XSL: Extensible Stylesheet Language Transformations (XSLT)

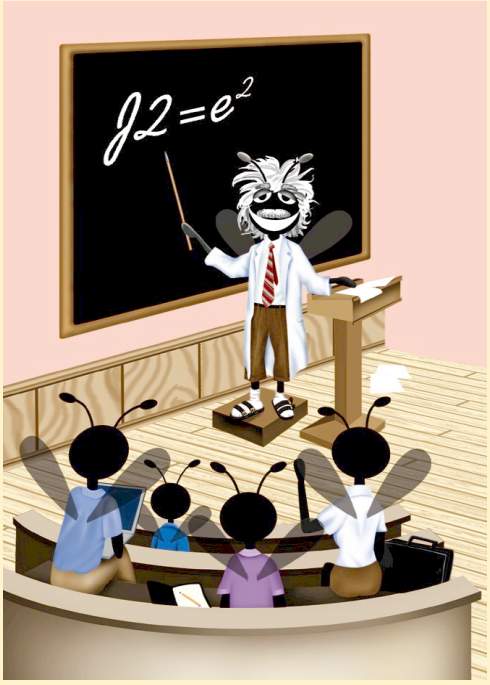
Objectives

- 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 be able to write XSLT documents.
- To be able to write templates.
- To be able to iterate through a node set.
- To be able to sort.
- To be able to perform conditional processing.
- To be able to declare variables.

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



Outline

D.1 Introduction

The *Extensible Stylesheet Language (XSL)* provides rules for formatting XML documents. In this appendix, we present the *XSL Transformation Language (XSLT)*. XSLT *transforms* an XML document to another text-based form, such as XHTML. We present many XSLT examples and show the results of the transformations.

D.2 Applying XSLTs with Java

To process XSLT documents, you will need an *XSLT processor*. We have created a Java program that uses the XSLT processor in the JAXP library to perform XSL transformations. Our program, **`Transform.java`** (Fig. D.1), takes as command-line arguments the name of the XML document to be transformed, the name of the XSLT document and the name of the document that will be created by the transformation.

```
1  // Fig. D.1 : Transform.java
2  // Performs XSL Transformations.
3
4  // Java core libraries
5  import java.io.*;
6  import java.util.*;
7
8  // Java standard extensions
9  import javax.xml.parsers.*;
10 import javax.xml.transform.*;
11 import javax.xml.transform.dom.*;
12 import javax.xml.transform.stream.*;
13
14 // third-party libraries
15 import org.w3c.dom.*;
16 import org.xml.sax.SAXException;
17
```

Fig. D.1 Java application that performs XSL transformations (part 1 of 2).

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

Fig. D.1 Java application that performs XSL transformations (part 2 of 2).

Lines 1–38 import the necessary classes and create a **`DocumentBuilderFactory`**, a **`DocumentBuilder`** and a **`TransformerFactory`**. Line 40 creates a **`Document`** object, passing it the XML file to be transformed. Line 43 creates a new ***`DOMSource`*** object from which the **`Transformer`** reads the XML **`Document`**. Line 46 creates a **`StreamSource`** object from which the **`Transformer`** reads the XSL **`File`**. **`DOMSource`** and **`StreamSource`** both implement the **`Source`** interface. Line 49 creates a **`StreamResult`** object to which the **`Transformer`** writes a **`File`** containing the result of the transformation. Lines 52–53 call method ***`newTransformer`***, passing it the **`Source`** object of the XSL file as an argument. Constructing a **`Transformer`** with an

XSL file allows the **`Transformer`** to apply the rules in that file when transforming XML documents. Line 56 applies the XSL transformation to the XML file to produce the result **`File`**. In this appendix, you will use this Java application to apply XSL transformations to XML files.

[*Note*: In this example we use a **`DOMSource`** 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 other examples in the book, we build the DOM tree in memory and do not have a file from which to create a **`StreamSource`** object.]

D.3 Templates

An XSLT document is an XML document with a root element ***`stylesheet`***. The namespace for an XSLT document is ***`http://www.w3.org/1999/XSL/Transform`***. The XSLT document shown in Fig. D.2 transforms **`intro.xml`** (Fig. D.3) into a simple XHTML document (Fig. D.4).

