></td>
<td><xsl:value-of select="low_temp" /></td>
<td><xsl:value-of select="conditions" /></td>
</tr>
</xsl:for-each>

```

3. Delete the `<xsl:template>` element that appears above the closing `</xsl:stylesheet>` tag along with its contents.



Be sure not to delete the closing `</xsl:template>` tag that appears directly after the `</html>` tag.

4. Save the **WeatherForecast.xml** file and open the **Forecast.xml** file in Internet Explorer. The file should render the same as it did before you added the `<xsl:for-each>` element.
5. Close your Web browser window.

Decision Making

XSLT includes two decision-making elements, `<xsl:if>` and `<xsl:choose>`, that are similar to some of the JavaScript decision-making statements you studied in Chapter 10. The `<xsl:if>` element is similar to the JavaScript `if` statement while the `<xsl:choose>`

648 Chapter 14 Extensible Stylesheet Language (XSL)

element is similar to the JavaScript `switch` statement. Both elements use conditional expressions to determine whether to apply transformation rules. XPath comparison operators work in a similar fashion to JavaScript comparison operators, although there are some differences, one of the most important is that you must use a character entity for any comparison operators that include the `<` or `>` characters. Table 14-3 lists some of the more common XPath comparison operators.

Table 14-3 Common XPath comparison operators

Operator	Description
<code>=</code>	Determines if the values are equal
<code>!=</code>	Determines if the values are not equal
<code>&lt;</code>	Determines if one value is less than another value
<code>&lt;=</code>	Determines if one value is less than or equal to another value
<code>&gt;</code>	Determines if one value is greater than another value
<code>&gt;=</code>	Determines if one value is greater than or equal to another value
<code>and</code>	Determines whether two conditional expressions are both true
<code>or</code>	Determines if either of two conditional expressions are true

First you will look at the `<xsl:if>` element.

The `<xsl:if>` Element

The `<xsl:if>` element applies transformation rules if a conditional expression is true. You must nest the `<xsl:if>` element beneath the `<xsl:template>` element or the `<xsl:for-each>` element. The `<xsl:if>` element includes a single attribute, `test`, to which you assign a conditional expression.

As an example of how to use the `<xsl:if>` element, consider the following XML document, which contains stock information for an investment portfolio.

```
<portfolio>
  <stock>
    <name>BEA Systems</name>
    <symbol>BEAS</symbol>
    <exchange>NASDAQ</exchange>
    <last_trade>
      <date>March 14</date>
      <price>11.07</price>
    </last_trade>
  </stock>
  <stock>
    <name>Oracle</name>
    <symbol>ORCL</symbol>
    <exchange>NASDAQ</exchange>
    <last_trade>
      <date>March 14</date>
```

```

        <price>11.94</price>
      </last_trade>
    </stock>
    <stock>
      <name>Bolt Technology</name>
      <symbol>BTJ</symbol>
      <exchange>AMEX</exchange>
      <last_trade>
        <date>March 14</date>
        <price>3.04</price>
      </last_trade>
    </stock>
    <stock>
      <name>Medifast</name>
      <symbol>MED</symbol>
      <exchange>AMEX</exchange>
      <last_trade>
        <date>March 14</date>
        <price>4.77</price>
      </last_trade>
    </stock>
    <stock>
      <name>WR Grace</name>
      <symbol>GRA</symbol>
      <exchange>NYSE</exchange>
      <last_trade>
        <date>March 14</date>
        <price>2.06</price>
      </last_trade>
    </stock>
  </portfolio>

```

To transform only the stocks that trade on the NASDAQ exchange, you use the following XSLT template that includes a nested `<xsl:if>` element. Notice that because the conditional expression assigned to the `test` attribute is contained within double quotations, NASDAQ is surrounded by single quotations. If a stock's exchange is NASDAQ, then a table row is added to the result tree that includes the node values. If you add the following template to an XSLT style sheet using the `<xsl:apply-templates>` element, then the result tree in Internet Explorer will appear similar to Figure 14-9.

```

<xsl:template match="portfolio/stock">
  <xsl:if test="exchange='NASDAQ'">
    <tr>
      <td><xsl:value-of select="name" /></td>
      <td><xsl:value-of select="symbol" /></td>
      <td><xsl:value-of select="last_trade/date" /></td>
      <td><xsl:value-of select="last_trade/price" /></td>
    </tr>
  </xsl:if>
</xsl:template>

```

650 Chapter 14 Extensible Stylesheet Language (XSL)

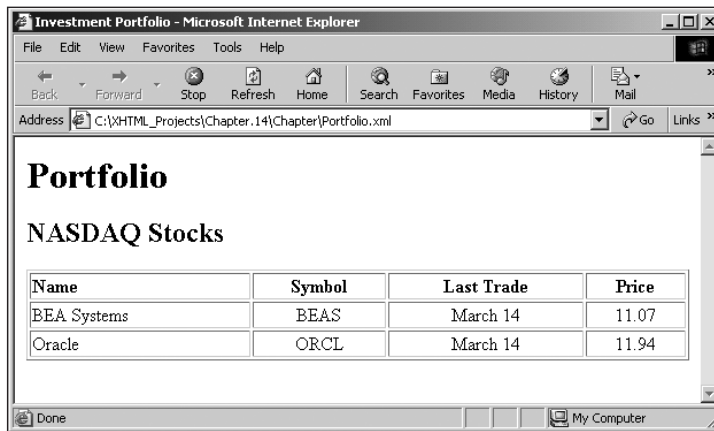


Figure 14-9 Results from a style sheet that includes an `<xsl:if>` element

The following template shows another example of how to use the `<xsl:if>` element. This time, the element uses the XPath less-than comparison operator (`<`) to transform any stocks that are selling for less than \$10.00. Figure 14-10 shows the result tree in Internet Explorer.

