5

XSL: Extensible Stylesheet Language Transformations (XSLT)

Objectives

- To understand what the Extensible Stylesheet Language is and how it relates to XML.
- To understand what an Extensible Stylesheet Language Transformation (XSLT) is.
- To write XSLT documents.
- To write templates.
- To iterate through a node set.
- To sort data during a transformation.
- To perform conditional processing.
- To declare variables.
- To combine multiple style sheets.

Guess if you can, choose if you dare.
Pierre Corneille

A Mighty Maze! but not without a plan.
Alexander Pope

Behind the outside pattern
the dim shapes get clearer every day.
It is always the same shape, only very numerous.
Charlotte Perkins Gilman

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

Outline

5.1 Introduction
5.2 Applying XSLTs with Java
5.3 Simple Transformation Example
5.4 Creating Elements and Attributes
5.5 Iteration and Sorting
5.6 Conditional Processing
5.7 Combining Style Sheets
5.8 Summary
5.9 Internet and World Wide Web Resources

5.1 Introduction

The *Extensible Stylesheet Language* (*XSL*) provides rules for formatting XML documents. In this chapter, we present *XSL Transformations* (*XSLT*). XSLT *transforms* an XML document into another text-based form, such as the Extensible HyperText Markup Language (XHTML).[1] In this chapter, we present many XSLT examples and show the results of transforming XML documents into XHTML and XML.

5.2 Applying XSLTs with Java

To process XSLT documents, an *XSLT processor* is required. We have created a Java program that uses the JAXP[2] library's XSLT processor to perform XSL transformations. Our program, **Transform.java**[3] (Fig. 5.1), takes as command-line arguments the name of the XML document to be transformed, the name of the XSLT document to apply and the name of the document that will contain the result of the transformation.

```
1  // Fig. 5.1 : Transform.java
2  // Performs XSL Transformations.
3  package com.deitel.jws1.xml;
4
5  // Java core libraries
6  import java.io.*;
7  import java.util.*;
8
9  // Java standard extensions
10 import javax.xml.parsers.*;
```

Fig. 5.1 Java application that performs XSL transformations. (Part 1 of 2.)

1. XHTML has replaced the HyperText Markup Language (HTML) as the primary means of describing Web content. XHTML provides more robust, richer and more extensible features than HTML. For more on XHTML/HTML, visit **www.w3.org/markup**.
2. In this chapter, we use the reference implementation for the Java API for XML Processing 1.2 (JAXP), which is part of the Java Web Services Developer Pack 1.0.
3. Before running the examples in this chapter, execute the batch file (**jws1_xml.bat**) provided with the book's examples.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
11  import javax.xml.transform.*;
12  import javax.xml.transform.dom.*;
13  import javax.xml.transform.stream.*;
14
15  // third-party libraries
16  import org.w3c.dom.*;
17  import org.xml.sax.SAXException;
18
19  public class Transform {
20
21     // execute application
22     public static void main( String args[] ) throws Exception
23     {
24        if ( args.length != 3 ) {
25           System.err.println( "Usage: java Transform input.xml "
26              + "input.xsl output" );
27           System.exit( 1 );
28        }
29
30        // factory for creating DocumentBuilders
31        DocumentBuilderFactory builderFactory =
32           DocumentBuilderFactory.newInstance();
33
34        // factory for creating Transformers
35        TransformerFactory transformerFactory =
36           TransformerFactory.newInstance();
37
38        DocumentBuilder builder =
39           builderFactory.newDocumentBuilder();
40
41        Document document = builder.parse( new File( args[ 0 ] ) );
42
43        // create DOMSource for source XML document
44        Source xmlSource = new DOMSource( document );
45
46        // create StreamSource for XSLT document
47        Source xslSource = new StreamSource( new File( args[ 1 ] ) );
48
49        // create StreamResult for transformation result
50        Result result = new StreamResult( new File( args[ 2 ] ) );
51
52        // create Transformer for XSL transformation
53        Transformer transformer =
54           transformerFactory.newTransformer( xslSource );
55
56        // transform and deliver content to client
57        transformer.transform( xmlSource, result );
58     }
59  }
```

Fig. 5.1 Java application that performs XSL transformations. (Part 2 of 2.)

Lines 6–39 **import** the necessary classes and create a **DocumentBuilderFactory**, a **TransformerFactory** and a **DocumentBuilder**. Line 41 creates a tree structure in memory from the XML document passed as a command-line argument. Line

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

44 creates a new ***DOMSource***[4] object from which the **Transformer** reads the XML **Document** referenced by **document**. Line 47 creates a **StreamSource** object from which the **Transformer** reads the XSL **File** passed as the second command-line argument. Line 50 creates a **StreamResult** object to which the **Transformer** writes a **File** containing the result of the transformation. Lines 53–54 call method ***newTransformer***, passing it the object referenced by **xslSource**. Constructing a **Transformer** that references an XSLT document allows the **Transformer** to apply the rules in that file when transforming XML documents. Line 57 calls method ***transform*** to apply the XSL document referenced by the **Transformer** to the XML document referenced by **xmlSource**. The transformation results are written to the **File** referenced by **result**.

5.3 Simple Transformation Example

In an XSL transformation, there are two trees of nodes. The first node tree is the *source tree*. The nodes in this tree correspond to the original XML document to which the transformation is applied. The document used as the source tree is not modified by an XSL transformation. The second tree is the *result tree*, which contains all of the nodes produced by the XSL transformation. The result tree represents the document produced by the transformation. The XSLT document shown in Fig. 5.2 transforms **introduction.xml** (Fig. 5.3) into a simple XHTML document (Fig. 5.4).

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.2 : introduction.xsl                  -->
4  <!-- Simple XSLT document for introduction.xml. -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8     xmlns = "http://www.w3.org/1999/xhtml">
9     <xsl:output method = "xml" omit-xml-declaration = "no"
10        doctype-system =
11           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
12        doctype-public =
13           "-//W3C//DTD XHTML 1.0 Strict//EN" />
14
15    <xsl:template match = "myMessage">
16       <html>
17          <head><title>Welcome</title></head>
18          <body><p><xsl:value-of select = "message"/></p></body>
19       </html>
20    </xsl:template>
21
22 </xsl:stylesheet>
```