XSLT uses *XPath* expressions to locate nodes in an XML document. [*Note*: A structured complete explanation of the XPath language is beyond the scope of this book. We explain each XPath expression as we encounter it in the examples.] 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 second node tree is the *result tree*. The result tree contains all of the nodes produced by the XSL transformation. This result tree represents the document produced by the transformation. The document used as the source tree is not modified by an XSL transformation.

Lines 6–7 contain the XSLT document's root element (i.e., **`xsl:stylesheet`**). Attribute ***`version`*** indicates the XSLT specification used. Namespace prefix ***`xsl`*** is defined and assigned the URI **`"http://www.w3.org/1999/XSL/Transform"`**.

Line 9 contains a ***`template`*** *element*. This element matches specific XML document nodes by using an XPath expression in attribute ***`match`***. In this case, we **`match`** any **`myMessage`** element nodes.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.2 : intro.xsl                -->
4  <!-- Simple XSLT document for intro.xml -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:template match = "myMessage">
10       <html>
11          <body><xsl:value-of select = "message"/></body>
12       </html>
13    </xsl:template>
14
15 </xsl:stylesheet>
```

Fig. D.2 Simple template.

Lines 10–12 are the contents of the **template** element. When a **myMessage** element node is matched 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(s) returned by the XPath expression are placed in the result tree.

Figure D.3 lists the input XML document. Figure D.4 lists the results of the transformation.

D.4 Creating Elements and Attributes

In the previous section, we demonstrated the use of XSLT for simple element matching. This section discusses the creation of new elements and attributes within an XSLT document. Figure D.5 lists an XML document that marks up various sports.

Figure D.6 lists the XSLT document that transforms the XML document in Fig. D.5 into another XML document.

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

Fig. D.3 Sample input XML document **intro.xml**.

```
1  <html>
2  <body>Welcome to XSLT!</body>
3  </html>
```

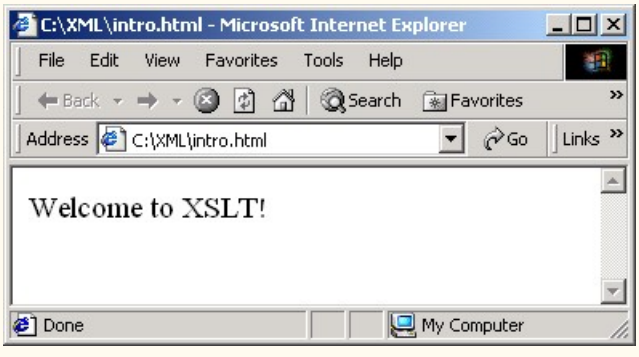


Fig. D.4 Results of XSL transformation.

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

<!-- Fig. D.5 : games.xml  -->
<!-- Sports Database       -->

<sports>

   <game title = "cricket">
      <id>243</id>

      <para>
         More popular among commonwealth nations.
      </para>
   </game>

   <game title = "baseball">
      <id>431</id>

      <para>
         More popular in America.
      </para>
   </game>

   <game title = "soccer">
      <id>123</id>

      <para>
         Most popular sport in the world.
      </para>
   </game>

</sports>
```

Fig. D.5 XML document containing a list of sports.

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

<!-- Fig. D.6 : elements.xsl              -->
<!-- Using xsl:element and xsl:attribute -->

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

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

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

Fig. D.6 Using XSLT to create elements and attributes (part 1 of 2).

```
18
19      <xsl:template match = "game">
20         <xsl:element name = "@title">
21
22            <xsl:attribute name = "id">
23               <xsl:value-of select = "id"/>
24            </xsl:attribute>
25
26            <comment>
27               <xsl:value-of select = "para"/>
28            </comment>
29
30         </xsl:element>
31      </xsl:template>
32
33   </xsl:stylesheet>
```

Fig. D.6 Using XSLT to create elements and attributes (part 2 of 2).