```
<xsl:template match="portfolio/stock">
  <xsl:if test="last_trade/price &lt; 10">
    <tr>
      <td><xsl:value-of select="name" /></td>
      <td><xsl:value-of select="symbol" /></td>
      <td><xsl:value-of select="last_trade/date" /></td>
      <td><xsl:value-of select="last_trade/price" /></td>
    </tr>
  </xsl:if>
</xsl:template>
```

Next, you will add an `<xsl:if>` element to `WeatherForecast.xml` that adds only cities that are expecting showers to the result tree.

To add an `<xsl:if>` element to `WeatherForecast.xml` that adds only cities that are expecting showers to the result tree:

1. Return to the **WeatherForecast.xml** document in your text editor.
2. Modify the `<xsl:for-each>` element so it includes an `<xsl:if>` element that adds only cities that are expecting showers to the result tree, as follows:

```
<xsl:for-each select="weather/forecast">
  <xsl:if test="conditions='Showers'">
    <tr>
      <td><xsl:value-of select="@city" /></td>
      <td><xsl:value-of select="high_temp" /></td>
      <td><xsl:value-of select="low_temp" /></td>
```



```

<td><xsl:value-of select="conditions" /></td>
</tr>
</xsl:if>
</xsl:for-each>

```

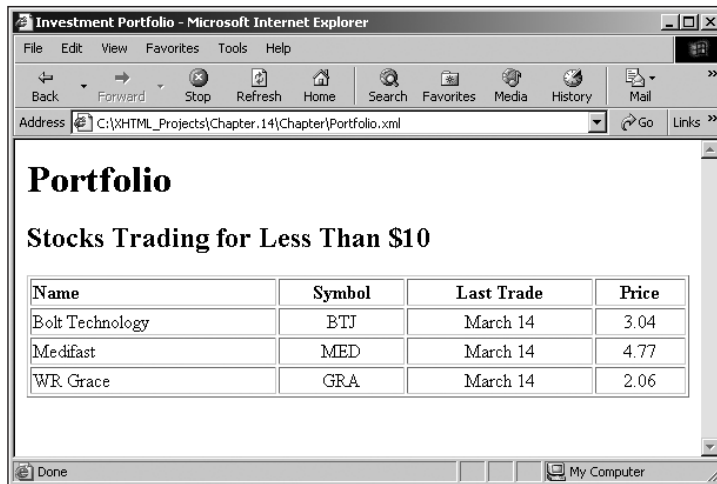


Figure 14-10 Results from a style sheet that includes an `<xsl:if>` element with a less-than comparison operator



The condition you assign to an `<xsl:if>` element's test attribute is case sensitive. Be sure to type "Showers" with an uppercase 'S'.

3. Save the **WeatherForecast.xsl** file and open the **Forecast.xml** file in Internet Explorer. The table should only show cities that are expecting showers, as shown in Figure 14-11.
4. Close your Web browser window.

14

The `<xsl:choose>` Element

The `<xsl:choose>` element applies different sets of transformation rules based on multiple conditional expression. As with the `<xsl:if>` element, the `<xsl:choose>` element cannot be used as a top-level element beneath the `<xsl:stylesheet>` root element. You must nest the `<xsl:choose>` element beneath the `<xsl:template>` element or the `<xsl:for-each>` element.

652 Chapter 14 Extensible Stylesheet Language (XSL)

City	High Temperature	Low Temperature	Conditions
Las Vegas	62	49	Showers
San Francisco	59	47	Showers
San Luis Obispo	60	45	Showers
Salt Lake City	48	32	Showers

Figure 14-11 Forecast.xml after adding an `<xsl:if>` element

The `<xsl:choose>` element is used with the `<xsl:when>` element and the `<xsl:otherwise>` element. You do not use the `test` attribute with the `<xsl:choose>` element. Instead, you use the `test` attribute with The `<xsl:when>` element. The `<xsl:when>` element is equivalent to a `switch` statement's `case` label while the `<xsl:otherwise>` element is equivalent to a `switch` statement's `default` label. You do not use the `test` attribute with the `<xsl:choose>` element. Instead, you use the `test` attribute with The `<xsl:when>` element. The following code shows an example of a template for the portfolio style sheet that includes nested `<xsl:if>`, `<xsl:when>`, and `<xsl:otherwise>` elements. The `<xsl:if>` element checks each `stock` node to determine its exchange. Then, `<xsl:when>` elements add a red table row to the result tree for NASDAQ, a blue table row for AMEX, and a green table row for NYSE. The `<xsl:otherwise>` element formats table rows in black for any nodes that do not match the `<xsl:when>` elements. Figure 14-12 shows how the result tree appears in Internet Explorer for the portfolio XML document you saw earlier.