Fig. 5.2 Simple XSLT document for transforming **introduction.xml** into XHTML.

4. In this example, we use a **DOMSource** object to provide the XML document to the **Transformer**, although we could have used a **StreamSource** to read from the XML file directly. We chose this method because, in previous examples in the book, we have built the DOM tree in memory, while we do not have a file from which to create a **StreamSource** object.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

An XSLT document is an XML document with a root element ***stylesheet***.[5] XSLT elements belong to the namespace ***http://www.w3.org/1999/XSL/Transform***, which typically is bound to the namespace prefix ***xsl***.

Line 6 contains the **stylesheet** root element. Attribute ***version*** specifies the version of XSLT to which this document conforms.[6] Namespace prefix **xsl** is defined and is bound to the XSLT URI defined by the W3C. When processed, the ***output*** element in lines 9–13 writes a document type declaration to the result tree. Attribute ***method*** is assigned the value **"xml"**, which indicates that XML is being output to the result tree. Attribute ***omit-xml-declaration*** is assigned the value **"no"**, which causes an XML declaration to be output to the result tree. Attributes ***doctype-system*** and ***doctype-public*** write the **DOCTYPE** DTD information to the result tree.

XSLT uses *XPath* expressions to locate nodes in an XML document.[7] [*Note*: A structured complete explanation of the XPath language is beyond the scope of this book. Instead, we explain XPath expressions as we encounter them in the examples.]

XSL documents contain one or more *templates* that describe how a specific node should be transformed. Line 15 contains a ***template*** *element*, which matches specific XML document nodes by using an XPath pattern in attribute ***match***. In this case, any **Element** nodes with the name **myMessage** are matched.

Lines 16–19 are the contents of the **template** element. When a **myMessage** element node is **match**ed in the source tree (i.e., the document being transformed), the contents of the **template** element are placed in the result tree (i.e., the document created by the transformation). By using element ***value-of*** and an XPath expression in attribute ***select***, the text contents of the node returned by the XPath expression are placed in the result tree. In line 18, element **message**'s value is written to the result tree. Figure 5.3 lists the input XML document. Figure 5.4 lists the results of the transformation. [*Note*: Fig. 5.4 has been modified for the purpose of presentation.]

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.3 : introduction.xml       -->
4  <!-- Simple introduction to XML markup -->
5
6  <myMessage>
7     <message>Welcome to XSLT!</message>
8  </myMessage>
```

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_02_03/introduction.xml chapter5/fig05_02_03/introduc-
tion.xsl chapter5/fig05_02_03/results.html
```

Fig. 5.3 Sample input XML document **introduction.xml**.

5. Although infrequently used, ***transform*** also may be used as the root element in an XSLT document.
6. XSLT 1.0 is a W3C Recommendation (**www.w3.org/TR/xslt**).
7. XML Path Language 1.0 is a W3C Recommendation (**www.w3.org/TR/xpath**).

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <html xmlns = "http://www.w3.org/1999/xhtml">
6     <head><title>Welcome</head></title>
7     <body>
8        <p>Welcome to XSLT!</p>
9     </body>
10 </html>
```

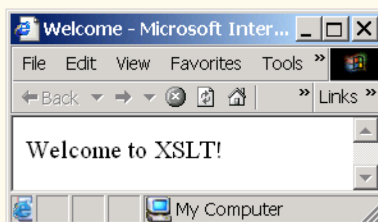

Fig. 5.4 Results of applying **introduction.xsl** to **introduction.xml**.

5.4 Creating Elements and Attributes

When transforming an XML document into another text-based format, it is often necessary to create element nodes and attribute nodes in the result tree. Figure 5.5 provides an XML document that marks up a list of various sports, and Fig. 5.6 presents the XSLT document that transforms the XML document in Fig. 5.5 into another XML document. The original XML document is transformed into a new XML document with the sport names as elements (instead of attributes, as in the original XML document).

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.5 : games.xml -->
4  <!-- Sports Database      -->
5
6  <sports>
7
8     <game title = "cricket">
9        <id>243</id>
10
11       <paragraph>
12          More popular among commonwealth nations.
13       </paragraph>
14    </game>
15
16    <game title = "baseball">
17       <id>431</id>
18
```