Lines 9–11 use the **match** attribute to select the *document root* (i.e., the parent of the root element) of the XML document. The **/** character represents the document root. Element ***apply-templates*** applies the **template**s to specific nodes. In this case, we have not specified any particular nodes. By default, element **apply-templates** matches all the child nodes of an element. In Fig. D.5, the child nodes of the document root are two comment nodes and the **sports** element node.

The XSLT recommendation defines *default templates* for the nodes of an XML document. If a programmer does not specify a **template** that matches a particular element, the default XSLT template is applied. Default templates are described in Fig. D.7.

Lines 13–17 match element **sports**. We output the **sports** element and apply templates to the child nodes of the **sports** element. Line 19 matches element **game**. In the input XML document, element **game** contains the name of a sport, its unique identifier and a description.

Template / Description

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

This template matches and applies templates to the child nodes of the document root (**/**) and any element nodes (*****).

```
<xsl:template match = "text() | @*">
   <xsl:value-of select = "."/>
</xsl:template>
```

This template matches and outputs the values of text nodes (**text()**) and attribute nodes (**@**).

Fig. D.7 Default XSLT templates (part 1 of 2).

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

Fig. D.7 Default XSLT templates (part 2 of 2).

Line 20 shows the element ***`element`***, which creates an element, with the element name specified in attribute **`name`**. Therefore, the name of this XML element will be the name of the sport contained in attribute **`title`** of element **`game`**. [*Note*: In XPath, the symbol **`@`** specifies an attribute.]

Lines 22–24 show element **`attribute`**, which creates an attribute for an element. Attribute **`name`** provides the name of the attribute. The text in element **`attribute`** specifies the attribute's value in the result tree. Here, the attribute statement will create attribute **`id`** for the new element, which contains the text in element **`id`** of element **`game`**. Lines 26–28 create element **`comment`** with the contents of element **`para`**.

Figure D.8 lists the output of the transformation. Your output may not look exactly like that in the figure, because we have modified the output in the figure for presentation. The original XML document has been transformed into a new XML document with sport names as elements (instead of attributes, as in the original document).

D.5 Iteration and Sorting

XSLT allows for iteration through a node set (i.e., all nodes an XPath expression matches). Node sets can also be sorted in XSLT. Figure D.9 shows an XML document containing information about a book.

```
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>
```

Fig. D.8 Output of transformation (part 1 of 2).

```
20      </soccer>
21
22   </sports>
```

Fig. D.8 Output of transformation (part 2 of 2).

```
1   <?xml version = "1.0"?>
2
3   <!-- Fig. D.9 : usage.xml -->
4   <!-- Book information     -->
5
6   <book isbn = "999-99999-9-X">
7      <title>Deitel&apos;s XML Primer</title>
8
9      <author>
10        <firstName>Paul</firstName>
11        <lastName>Deitel</lastName>
12     </author>
13
14     <chapters>
15        <preface num = "1" pages = "2">Welcome</preface>
16        <chapter num = "2" pages = "2">XML Elements?</chapter>
17        <chapter num = "1" pages = "4">Easy XML</chapter>
18        <appendix num = "1" pages = "9">Entities</appendix>
19     </chapters>
20
21     <media type = "CD"/>
22  </book>
```

Fig. D.9 Book table of contents as XML.

Figure D.10 shows the XSLT style sheet that transforms the document in Fig. D.9.

```
1   <?xml version = "1.0"?>
2
3   <!-- Fig. D.10 : usage.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
9      <xsl:template match = "/">
10        <html>
11           <xsl:apply-templates/>
12        </html>
13     </xsl:template>
14
15     <xsl:template match = "book">
16        <head>
17           <title>ISBN <xsl:value-of select = "@isbn"/> -
18              <xsl:value-of select = "title"/></title>
19        </head>
```