```
<xsl:template match="portfolio/stock">
  <xsl:choose>
    <xsl:when test="exchange='NASDAQ'">
      <tr style="color: red">
        <td><xsl:value-of select="name" /></td>
        <td><xsl:value-of select="symbol" /></td>
        <td><xsl:value-of select="last_trade/date" /></td>
        <td><xsl:value-of select="last_trade/price" /></td>
      </tr>
    </xsl:when>
    <xsl:when test="exchange='AMEX'">
      <tr style="color: blue">
        <td><xsl:value-of select="name" /></td>
        <td><xsl:value-of select="symbol" /></td>
        <td><xsl:value-of select="last_trade/date" /></td>
```

```

<td><xsl:value-of select="last_trade/price" /></td>
</tr>
</xsl:when>
<xsl:when test="exchange='NYSE'">
<tr style="color: green">
<td><xsl:value-of select="name" /></td>
<td><xsl:value-of select="symbol" /></td>
<td><xsl:value-of select="last_trade/date" /></td>
<td><xsl:value-of select="last_trade/price" /></td>
</tr>
</xsl:when>
<xsl:otherwise>
<tr style="color: black">
<td><xsl:value-of select="name" /></td>
<td><xsl:value-of select="symbol" /></td>
<td><xsl:value-of select="last_trade/date" /></td>
<td><xsl:value-of select="last_trade/price" /></td>
</tr>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

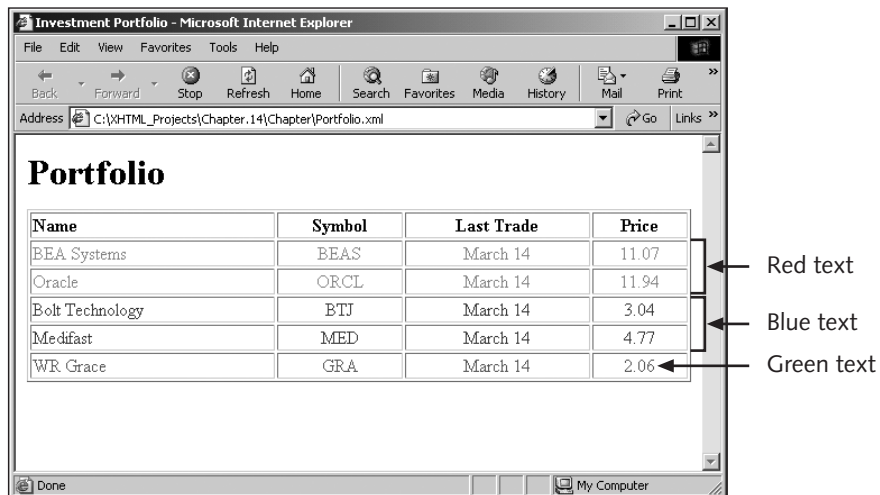


Figure 14-12 Results from a style sheet that includes an `<xsl:choose>` element

Next, you will add to `WeatherForecast.xml` an `<xsl:choose>` element that formats cities that are expecting showers in red.

To add to `WeatherForecast.xml` an `<xsl:choose>` element that formats cities that are expecting showers in red:

1. Return to the **WeatherForecast.xml** file in your text editor.

654 Chapter 14 Extensible Stylesheet Language (XSL)

2. Replace the `<xsl:if>` element in the `<xsl:for-each>` element with an `<xsl:choose>` element that formats cities that are expecting showers in red, as follows:

```
<xsl:for-each select="weather/forecast">
  <xsl:choose>
    <xsl:when test="conditions='Showers'">
      <tr style="color:red">
        <td><xsl:value-of select="@city" /></td>
        <td><xsl:value-of select="high_temp" /></td>
        <td><xsl:value-of select="low_temp" /></td>
        <td><xsl:value-of select="conditions" /></td>
      </tr>
    </xsl:when>
    <xsl:otherwise>
      <tr>
        <td><xsl:value-of select="@city" /></td>
        <td><xsl:value-of select="high_temp" /></td>
        <td><xsl:value-of select="low_temp" /></td>
        <td><xsl:value-of select="conditions" /></td>
      </tr>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

3. Save the **WeatherForecast.xsl** file and open the **Forecast.xml** file in Internet Explorer. The cities that are expecting showers should be formatted in red, as shown in Figure 14-13.

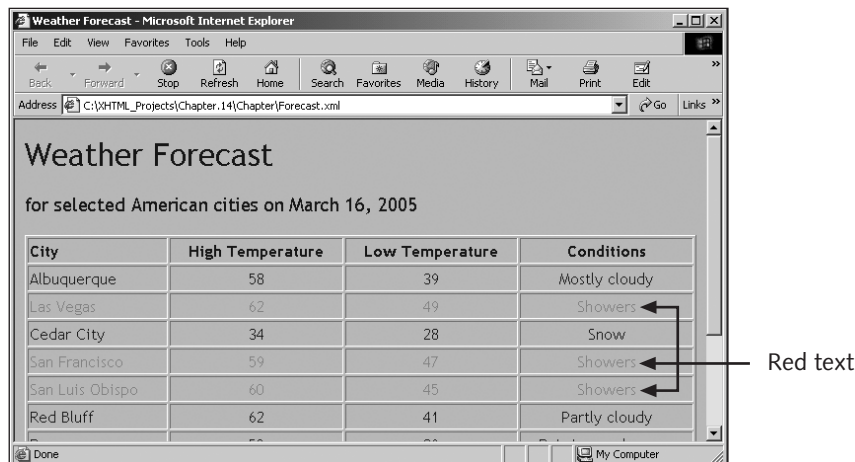


Figure 14-13 Forecast.xml after adding an `<xsl:choose>` element

4. Close your Web browser window.

Sorting

The **<xsl:sort> element** allows you to sort the nodes that are added to the result tree. You can nest the **<xsl:sort>** element beneath the **<xsl:template>** and **<xsl:for-each>** elements. The **<xsl:sort>** element includes the attributes listed in Table 14-4.

Table 14-4 Attributes of the **<xsl:sort>** element

Attribute	Description
case-order	Specifies whether uppercase letters should be sorted before lowercase letters; valid values are "upper-first" and "lower-first"
data-type	Specifies the data type of the nodes to be sorted; valid values include "text" and "number"
lang	Specifies the language of the nodes to be sorted; accepts the same values as the lang and xml:lang standard attributes
order	Determines the order that the nodes should be sorted; valid values are "ascending" or "descending"
select	Specifies the node you want sorted

The most important of the **<xsl:sort>** element attributes is the **select** attribute, which specifies the node by which to sort. As an example of how to use the **<xsl:sort>** element, consider the following XML document, which lists populations of American cities.

```

<demographics>
  <municipality>
    <city>Los Angeles</city>
    <state>California</state>
    <population>3694820</population>
  </municipality>
  <municipality>
    <city>San Francisco</city>
    <state>California</state>
    <population>776733</population>
  </municipality>
  <municipality>
    <city>San Diego</city>
    <state>California</state>
    <population>1223400</population>
  </municipality>
  <municipality>
    <city>San Antonio</city>
    <state>Texas</state>
    <population>1144646</population>
  </municipality>
</demographics>

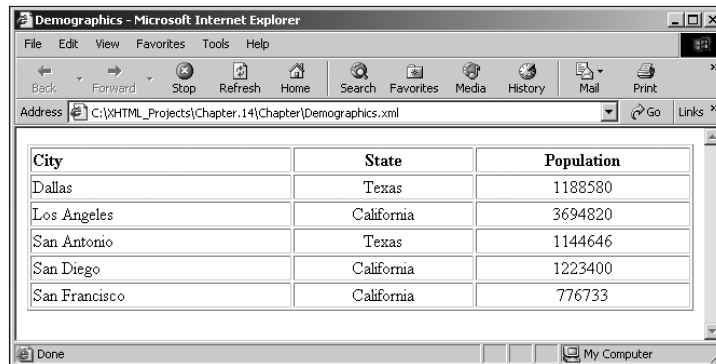
```

656 Chapter 14 Extensible Stylesheet Language (XSL)

```
<city>Dallas</city>
<state>Texas</state>
<population>1188580</population>
</municipality>
</demographics>
```

You can use the following `<xsl:for-each>` element that includes a nested `<xsl:sort>` element to sort the demographics XML document by city. Figure 14-14 shows how the result tree appears in Internet Explorer.

```
<xsl:for-each select="demographics/municipality">
  <xsl:sort select="city" order="ascending"
    data-type="text" />
  <tr>
    <td><xsl:value-of select="city" /></td>
    <td><xsl:value-of select="state" /></td>
    <td><xsl:value-of select="population" /></td>
  </tr>
</xsl:for-each>
```



City	State	Population
Dallas	Texas	1188580
Los Angeles	California	3694820
San Antonio	Texas	1144646
San Diego	California	1223400
San Francisco	California	776733

Figure 14-14 Demographics XML document sorted by city name

You can include multiple levels of sorting by adding additional `<xsl:sort>` elements. For example, the following `<xsl:for-each>` element for the demographics XML document includes two `<xsl:sort>` elements. The first `<xsl:sort>` element sorts the nodes by state; the second `<xsl:sort>` element then sorts the cities within each state by population. Figure 14-15 shows how the result tree appears in Internet Explorer.