Fig. 5.5 XML document containing a list of sports. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
19          <paragraph>
20             More popular in America.
21          </paragraph>
22       </game>
23
24       <game title = "soccer">
25          <id>123</id>
26
27          <paragraph>
28             Most popular sport in the world.
29          </paragraph>
30       </game>
31
32    </sports>
```

Fig. 5.5 XML document containing a list of sports. (Part 2 of 2.)

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.6 : games.xsl                -->
4  <!-- Using xsl:element and xsl:attribute -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <!-- match sports elements -->
10    <xsl:template match = "/sports">
11       <sports>
12          <xsl:apply-templates select = "game"/>
13       </sports>
14    </xsl:template>
15
16    <!-- match game elements -->
17    <xsl:template match = "game">
18
19       <!-- create child element -->
20       <xsl:element name = "{@title}">
21
22          <!-- create attribute -->
23          <xsl:attribute name = "id">
24             <xsl:value-of select = "id"/>
25          </xsl:attribute>
26
27          <comment>
28             <xsl:value-of select = "paragraph"/>
29          </comment>
30
31       </xsl:element>
32    </xsl:template>
33
34 </xsl:stylesheet>
```

Fig. 5.6 Using XSLT to create elements and attributes. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_05_06/games.xml chapter5/fig05_05_06/games.xsl
chapter5/fig05_05_06/results.xml
```

Fig. 5.6 Using XSLT to create elements and attributes. (Part 2 of 2.)

The template in lines 10–14 **match** the root element **sports**. The **/** *pattern* specifies the source tree's *document root* (i.e., the parent of the root element). Element ***apply-templates*** (line 12) specifies that the **template** for the node corresponding to the **game** element (lines 17–32) should be applied. The **sports** element is output to the result tree.

Line 20 contains the element ***element***, which creates an element in the result tree, with the name specified in attribute ***name***. Therefore, the name of this XML element will be the name of the sport contained in element **game**'s **title** attribute. The **@** *pattern* specifies an attribute. *Braces*, **{ }**, enclose the pattern, because **@** is otherwise an illegal character in an element name. Attribute **name** is assigned a *literal result value* (i.e., the value assigned to **name** is the name of the element in the result tree). Braces indicate that the **@title** should not be taken literally for the element name.

Lines 23–25 contain element ***attribute***, which creates an attribute for its parent element in the result tree. Attribute **name** provides the name of the attribute. The text in element **attribute** specifies the attribute's value in the result tree. In this example, the attribute **id** is created, which is assigned the **id** element's character data. Lines 27–29 create the element **comment** in the result tree. Element **paragraph**'s character data are written to the result tree's **comment** element in lines 27–29. Figure 5.7 lists the output of the transformation. [*Note*: The output has been modified for the purpose of presentation.]

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2  <sports>
3
4     <cricket id = "243">
5        <comment>
6           More popular among commonwealth nations.
7        </comment>
8     </cricket>
9
10    <baseball id = "431">
11       <comment>
12          More popular in America.
13       </comment>
14    </baseball>
15
16    <soccer id = "123">
17       <comment>
18          Most popular sport in the world.
19       </comment>
20    </soccer>
21
22 </sports>
```

Fig. 5.7 Results of applying **games.xsl** to **games.xml**.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

5.5 Iteration and Sorting

XSLT allows for iteration through a *node set* (i.e., all nodes that an XPath expression matches). Node sets also can be sorted in XSLT. Figure 5.8 presents an XML document (**sorting.xml**) that contains information about a book. Figure 5.9 presents the XSLT document (**sorting.xsl**) that transforms **sorting.xml** to XHTML. Line 16 (Fig. 5.9) specifies a **template** that **match**es element **book**.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.8: sorting.xml                    -->
4  <!-- XML document containing book information -->
5
6  <book isbn = "999-99999-9-X">
7     <title>Russ Tick&apos;s XML Primer</title>
8
9     <author>
10       <firstName>Russ</firstName>
11       <lastName>Tick</lastName>
12    </author>
13
14    <chapters>
15       <frontMatter>
16          <preface pages = "2" />
17          <contents pages = "5" />
18          <illustrations pages = "4" />
19       </frontMatter>
20
21       <chapter number = "3" pages = "44">
22          Advanced XML</chapter>
23
24       <chapter number = "2" pages = "35">
25          Intermediate XML</chapter>
26
27       <appendix number = "B" pages = "26">
28          Parsers and Tools</appendix>
29
30       <appendix number = "A" pages = "7">
31          Entities</appendix>
32
33       <chapter number = "1" pages = "28">
34          XML Fundamentals</chapter>
35    </chapters>
36
37    <media type = "CD" />
38 </book>
```

Fig. 5.8 XML document containing information about a book.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
```