Fig. D.10 Transforming XML data into XHTML (part 1 of 2).

```
20
21          <body>
22             <h1><xsl:value-of select = "title"/></h1>
23
24             <h2>by <xsl:value-of select = "author/lastName"/>,
25                <xsl:value-of select = "author/firstName"/></h2>
26
27             <table border = "1">
28                <xsl:for-each select = "chapters/preface">
29                   <xsl:sort select = "@num" order = "ascending"/>
30                   <tr>
31                      <td align = "right">
32                         Preface <xsl:value-of select = "@num"/>
33                      </td>
34
35                      <td>
36                         <xsl:value-of select = "."/> (
37                         <xsl:value-of select = "@pages"/> pages )
38                      </td>
39                   </tr>
40                </xsl:for-each>
41
42                <xsl:for-each select = "chapters/chapter">
43                   <xsl:sort select = "@num" order = "ascending"/>
44                   <tr>
45                      <td align = "right">
46                         Chapter <xsl:value-of select = "@num"/>
47                      </td>
48
49                      <td>
50                         <xsl:value-of select = "."/> (
51                         <xsl:value-of select = "@pages"/> pages )
52                      </td>
53                   </tr>
54                </xsl:for-each>
55
56                <xsl:for-each select = "chapters/appendix">
57                   <xsl:sort select = "@num" order = "ascending"/>
58                   <tr>
59                      <td align = "right">
60                         Appendix <xsl:value-of select = "@num"/>
61                      </td>
62
63                      <td>
64                         <xsl:value-of select = "."/> (
65                         <xsl:value-of select = "@pages"/> pages )
66                      </td>
67                   </tr>
68                </xsl:for-each>
69             </table>
70          </body>
71       </xsl:template>
72
73    </xsl:stylesheet>
```

Fig. D.10 Transforming XML data into XHTML (part 2 of 2).

Line 15 is an XSLT template that matches the **book** element. In this template, we construct the body of the XHTML document. Lines 17–18 create the title for the XHTML document. We use the ISBN of the book from attribute **isbn** and also the contents of element **title** to create the title string, resulting in **ISBN 999-99999-9-X - Deitel's XML Primer**. Line 22 creates a header element with the title of the book, selected from element **title**. Lines 24–25 create another header element, displaying the name of the book's author. The XPath expression **author/lastName** selects the author's last name, and the expression **author/firstName** selects the author's first name.

Line 28 shows XSLT element ***for-each***, which applies the contents of the element to each of the nodes selected by attribute **select**. In this case, we select all **preface** elements of the **chapters** element. XSLT element **sort** (line 29) sorts the nodes selected by the **for-each** element by the field in attribute **select**. Attribute ***order*** specifies how these nodes should be ordered. Attribute **order** has possible values such as **"ascending"** (i.e., A–Z) and **"descending"** (i.e., Z–A). In this **for-each** element, we sort the nodes by attribute **num**, in ascending order.

Lines 30–39 output a table row displaying the preface number, the title of the preface and the number of pages in that preface for each **preface** element. Similarly, lines 42–54 output the **chapter** elements, and lines 56–68 output the **appendix** elements.

Figure D.11 shows the results of the transformation. Your output may look different, because we have modified ours for presentation. Notice that the chapters have appeared in the correct order, even though the XML document contained the elements in a different order.

```
1  <html>
2     <head>
3        <title>ISBN 999-99999-9-X - Deitel's XML Primer</title>
4     </head>
5
6     <body>
7        <h1>Deitel's XML Primer</h1>
8        <h2>by Deitel, Paul</h2>
9
10       <table border = "1">
11          <tr>
12             <td align = "right">Preface 1</td>
13             <td>Welcome ( 2 pages )</td>
14          </tr>
15
16          <tr>
17             <td align = "right">Chapter 1</td>
18             <td>Easy XML ( 4 pages )</td>
19          </tr>
20
21          <tr>
22             <td align = "right">Chapter 2</td>
23             <td>XML Elements? ( 2 pages )</td>
24          </tr>
25
26          <tr>
27             <td align = "right">Appendix 1</td>
```

Fig. D.11 Output of the transformation (part 1 of 2).

```
28              <td>Entities ( 9 pages )</td>
29           </tr>
30        </table>
31     </body>
32
33  </html>
```