```
<xsl:for-each select="demographics/municipality">
  <xsl:sort select="state" order="ascending"
    data-type="text" />
  <xsl:sort select="population" order="descending"
    data-type="number" />
  <tr>
    <td><xsl:value-of select="city" /></td>
```

```

<td><xsl:value-of select="state" /></td>
<td><xsl:value-of select="population" /></td>
</tr>
</xsl:for-each>

```

City	State	Population
Los Angeles	California	3694820
San Diego	California	1223400
San Francisco	California	776733
Dallas	Texas	1188580
San Antonio	Texas	1144646

Figure 14-15 Demographics XML document sorted by state and population

Next, you will add to WeatherForecast.xsl an `<xsl:sort>` element that sorts the data by city name.

To add to WeatherForecast.xsl an `<xsl:sort>` element that sorts the data by city name:

1. Return to the **WeatherForecast.xsl** file in your text editor.
2. Add an `<xsl:sort>` element after the opening `<xsl:for-each>` tag, but above the opening `<xsl:choose>` tag, as follows:

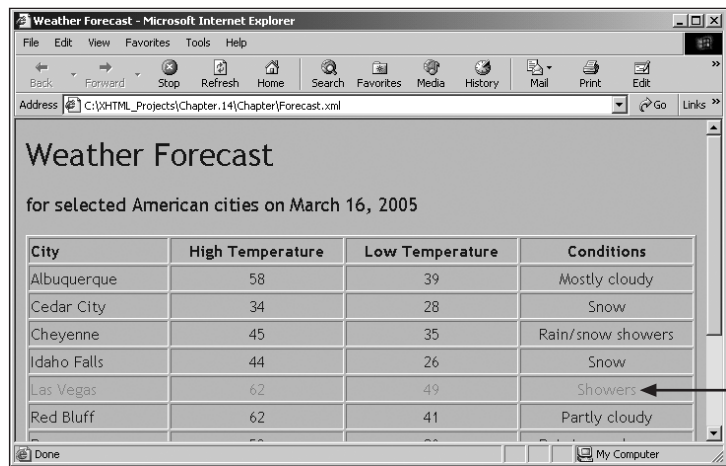
```