Fig. 5.9 XSLT document for transforming **sorting.xml**. (Part 1 of 3.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
3  <!-- Fig. 5.9: sorting.xsl                              -->
4  <!-- Transformation of book information into XHTML -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8     xmlns = "http://www.w3.org/1999/xhtml">
9     <!-- write XML declaration and DOCTYPE DTD information -->
10    <xsl:output method = "xml" omit-xml-declaration = "no"
11       doctype-system =
12          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
13       doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
14
15    <!-- match book -->
16    <xsl:template match = "/book">
17    <html>
18       <head>
19          <title>ISBN <xsl:value-of select = "@isbn" /> -
20             <xsl:value-of select = "title" /></title>
21       </head>
22
23       <body>
24          <h1 style = "color: blue">
25             <xsl:value-of select = "title"/></h1>
26
27          <h2 style = "color: blue">by <xsl:value-of
28             select = "author/lastName" />,
29             <xsl:value-of select = "author/firstName" /></h2>
30
31          <table style =
32             "border-style: groove; background-color: wheat">
33
34             <xsl:for-each select = "chapters/frontMatter/*">
35                <tr>
36                   <td style = "text-align: right">
37                      <xsl:value-of select = "name()" />
38                   </td>
39
40                   <td>
41                     ( <xsl:value-of select = "@pages" /> pages )
42                   </td>
43                </tr>
44             </xsl:for-each>
45
46             <xsl:for-each select = "chapters/chapter">
47                <xsl:sort select = "@number" data-type = "number"
48                     order = "ascending" />
49                <tr>
50                   <td style = "text-align: right">
51                      Chapter <xsl:value-of select = "@number" />
52                   </td>
53
54                   <td>
55                      ( <xsl:value-of select = "@pages" /> pages )
```

Fig. 5.9 XSLT document for transforming **sorting.xml**. (Part 2 of 3.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
56                      </td>
57                   </tr>
58                </xsl:for-each>
59
60                <xsl:for-each select = "chapters/appendix">
61                   <xsl:sort select = "@number" data-type = "text"
62                        order = "ascending" />
63                   <tr>
64                      <td style = "text-align: right">
65                         Appendix <xsl:value-of select = "@number" />
66                      </td>
67
68                      <td>
69                         ( <xsl:value-of select = "@pages" /> pages )
70                      </td>
71                   </tr>
72                </xsl:for-each>
73             </table>
74
75             <p style = "color: blue">Pages:
76                <xsl:variable name = "pagecount"
77                   select = "sum(chapters//*/@pages)" />
78                <xsl:value-of select = "$pagecount" />
79             <br />Media Type:
80                <xsl:value-of select = "media/@type" /></p>
81          </body>
82       </html>
83       </xsl:template>
84
85    </xsl:stylesheet>
```

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_08_09/sorting.xml chapter5/fig05_08_09/sorting.xsl
chapter5/fig05_08_09/results.html
```

Fig. 5.9 XSLT document for transforming **`sorting.xml`**. (Part 3 of 3.)

Lines 19–20 create the title for the XHTML document. The value of the source tree's **`isbn`** attribute and the contents of the source tree's **`title`** element are combined to create the title string **ISBN 999-99999-9-X - Russ Tick's XML Primer**.

Lines 27–29 write a header element that contains the book's author to the result tree. Because the *context node* (i.e., the current node being processed) is **`book`**, the XPath expression **`author/lastName`** selects the author's last name from the source tree, and the expression **`author/firstName`** selects the author's first name from the source tree.

Line 34 selects each element (indicated by an asterisk pattern) that is a child of element **`frontMatter`** (which is a child of **`chapters`**). Iteration in XSLT is performed with the ***`for-each`*** element. Line 37 calls *node-set function* ***`name`*** to retrieve the current node's element name (e.g., **`preface`**). The current node is the context node specified in the **`for-each`** (line 34).

Lines 47–48 sort **`chapter`**s by number in ascending order, using element ***`sort`***. Attribute **`select`** selects the value of context node **`chapter`**'s **`number`** attribute. Attribute

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

data-type with value **"number"** specifies a numeric sort, and attribute ***order*** specifies **"ascending"** order. Attribute **data-type** also can be assigned the value ***"text"*** (line 61), and attribute **order** may be assigned the value ***"descending"***.

Lines 76–77 use an *XSL variable* to store the value of the book's page count and output it to the result tree. Attribute **name** specifies the variable's name, and attribute **select** assigns it a value. Function ***sum*** totals all **page** attribute values. The two slashes between **chapters** and ***** indicate that all descendent nodes of **chapters** are searched for elements that contain an attribute named **pages**. A variable's value is retrieved by preceding the variable name with a dollar sign (line 78).

Figure 5.10 shows the results of the transformation. Notice that the chapters and appendices appear in the correct order, even though the XML document contained their corresponding elements in a different order.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4  <html xmlns = "http://www.w3.org/1999/xhtml">
5     <head>
6        <title>ISBN 999-99999-9-X - Russ Tick's XML Primer</title>
7     </head>
8     <body>
9        <h1 style="color: blue">Russ Tick's XML Primer</h1>
10       <h2 style="color: blue">by Tick, Russ</h2>
11       <table style="border-style: groove; background-color: wheat">
12          <tr>
13             <td style="text-align: right">preface</td>
14             <td>( 2 pages )</td>
15          </tr>
16          <tr>
17             <td style="text-align: right">contents</td>
18             <td>( 5 pages )</td>
19          </tr>
20          <tr>
21             <td style="text-align: right">illustrations</td>
22             <td>( 4 pages )</td>
23          </tr>
24          <tr>
25             <td style="text-align: right">Chapter 1</td>
26             <td>( 28 pages )</td>
27          </tr>
28          <tr>
29             <td style="text-align: right">Chapter 2</td>
30             <td>( 35 pages )</td>
31          </tr>
32          <tr>
33             <td style="text-align: right">Chapter 3</td>
34             <td>( 44 pages )</td>
35          </tr>
36          <tr>
```