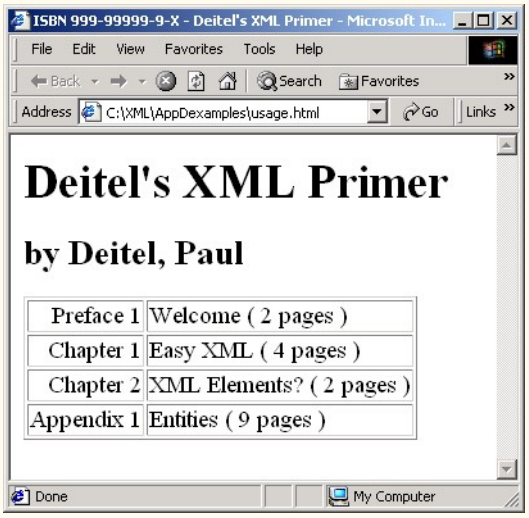


Fig. D.11 Output of the transformation (part 2 of 2).

D.6 Conditional Processing

In the previous section, we discussed the iteration of a node set. XSLT also provides elements to perform conditional processing, such as **`if`** elements. Figure D.12 is an XML document that a day planner application might use.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.12 : planner.xml    -->
4  <!-- Day Planner XML document   -->
5
6  <planner>
7
8     <year value = "2001">
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>
```

Fig. D.12 Day planner XML document (part 1 of 2).

```
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       </year>
32
33    </planner>
```

Fig. D.12 Day planner XML document (part 2 of 2).

Figure D.13 is an XSLT document for transforming the day planner XML document into an XHTML document.

XSLT provides the ***choose*** element (lines 34–60) to allow alternative conditional statements, similar to a nested **if/else** structure in Java. Element ***when*** corresponds to an **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 code inside the **when** element. Line 37, for instance, is an expression that will be true when the **time** 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 is satisfied will be executed). The element **otherwise** (lines 56–58) corresponds to the final **else** statement in a nested **if/else** structure and is likewise optional.

Lines 64–66 show the **if** conditional statement. This **if** determines whether the current node (represented as **.**) is empty. If so, **n/a** is inserted into the result tree. Unlike element **choose**, element **if** provides a single conditional test.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.13 : 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
9     <xsl:template match = "/">
10       <html>
11
12          <body>
13             Appointments
14             <br/>
```

Fig. D.13 Using conditional elements (part 1 of 3).

```
15             <xsl:apply-templates select = "planner/year"/>
16          </body>
17
18       </html>
19    </xsl:template>
20
21    <xsl:template match = "year">
22       <strong>Year: </strong>
23       <xsl:value-of select = "@value"/>
24       <br/>
25       <xsl:for-each select = "date/note">
26          <xsl:sort select = "../@day" order = "ascending"
27             data-type = "number"/>
28          <strong>
29             Day:
30             <xsl:value-of select = "../@day"/>/
31             <xsl:value-of select = "../@month"/>
32          </strong>
33
34          <xsl:choose>
35
36             <xsl:when test =
37                "@time &gt; '0500' and @time &lt; '1200'">
38                Morning (<xsl:value-of select = "@time"/>):
39             </xsl:when>
40
41             <xsl:when test =
42                "@time &gt; '1200' and @time &lt; '1700'">
43                Afternoon (<xsl:value-of select = "@time"/>):
44             </xsl:when>
45
46             <xsl:when test =
47                "@time &gt; '1700' and @time &lt; '2000'">
48                Evening (<xsl:value-of select = "@time"/>):
49             </xsl:when>
50
51             <xsl:when test =
52                "@time &gt; '2000' and @time &lt; '2400'">
53                Night (<xsl:value-of select = "@time"/>):
54             </xsl:when>
55
56             <xsl:otherwise>
57                Entire day:
58             </xsl:otherwise>
59
60          </xsl:choose>
61
62          <xsl:value-of select = "."/>
63
64          <xsl:if test = ". = ''">
65             n/a
66          </xsl:if>
67
```

Fig. D.13 Using conditional elements (part 2 of 3).

```
68            <br/>
69         </xsl:for-each>
70
71      </xsl:template>
72
73   </xsl:stylesheet>
```