...
<xsl:for-each select="weather/forecast">
  <xsl:sort select="@city" order="ascending"
    data-type="text" />
  <xsl:choose>
    ...

```

3. Save the **WeatherForecast.xsl** file and open the **Forecast.xml** file in Internet Explorer. The table rows should be sorted by city name, as shown in Figure 14-16.
4. Close your Web browser window and text editor.

658 Chapter 14 Extensible Stylesheet Language (XSL)



City	High Temperature	Low Temperature	Conditions
Albuquerque	58	39	Mostly cloudy
Cedar City	34	28	Snow
Cheyenne	45	35	Rain/snow showers
Idaho Falls	44	26	Snow
Las Vegas	62	49	Showers
Red Bluff	62	41	Partly cloudy

Figure 14-16 Forecast.xml after adding an `<xsl:sort>` element

CHAPTER SUMMARY

- ❑ You can create formatted Web pages using Extensible Markup Language (XML) and Extensible Stylesheet Language (XSL), which is a style sheet language for XML.
- ❑ The term “transformation” refers to the conversion of XML data into another type of document.
- ❑ XSL Transformations (XSLT) is a language that transforms one XML document into another XML document.
- ❑ XML Path Language (XPath) is a language this is used in XSLT to access or refer to the parts of an XML document.
- ❑ XSL Formatting Objects (XSL-FO) is a language that determines how an XML document should be displayed.
- ❑ An XML processor is an application that builds a new XML document by reading a source XML document and applying the rules in an associated XSLT style sheet.
- ❑ Each element and attribute in an XML document tree is referred to as a node.
- ❑ A pattern is a sequence of nested elements that represents a branch in an XML document tree.
- ❑ A source tree is the document tree of an XML document that is being transformed.
- ❑ A result tree is the document tree of the transformed XML document.
- ❑ You add XSLT to XSLT style sheets using a set of predefined elements that begin with the `xsl` namespace.

Review Questions 659

- A template is created with the `<xsl:template>` element and defines the transformation procedures for a node or group of nodes that match a given pattern.
- One way to specify which nodes you want included in the transformation is to use the `<xsl:value-of>` element to access a node's value.
- The current node refers to the node that is assigned to an `<xsl:template>` element's `match` attribute.
- In order to use the new template, you need use the `<xsl:apply-templates>` element to specify where the nodes should be placed in the result tree.
- The `<xsl:for-each>` element loops through the nodes in a source tree that match a given pattern, applying the same transformation rules to each node.
- The `<xsl:if>` element applies transformation rules if a conditional expression is true.
- The `<xsl:choose>` element applies different sets of transformation rules based on multiple conditional expression.
- The `<xsl:sort>` element allows you to sort the nodes that are added to the result tree.

REVIEW QUESTIONS

1. Explain how XSL differs from CSS and when you should use each technology.
2. Which of the following technologies are parts of XSL? (Choose all that apply.)
 - a. XSL Transformations
 - b. Cascading Style Sheets
 - c. XML Path Language
 - d. XSL Formatting Objects
3. A(n) _____ builds a new XML document by reading a source XML document and applying the rules in an associated XSLT style sheet.
 - a. JavaScript function
 - b. DOM method
 - c. XML processor
 - d. XSLT template
4. Each element and attribute in an XML document tree is referred to as a _____.
 - a. node
 - b. branch
 - c. object
 - d. method

660 Chapter 14 Extensible Stylesheet Language (XSL)

5. A pattern represents a branch in an XML document tree. True or False?
6. Which of the following refers to the document tree of an XML document that is being transformed
 - a. directory
 - b. source tree
 - c. result tree
 - d. target tree
7. You can use either the `<xsl:stylesheet>` or `<xsl:transform>` as the root element of an XSLT style sheet. True or False?
8. What value do you assign to an `<xsl:template>` element's `match` attribute to specify that the template will apply to the root node?
 - a. /
 - b. //
 - c. .
 - d. ..
9. What is the only required attribute of the `<xsl:value-of>` element?
 - a. `test`
 - b. `node`
 - c. `select`
 - d. `match`
10. What character do you use to access the value of an attribute node?
 - a. *
 - b. &
 - c. @
 - d. #
11. Explain how to create more than one template in an XSLT style sheet and how to use the `<xsl:apply-templates>` element to specify where the nodes transformed by the template should be placed in the result tree.
12. You must nest an `<xsl:for-each>` element within an `<xsl:apply-templates>` element. True or False?
13. What is the correct way of using the greater than or equal to XPath comparison operator?
 - a. `>=`
 - b. `>>=`
 - c. `>=;`
 - d. `>=`

14. Beneath which elements can you nest the `<xsl:if>` and `<xsl:choose>` elements? (Choose all that apply.)
 - a. `<xsl:template>`
 - b. `<xsl:for-each>`
 - c. `<xsl:stylesheet>`
 - d. `<xsl:transform>`
15. Which of the following are valid values that you can apply to an `<xsl:sort>` element's `order` attribute? (Choose all that apply.)
 - a. alpha
 - b. numeric
 - c. ascending
 - d. descending

HANDS-ON PROJECTS



Project 14-1

In this project, you will create an XSL style sheet that formats and displays an XML document containing data you would find in an e-mail message. Your Chapter.14\Projects folder contains a file named Message.xml that you can use for this project.

1. Create a new document in your text editor.
2. Type the following elements that form the basis of an XSLT style sheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <html xmlns="http://www.w3.org/1999/xhtml" lang="en"
    xml:lang="en" dir="ltr">
    <head>
      <title>E-mail Message</title>
      <meta http-equiv="content-type" content="text/html;
        charset=iso-8859-1" />
    </head>
    <body>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

3. Add the following `<xsl:template>` element above the `<html>` tag to give the style sheet access to all the nodes in the Message.xml document:

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

662 Chapter 14 Extensible Stylesheet Language (XSL)

- Next, add the following elements to the document body. The elements include `<xsl:value-of>` elements that add nodes in the source tree to the result tree.

```
<h1>E-mail Message</h1>
<p><strong>To</strong>: <xsl:value-of select="message/to"
/><br />
<strong>From</strong>: <xsl:value-of select="message/from"
/><br />
<strong>Date</strong>: <xsl:value-of
select="message/received" /><br />
<strong>Subject</strong>: <xsl:value-of
select="message/subject" /></p>
<hr />
<p><xsl:value-of select="message/body" /></p>
```

- Save the XSLT style sheet as **Message.xsl** in your Chapter.14\Projects folder.
- Open the **Message.xml** document from your Chapter.14\Projects in your text editor and add the following statement immediately after the XML declaration to give the document access to the Message.xsl style sheet:

```
<?xml-stylesheet type="text/xsl" href="Message.xsl"?>
```

- Save the **Message.xml** document and open it in Internet Explorer. Figure 14-17 shows how the transformed XML document should appear.
- Close your Web browser window.

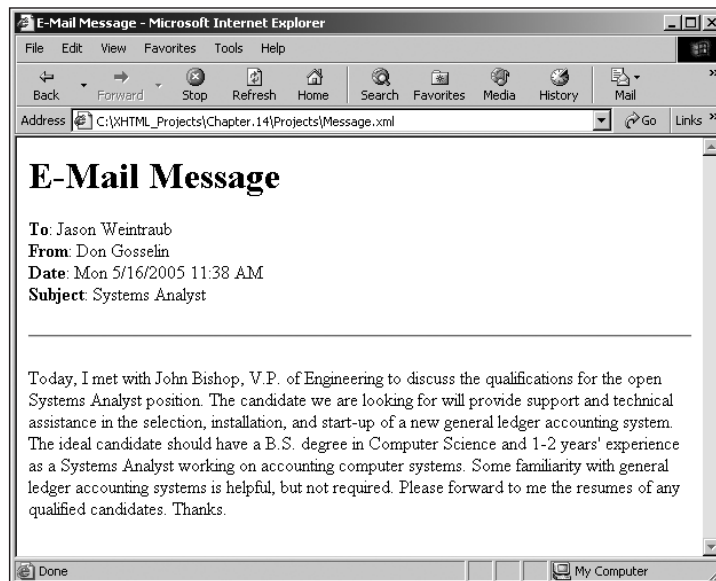


Figure 14-17 Project 14-1



Project 14-2

In this project, you will create an XSL style sheet that formats and displays an XML document containing several paragraphs from a chapter of the book *Call of the Wild*, by Jack London. Your Chapter.14\Projects folder contains a file named Book.xml that you can use for this project.

1. Create a new document in your text editor.
2. Type the following elements that form the basis of an XSLT style sheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>Call of the Wild</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
</head>
<body>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

3. Add the following `<xsl:template>` element that above the `<html>` tag to give the style sheet access to all the nodes in the Book.xml document:

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

4. Next, add the following elements to the document body. The elements include an `<xsl:value-of>` elements and an `<xsl:apply-templates>` element that calls a template that applies all the paragraph nodes in the source tree to the result tree.

```
<h1>Call of the Wild</h1>
<h2>By <xsl:value-of select="book/author" /></h2>
<h3>Chapter <xsl:value-
of select="book/chapter/chapter_num"
/>, <xsl:value-of select="book/chapter/chapter_title"
/></h3>
<xsl:apply-templates select="book/chapter/paragraph" />
```

5. Finally, add the following `<xsl:template>` element above the closing `<xsl:stylesheet>` element. This element applies all of the paragraph nodes in the source tree to the result tree.

```
<xsl:template match="book/chapter/paragraph">
<p><xsl:value-of select="." /></p>
</xsl:template>
```

6. Save the XSLT style sheet as **Book.xsl** in your Chapter.14\Projects folder.

664 Chapter 14 Extensible Stylesheet Language (XSL)

- Open the **Book.xml** document from your Chapter.14\Projects folder in your text editor and add the following statement immediately after the XML declaration to give the document access to the Book.xml style sheet:

```
<?xml-stylesheet type="text/xsl" href="Book.xsl"?>
```

- Save the **Book.xml** document and open it in Internet Explorer. Figure 14-18 shows how the transformed XML document should appear.
- Close your Web browser window.

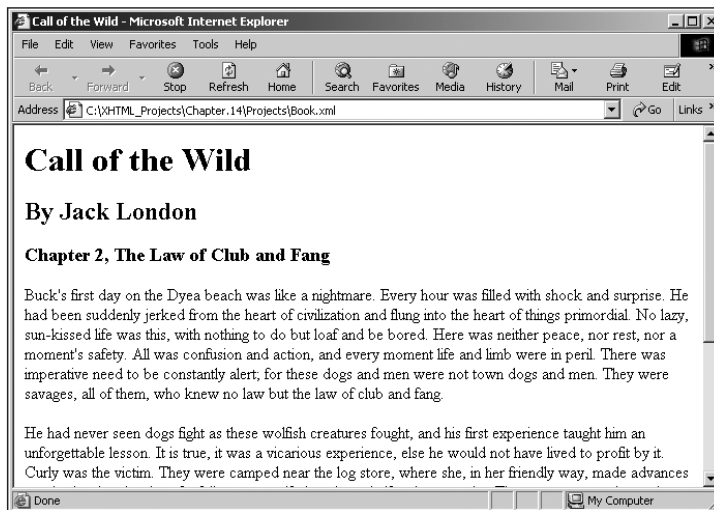


Figure 14-18 Project 14-2



Project 14-3

In this project, you will create an XSL style sheet that uses the `<xsl:apply-templates>` element to format and display an XML document containing information on the world's ten highest mountains. Your Chapter.14\Projects folder contains a file named Mountains.xml that you can use for this project.

- Create a new document in your text editor.
- Type the following elements that form the basis of an XSLT style sheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>World's Highest Mountains</title>
```

```

<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
</head>
<body>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