Fig. 5.10 Results of applying **sorting.xsl** to **sorting.xml**. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
37                <td style="text-align: right">Appendix A</td>
38                <td>( 7 pages )</td>
39             </tr>
40             <tr>
41                <td style="text-align: right">Appendix B</td>
42                <td>( 26 pages )</td>
43             </tr>
44          </table>
45          <p style="color: blue">Pages: 151<br />Media Type:
46             CD</p>
47       </body>
48    </html>
```

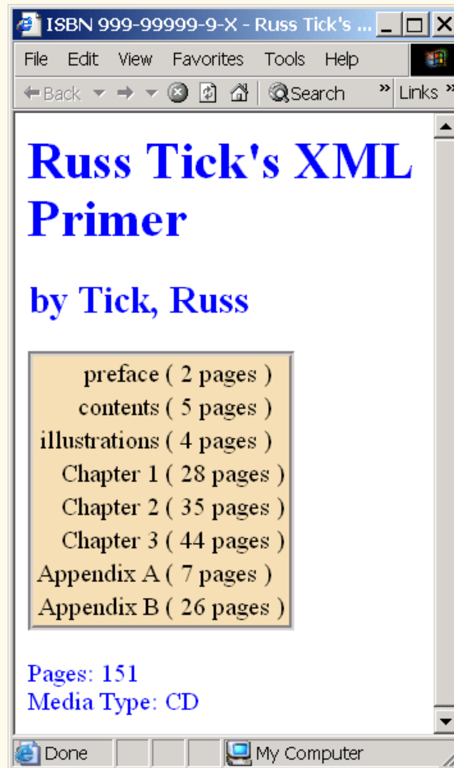

Fig. 5.10 Results of applying **sorting.xsl** to **sorting.xml**. (Part 2 of 2.)

5.6 Conditional Processing

In the previous section, we discussed iterating through a node set, using **for-each**. XSLT also provides elements for conditional processing, such as **if** elements. Figure 5.11 is an XML document that a day-planner application might use. Figure 5.12 is an XSLT document (**conditional.xsl**) for transforming the day-planner XML document (**planner.xml**) into an XHTML document. This style sheet demonstrates some of XSLT's conditional-processing capabilities.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.11 : planner.xml  -->
4  <!-- Day Planner XML document -->
5
6  <planner>
7
8     <year value = "2002">
9
10       <date month = "7" day = "15">
11          <note time = "1430">Doctor&apos;s appointment</note>
12          <note time = "1620">Physics class at BH291C</note>
13       </date>
14
15       <date month = "7" day = "4">
16          <note>Independence Day</note>
17       </date>
18
19       <date month = "7" day = "20">
20          <note time = "0900">General Meeting in room 32-A</note>
21       </date>
22
23       <date month = "7" day = "20">
24          <note time = "1900">Party at Joe&apos;s</note>
25       </date>
26
27       <date month = "7" day = "20">
28          <note time = "1300">Financial Meeting in room 14-C</note>
29       </date>
30
31       <date month = "7" day = "9">
32          <note />
33       </date>
34    </year>
35
36 </planner>
```

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_11_12/planner.xml chapter5/fig05_11_12/conditional.xsl
chapter5/fig05_11_12/results.html
```

Fig. 5.11 Day-planner XML document.

XSLT provides the ***choose*** element (lines 50–80) to allow alternative conditional statements, similar to an **if**/**else** structure in Java. Element ***when*** corresponds to a Java **if** statement. The ***test*** attribute of the **when** element specifies the expression that is being tested. If the expression is true, the XSLT processor evaluates the markup inside the **when** element. Lines 52–53, for instance, provide an expression that evaluates to true when the value of the source tree's **time** attribute is between **"0500"** and **"1200"**. The element **choose** serves to group all the ***when*** elements, thereby making them exclusive of one another (i.e., the first **when** element whose conditional statement is satisfied will be

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

executed). The element ***otherwise*** (lines 76–78) corresponds to Java's **else** statement in an **if**/**else** structure and is optional.

Lines 84–86 compose the ***if*** element. This **if** element determines whether the current node (represented as **.**) being processed is empty. If so, **n/a** is inserted into the result tree. Unlike element **choose**, element **if** provides a single conditional test.

5.7 Combining Style Sheets

XSLT allows for modularity in style sheets. This feature enables XSLT documents to use other XSLT documents. Figure 5.13 lists an XSLT document that is imported into the XSLT document in Fig. 5.15, using element ***import***.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.12 : conditional.xsl             -->
4  <!-- xsl:choose, xsl:when and xsl:otherwise. -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8        xmlns = "http://www.w3.org/1999/xhtml">
9     <xsl:output method = "xml" omit-xml-declaration = "no"
10        doctype-system =
11           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
12        doctype-public =
13           "-//W3C//DTD XHTML 1.0 Strict//EN" />
14
15    <xsl:template match = "/">
16       <html>
17          <head><title>Conditional Processing</title></head>
18          <body>
19             <p>Appointments
20                <br />
21                <xsl:apply-templates select = "planner/year" />
22             </p>
23          </body>
24       </html>
25    </xsl:template>
26
27    <xsl:template match = "year">
28       <strong>Year:</strong>
29
30       <xsl:value-of select = "@value" />
31
32       <br />
33
34       <xsl:for-each select = "date/note">
35
36          <!-- sort by date's day attribute value -->
37          <xsl:sort select = "../@day" order = "ascending"
38             data-type = "number" />
39
40          <br />
```