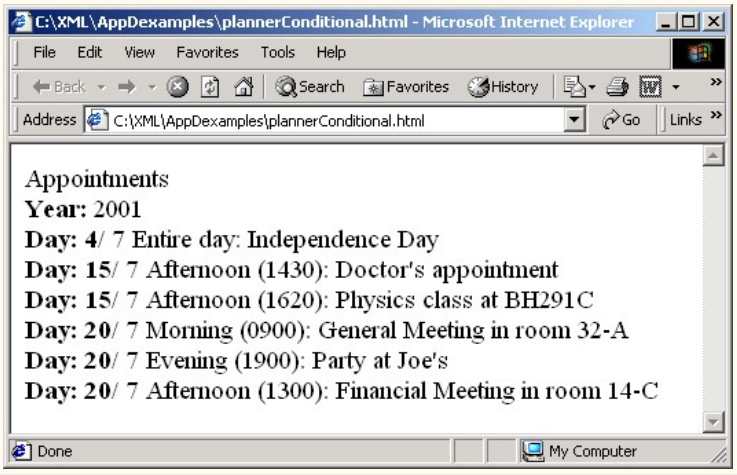


Fig. D.13 Using conditional elements (part 3 of 3).

D.7 Combining Style Sheets

XSLT allows for modularity in style sheets. This feature enables XSLT documents to import other XSLT documents. Figure D.14 lists an XSLT document that is imported into the XSLT document in Fig. D.15 using element ***import***.

```
1   <?xml version = "1.0"?>
2
3   <!-- Fig. D.14 : usage2.xsl     -->
4   <!-- xsl:import example         -->
5
6   <xsl:stylesheet version = "1.0"
7      xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9      <xsl:template match = "book">
10        <html>
11
12           <body>
13              <xsl:apply-templates/>
14           </body>
15        </html>
16
17     </xsl:template>
18
```

Fig. D.14 XSLT document being imported (part 1 of 2).

```
19      <xsl:template match = "title">
20         <xsl:value-of select = "."/>
21      </xsl:template>
22
23      <xsl:template match = "author">
24         <br/>
25
26         <p>Author:
27            <xsl:value-of select = "lastName"/>,
28            <xsl:value-of select = "firstName"/>
29         </p>
30
31      </xsl:template>
32
33      <xsl:template match = "*|text()"/>
34
35   </xsl:stylesheet>
```

Fig. D.14 XSLT document being imported (part 2 of 2).

The XSLT document in Fig. D.14 is similar to the ones presented earlier. One notable difference is line 33, which provides a **`template`** to match any element or text nodes. When these nodes are matched, this **`template`** indicates that no data will be written to the result tree. If the **`template`** is not provided, default **`template`**s will output the other nodes.

Line 9 in Fig. D.15 uses element ***`import`*** to use the **`template`**s defined in the XSLT document (Fig. D.14) 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 imported template, so it is used instead of the **`import`**ed template. Figure D.16 shows the result of transforming **`usage.xml`** (Fig. D.9).

```
1   <?xml version = "1.0"?>
2
3   <!-- Fig. D.15 : usage1.xsl              -->
4   <!-- xsl:import example using usage.xml -->
5
6   <xsl:stylesheet version = "1.0"
7      xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9      <xsl:import href = "usage2.xsl"/>
10
11     <!-- this template has higher precedence over
12          templates being imported  -->
13     <xsl:template match = "title">
14
15        <h2>
16           <xsl:value-of select = "."/>
17        </h2>
18
19     </xsl:template>
```

Fig. D.15 Importing another XSLT document (part 1 of 2).

```
20
21   </xsl:stylesheet>
```

Fig. D.15 Importing another XSLT document (part 2 of 2).

```
1   <html>
2      <body>
3         <h2>Deitel's XML Primer</h2>
4         <br>
5         <p>
6            Author: Deitel, Paul
7         </p>
8      </body>
9   </html>
```

Fig. D.16 Transformation results.