```

3. Use an `<xsl:apply-templates>` element to display the Mountains.xml document as shown in Figure 14-19.

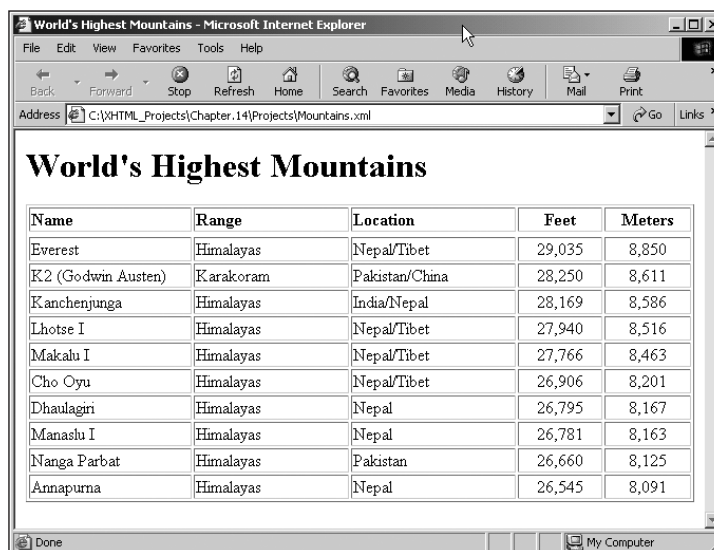


Figure 14-19 Project 14-3

14

4. Save the XSLT style sheet as **Mountains.xsl** in your Chapter.14\Projects folder.
5. Open the **Mountains.xml** document from your Chapter.14\Projects folder in your text editor and add the appropriate statements to give the document access to the Mountains.xsl style sheet:
6. Save the **Mountains.xml** document and open it in Internet Explorer. Your Web browser should resemble Figure 14-19.
7. Close your Web browser window.



Project 14-4

In this project, you will create an XSL style sheet that uses the `<xsl:for-each>` element to format and display an XML document containing the names of the world's 10 busiest airports along with average numbers of passengers. Your Chapter.14\Projects folder contains a file named Airports.xml that you can use for this project.

666 Chapter 14 Extensible Stylesheet Language (XSL)

1. Create a new document in your text editor.
2. Type the following elements that form the basis of an XSLT style sheet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="/">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en" dir="ltr">
<head>
<title>World's 10 Busiest Airports</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
</head>
<body>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

3. Use an `<xsl:for-each>` element to display the Airports.xml document as shown in Figure 14-20.

City	Name	ID	Passengers
Atlanta	Hartsfield	ATL	75,849,375
Chicago	O'Hare	ORD	66,805,339
Los Angeles	Los Angeles International Airport	LAX	61,024,541
London	Heathrow	LHR	60,743,154
Tokyo	Haneda	HND	58,692,688
Dallas/Ft. Worth	Dallas/Ft. Worth International Airport	DFW	55,150,689
Frankfurt	Frankfurt-Main	FRA	8,559,980
Paris	Charles de Gaulle	CDG	47,996,223
Amsterdam	Schiphol	AMS	39,538,483
Denver	Denver International Airport	DEN	36,086,751

Figure 14-20 Project 14-4

4. Save the XSLT style sheet as **Airports.xsl** in your Chapter.14\Projects folder.
5. Open the **Airports.xml** document from your Chapter.14\Projects folder in your text editor and add the appropriate statements to give the document access to the Airports.xsl style sheet:

6. Save the **Airports.xml** document and open it in Internet Explorer. Your Web browser should resemble Figure 14-20.
7. Close your Web browser window.

CASE PROJECTS

For the following projects, save the files you create in the Chapter.14\Cases folder. Be sure to validate the files you create with the W3C Markup Validation Service.



Project 14-1

Create an XML document that contains elements you would find in a business memo. Use `<memo>` as the root element. Add a `date` attribute to the root element that is assigned the date the memo was written. Include elements such as sender, recipient, subject, salutation, and paragraph. The XML document should contain multiple `<paragraph>` elements. Add whatever you like as the content of each element. Create an XSLT style sheet that formats the XML document for display in a Web browser. Save the XML document as Memo.xml and the XSLT template as Memo.xsl.



Project 14-2

Create an XML document that contains elements you would find in a resume. Use `<resume>` as the root element. Include elements such as your name and the position desired. Use an `<employer_name>` element to contain information about each employer. The `<employer_name>` element should include two attributes, `start_date` and `end_date` for the employment period. Create any other nested elements that you deem appropriate, such as `<special_skills>`. Use your own employment and educational experience as the content of the elements and be sure to include multiple elements for former employers, education, and references. If you do not have a great deal of employment experience, make something up. Create an XSLT style sheet that formats the XML document for display in a Web browser. Save the XML document as Resume.xml and the XSLT template as Resume.xsl.



Project 14-3

Create an accounts receivable XML document. Use `<accts_receivable>` as the root element and `<payment>` as the top-level elements beneath the root element. Beneath each `<payment>` element, include elements such as `<name>`, `<date>`, and `<amount>`. Within the `<vendor>` element, include a `pay_method` attribute that you can assign one of the following types of payment options: check, credit card, or cash. For check payments, include a `<check_number>` element beneath the `<payment>` element. For credit card payments, include `<card_name>` and `<card_number>` elements beneath the `<payment>` element. Make up some data and be sure to use at least one of the three payment methods. Create an XSLT style sheets that formats the XML document for display in a Web browser by payment type (check, credit card, or cash). Also, sort the payments by vendor name. Save the XML document as AccountsReceivable.xml and the XSLT template as AccountsReceivable.xsl.