Fig. 5.12 Using conditional elements. (Part 1 of 3.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
41
42             <strong>
43                Day:
44                <xsl:value-of select = "../@month"/>/
45                <xsl:value-of select = "../@day"/>
46             </strong>
47
48             <br />
49
50             <xsl:choose>
51
52                <xsl:when test =
53                   "@time &gt; '0500' and @time &lt; '1200'">
54
55                   Morning (<xsl:value-of select = "@time" />):
56                </xsl:when>
57
58                <xsl:when test =
59                   "@time &gt; '1200' and @time &lt; '1700'">
60
61                   Afternoon (<xsl:value-of select = "@time" />):
62                </xsl:when>
63
64                <xsl:when test =
65                   "@time &gt; '1700' and @time &lt; '2200'">
66
67                   Evening (<xsl:value-of select = "@time" />):
68                </xsl:when>
69
70                <xsl:when test =
71                   "@time &gt; '2200' and @time &lt; '500'">
72
73                   Night (<xsl:value-of select = "@time" />):
74                </xsl:when>
75
76                <xsl:otherwise>
77                   Entire day:
78                </xsl:otherwise>
79
80             </xsl:choose>
81
82             <xsl:value-of select = "." />
83
84             <xsl:if test = ". = ''">
85                n/a
86             </xsl:if>
87
88             <br />
89          </xsl:for-each>
90
91       </xsl:template>
92
93    </xsl:stylesheet>
```

Fig. 5.12 Using conditional elements. (Part 2 of 3.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

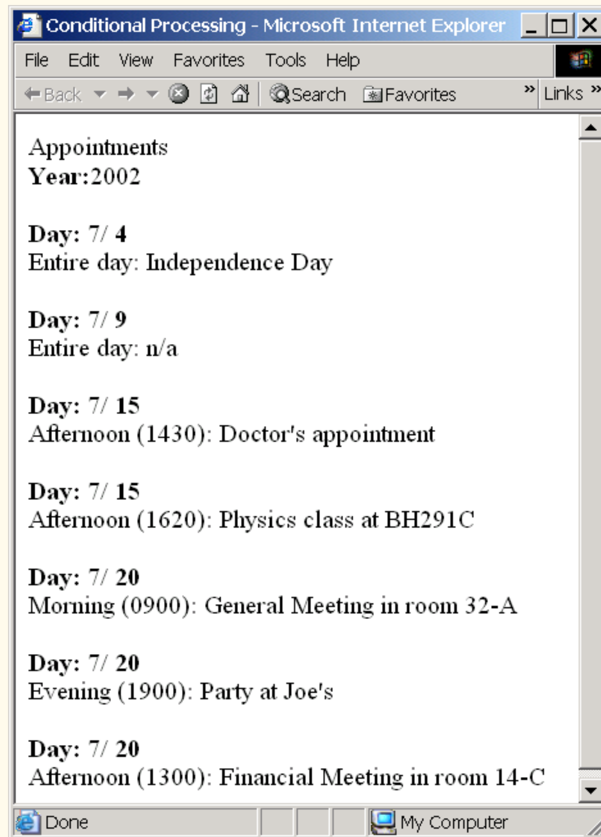

Fig. 5.12 Using conditional elements. (Part 3 of 3.)

```
1   <?xml version = "1.0" encoding = "UTF-8"?>
2
3   <!-- Fig. 5.13 : style.xsl -->
4   <!-- xsl:import example    -->
5
6   <xsl:stylesheet version = "1.0"
7      xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8      xmlns = "http://www.w3.org/1999/xhtml">
9      <xsl:output method = "xml" omit-xml-declaration = "no"
10         doctype-system =
11            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
12         doctype-public =
13            "-//W3C//DTD XHTML 1.0 Strict//EN" />
14
15     <xsl:template match = "book">
16        <html>
17           <head><title>Combining Style Sheets</title></head>
```

Fig. 5.13 XSLT document being imported. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
18            <body>
19               <xsl:apply-templates />
20            </body>
21         </html>
22      </xsl:template>
23
24      <xsl:template match = "title">
25         <xsl:value-of select = "." />
26      </xsl:template>
27
28      <xsl:template match = "author">
29
30         <p>Author:
31            <xsl:value-of select = "lastName" />,
32            <xsl:value-of select = "firstName" />
33         </p>
34
35      </xsl:template>
36
37      <x\sl:template match = "*|text()"/>
38
39   </xsl:stylesheet>
```

Fig. 5.13 XSLT document being imported. (Part 2 of 2.)

The XSLT recommendation defines *default templates*. If a programmer does not specify a **template** that **match**es a particular element, the default XSLT template is applied. Figure 8.14 describes some default templates.