Figure D.17 shows an example of the XSLT element ***include***, which includes other XSLT documents in the current XSLT document. Lines 28–29 contain element **include**, which includes the files 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 templates are duplicated, the **template** that occurs last is used.

```
1   <?xml version = "1.0"?>
2
3   <!-- Fig. D.17 : book.xsl                  -->
4   <!-- xsl:include example using usage.xml -->
5
6   <xsl:stylesheet version = "1.0"
7      xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9      <xsl:template match = "/">
10
11     <html>
12        <body>
13           <xsl:apply-templates select = "book"/>
14        </body>
15     </html>
16
17     </xsl:template>
18
19     <xsl:template match = "book">
20
21        <h2>
22           <xsl:value-of select = "title"/>
23        </h2>
24
```

Fig. D.17 Combining style sheets using **xsl:include** (part 1 of 2).

```
25          <xsl:apply-templates/>
26       </xsl:template>
27
28       <xsl:include href = "author.xsl"/>
29       <xsl:include href = "chapters.xsl"/>
30
31       <xsl:template match = "*|text()"/>
32
33    </xsl:stylesheet>
```

Fig. D.17 Combining style sheets using **`xsl:include`** (part 2 of 2).

Figure D.18 and Figure D.19 list the XSLT documents being included by Figure D.17.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.18 : author.xsl              -->
4  <!-- xsl:include example using usage.xml -->
5
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>Author:
12          <xsl:value-of select = "lastName"/>,
13          <xsl:value-of select = "firstName"/>
14       </p>
15
16    </xsl:template>
17
18 </xsl:stylesheet>
```

Fig. D.18 XSLT document for rendering the author's name.

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.19 : chapters.xsl            -->
4  <!-- xsl:include example using usage.xml -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:template match = "chapters">
10       Chapters:
11
12       <ul>
13          <xsl:apply-templates select = "chapter"/>
14       </ul>
15    </xsl:template>
```

Fig. D.19 XSLT document for rendering chapter names (part 1 of 2).

```
16
17      <xsl:template match = "author">
18
19         <p><strong>Author is: </strong>
20            <xsl:value-of select = "lastName"/>,
21            <xsl:value-of select = "firstName"/>
22         </p>
23
24      </xsl:template>
25
26      <xsl:template match = "chapter">
27
28         <li>
29            <xsl:value-of select = "."/>
30         </li>
31
32      </xsl:template>
33
34   </xsl:stylesheet>
```

Fig. D.19 XSLT document for rendering chapter names (part 2 of 2).

Figure D.20 shows the result of the XSLT document (Fig. D.17) applied to the XML document describing a book (Fig. D.9).

```
1   <html>
2      <body>
3         <h2>Deitel's XML Primer</h2>
4         <strong>Author is: </strong>Deitel,
5            Paul</p>
6
7         Chapters:
8         <ul>
9            <li>XML Elements?</li>
10           <li>Easy XML</li>
11        </ul>
12     </body>
13  </html>
```

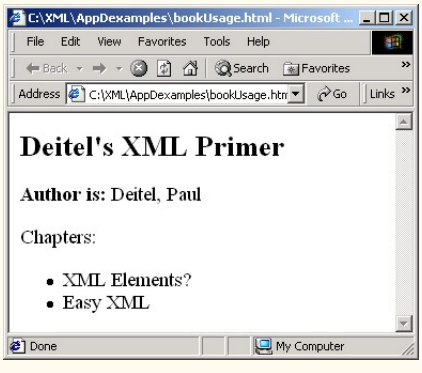


Fig. D.20 Output of an XSLT document using element **`include`**.

D.8 Variables

XSLT provides variables for storing information. These variables are not like Java variables, but rather more like Java constants. Figure D.21 provides an example of an XSLT document using element ***variable***.

Lines 13–14 use element **variable** to create a variable **named** **pageCount**, which stores the total number of pages in the book. Attribute **select** has the value **"sum(book/chapters/*/@pages)"**, which is an XPath expression for summing the number of **pages**. This XPath expression uses function **sum** to iterate over a set of nodes and sum their values. The set of nodes includes any element (*****) containing an attribute **pages** that is a child of **book/chapters**.