In line 19, the **apply-templates** element indicates that the default **template**s should be applied. If a more specific template for a node exists in the XSLT document, it is applied instead of the corresponding default template. For example, the **template**s on lines 15, 24 and 28 **match** specific elements, and the **template** on line 37 **match**es any element node or text node. The **template** in line 37 does not perform any action other than overriding the default **template** for element nodes and text nodes, which display the element nodes' character data. In this example, we do not want the default template to output every element's character data: We want only the book title and author name, not the chapter and appendix information.

Template / Description
`<xsl:template match = "/	*">` ` <xsl:apply-templates/>` `</xsl:template>` This **template** **match**es and applies **template**s to the child nodes of the document root (**/**) and any element nodes (*****).

Fig. 5.14 Some default XSLT templates. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

Template / Description

`<xsl:template match = "text() | @*">`
`   <xsl:value-of select = "."/>`
`</xsl:template>`
This **template** **match**es and outputs the values of text nodes (**text()**) and attribute nodes (**@**).

`<xsl:template match = "processing-instruction() | comment()"/>` This **template** **match**es processing-instruction nodes (**processing-instruction()**) and comment nodes (**comment()**), but does not perform any actions with them.

Fig. 5.14 Some default XSLT templates. (Part 2 of 2.)

Line 9 in Fig. 5.15 ***import***s the **template**s defined in the XSLT document (Fig. 5.13) referenced by attribute **href**. Line 13 provides a **template** for element **title**, which already has been defined in the XSLT document being **import**ed. This *local template* has higher precedence than the **import**ed **template**, so it is used instead of the **import**ed **template**. Figure 5.16 shows the results of the transformation of **sorting.xml** (Fig. 5.8) into XHTML.

Common Programming Error 5.1

When using the ***import*** *element, not placing it as the first child of* ***stylesheet*** *or* ***transform*** *is an error.*

```
 1  <?xml version = "1.0" encoding = "UTF-8"?>
 2
 3  <!-- Fig. 5.15 : importer.xsl            -->
 4  <!-- xsl:import example using style.xsl. -->
 5
 6  <xsl:stylesheet version = "1.0"
 7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
 8
 9     <xsl:import href = "style.xsl" />
10
11     <!-- this template has higher precedence over the -->
12     <!-- templates being imported                     -->
13     <xsl:template match = "title">
14
15        <h2 xmlns = "http://www.w3.org/1999/xhtml">
16           <xsl:value-of select = "." />
17        </h2>
18
19     </xsl:template>
20
21  </xsl:stylesheet>
```

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_13_15/sorting.xml chapter5/fig05_13_15/importer.xsl
chapter5/fig05_13_15/results.html
```

Fig. 5.15 Importing another XSLT document.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml">
5     <head>
6        <title>Combining Style Sheets</title>
7     </head>
8     <body>
9        <h2>Russ Tick's XML Primer</h2>
10       <p>Author: Tick, Russ</p>
11    </body>
12 </html>
```

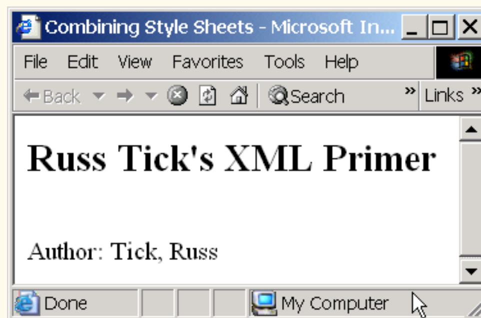

Fig. 5.16 Results of applying **importer.xsl** to **sorting.xml**.

Style sheets can include other style sheets. Figure 5.17 shows an example of the XSLT element ***include***, which merges other XSLT documents in the current XSLT document. Lines 45–46 contain element **include**, which includes the style sheets referenced by attribute **href**. The difference between element **include** and element **import** is that **template**s that are **include**d have the same precedence as the local templates. Therefore, if any **template**s are duplicated, the **template** included replaces the **template** in the document that **include**s the **template**. Notice that **book.xsl** (Fig. 5.17) contains a **template** element for **chapter** in lines 36–43. Figure 5.18 and Fig. 5.19 list the XSLT documents being **include**d by Fig. 5.17. Notice that **chapters.xsl** (Fig. 5.19) contains a **template** element for **chapter** in lines 17–23.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2  
3  <!-- Fig. 5.17 : book.xsl -->
4  <!-- xsl:include example  -->
5  
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8     xmlns = "http://www.w3.org/1999/xhtml">
9     <xsl:output method = "xml" omit-xml-declaration = "no"
10       doctype-system =
11          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
```

Fig. 5.17 Combining style sheets using **xsl:include**. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
12          doctype-public =
13             "-//W3C//DTD XHTML 1.0 Strict//EN" />
14
15    <xsl:template match = "/">
16
17       <html>
18          <head><title>Including Style Sheets</title></head>
19          <body>
20             <xsl:apply-templates select = "book" />
21          </body>
22       </html>
23
24    </xsl:template>
25
26    <xsl:template match = "book">
27
28       <h2>
29          <xsl:value-of select = "title" />
30       </h2>
31
32       <xsl:apply-templates />
33    </xsl:template>
34
35
36    <xsl:template match = "chapter">
37
38       <h3>
39          <em><xsl:value-of select = "." /></em>
40       </h3>
41
42       <xsl:apply-templates />
43    </xsl:template>
44
45    <xsl:include href = "author.xsl" />
46    <xsl:include href = "chapters.xsl" />
47
48    <xsl:template match = "*|text()" />
49
50 </xsl:stylesheet>
```

```
c:\examples>java -classpath %CLASSPATH% com.deitel.jws1.xml.Transform
chapter5/fig05_17_18_19/sorting.xml chapter5/fig05_17_18_19/book.xsl
chapter5/fig05_17_18_19/results.html
```

Fig. 5.17 Combining style sheets using **xsl:include**. (Part 2 of 2.)

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.18 : author.xsl -->
4  <!-- xsl:include example    -->
5
```

Fig. 5.18 XSLT document for rendering the author's name. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:template match = "author">
10
11       <p xmlns = "http://www.w3.org/1999/xhtml">Author:
12          <xsl:value-of select = "lastName" />,
13          <xsl:value-of select = "firstName" />
14       </p>
15
16    </xsl:template>
17
18 </xsl:stylesheet>
```

Fig. 5.18 XSLT document for rendering the author's name. (Part 2 of 2.)

Figure 5.20 shows the results of applying the XSLT document (Fig. 5.17) to the document that describes a book (Fig. 5.8). The XHTML that is created marks up the chapter names as an unordered list and not as a series of **h3** headers containing emphasized text.

```
1  <?xml version = "1.0" encoding = "UTF-8"?>
2
3  <!-- Fig. 5.19 : chapters.xsl -->
4  <!-- xsl:include example.     -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform"
8     xmlns = "http://www.w3.org/1999/xhtml">
9     <xsl:template match = "chapters">
10       <p>Chapters:</p>
11
12       <ul>
13          <xsl:apply-templates select = "chapter" />
14       </ul>
15    </xsl:template>
16
17    <xsl:template match = "chapter">
18
19       <li>
20          <xsl:value-of select = "." />
21       </li>
22
23    </xsl:template>
24
25 </xsl:stylesheet>
```

Fig. 5.19 XSLT document **chapters.xsl**.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

Fig. 5.20 Result of applying **book.xsl** to **sorting.xml**. (Part 1 of 2.)

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

```
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4   <html xmlns="http://www.w3.org/1999/xhtml">
5      <head>
6         <title>Including Style Sheets</title>
7      </head>
8      <body>
9         <h2>Russ Tick's XML Primer</h2>
10        <p>Author: Tick, Russ</p>
11        <p>Chapters:</p>
12        <ul>
13           <li>Advanced XML</li>
14           <li>Intermediate XML</li>
15           <li>XML Fundamentals</li>
16        </ul>
17     </body>
18  </html>
```

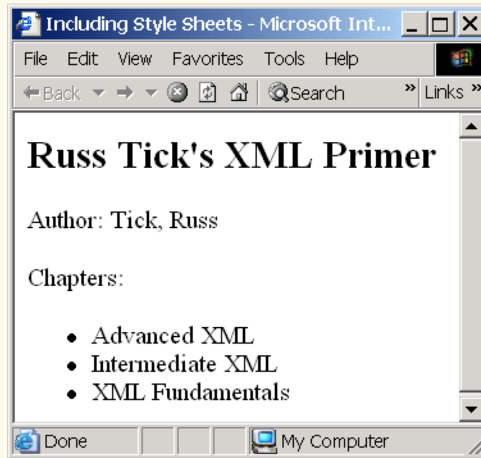

Fig. 5.20 Result of applying **book.xsl** to **sorting.xml**. (Part 2 of 2.)

In this chapter, we have discussed the applicaton of Extensible Stylesheet Language Transformations for transforming XML documents into another text-based format. In Chapter 6, we introduce an XML vocabulary called the *Simple Object Access Protocol* (*SOAP*), which is used for marking up requests and responses so that they can be transferred via protocols such as HTTP. In particular, we discuss several popular platforms for deploying Web services.

5.8 Summary

Extensible Stylesheet Language Transformation (XSLT) is a W3C recommendation for transforming XML documents into other text-based formats. XSLT uses XPath expressions to match nodes when transforming an XML document into a different document. In order to process XSLT documents, an XSLT processor is required.

XSLT documents contain one or more templates that describe how a node is transformed. XSL transformations involve two trees: The source tree, which corresponds to the XML document, and the result tree, which corresponds to the document created as a result

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.

of the transformation. XSLT documents can be reused by either including them or importing them into another XSLT document.

5.9 Internet and World Wide Web Resources

`www.w3.org/Style/XSL`
The W3C Extensible Stylesheet Language Web site.

`www.w3.org/TR/xsl`
The W3C XSL recommendation.

`www.w3schools.com/xsl`
This site features an XSL tutorial, along with a list of links and resources.

`www.dpawson.co.uk/xsl/xslfaq.html`
A comprehensive collection of FAQs on XSL.

`msdn.microsoft.com/xml`
Microsoft Developer Network XML home page, which provides information on XML and XML-related technologies, such as XSL/XSLT.

`xml.apache.org/xalan-j/index.html`
Home page for Apache's XSLT processor Xalan.

`java.sun.com/xml/xml_jaxp.html`
Home page for JAXP, an implementation of XSLT in Java.

© Copyright 1992-2002 by Prentice Hall and Deitel & Associates, Inc. All Rights Reserved.