Line 15 uses element **value-of** to output the variable **pageCount**'s value. The *dollar sign* (**$**) references the variable's content.

Figure D.22 shows the output of the XSLT document (Fig. D.21) when it is applied to **usage.xml** (Fig. D.9). The total number of pages is 17.

D.9 Internet and World Wide Web Resources

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

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. D.21 : variables.xsl  -->
4  <!-- using xsl:variables        -->
5
6  <xsl:stylesheet version = "1.0"
7     xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
8
9     <xsl:template match = "/">
10
11       <total>
12          Number of pages =
13          <xsl:variable name = "pageCount"
14             select = "sum(book/chapters/*/@pages)"/>
15          <xsl:value-of select = "$pageCount"/>
16       </total>
17
18    </xsl:template>
19
20 </xsl:stylesheet>
```

Fig. D.21 Demonstrating **xsl:variable**.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <total>
3        Number of pages =
4        17</total>
```

Fig. D.22 Result of **variables.xsl** transformation.

www.w3.org/TR/xsl
The most current 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 XSL FAQs.

www.bayes.co.uk/xml/index.xml
A portal site that heavily uses XML and 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.

SUMMARY

- XSL Transformation Language (XSLT) transforms XML documents into other text-based documents using XSL format instructions. XSLT uses XPath to match nodes when transforming an XML document into a different document. The resulting document may be XML, XHTML, plain text or any other text-based document.
- To process XSLT documents, you will need an XSLT processor such as the **Transformer**-related classes in JAXP.
- An XSLT document is an XML document with a root element **xsl:stylesheet**. Attribute **version** indicates the XSLT specification used.
- An XSLT document's namespace URI is **http://www.w3.org/1999/XSL/Transform**.
- A **template** element matches specific XML document nodes by using an XPath expression in attribute **match**.
- Element **apply-templates** applies an XSLT document's templates to specific element nodes. By default, element **apply-templates** matches all element child nodes.
- The XSLT specification defines default templates for an XML document's nodes. The template

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

matches the document root node and any element nodes of an XML document and applies templates to their child nodes. The template

```
<xsl:template match = "text() | @*">
   <xsl:value-of select = "."/>
</xsl:template>
```

matches text nodes and attribute nodes and outputs their values. The template

```
<xsl:template match ="processing-instruction() | comment()"/>
```

matches processing-instruction nodes and comment nodes, but does not perform any actions with them.

- Element **element** creates an element with the name specified in attribute **name**.
- Element **attribute** creates an attribute for an element and can be contained only within an **element** element. Attribute **name** provides the name of the attribute.
- XSLT provides the capability to iterate through a node set returned by an XPath expression. XSLT also provides the capability to sort a node set.
- XSLT element **for-each** applies the element's contents to each of the nodes specified by attribute **select**.
- Element **sort** sorts (in the order specified by attribute **order**) the nodes specified in attribute **select**. Attribute **order** has values **ascending** (i.e., A–Z) and **descending** (i.e., Z–A).
- XSLT provides elements to perform conditional processing.
- Element **choose** allows alternative conditional statements to be processed.
- XSLT allows for modularity in style sheets. This feature allows XSLT documents to import other XSLT documents by using element **import**. Other XSLT document are referenced using attribute **href**.
- Local templates have higher precedence than imported templates. XSLT element **include** includes other XSLT documents in the current XSLT document.
- The difference between element **include** and element **import** is that templates included using element **include** have the same precedence as the local templates. Therefore, if any templates are duplicated, the template that occurs last is used.
- XSLT provides variables for storing values.

TERMINOLOGY

$
ascending
attribute **href**
attribute **match**
attribute **name**
attribute **order**
attribute **select**
attribute **test**
attribute **type**
attribute **version**
choose element
conditional processing
descending
DOMSource
element **apply-templates**
element **attribute**
element **element**
element **for-each**
element **if**
element **import**
element **include**
element **otherwise**
element **sort**
element **value-of**
element **variable**
element **when**
Extensible Stylesheet Language (XSL)
http://www.w3.org/1999/XSL/Transform URI
if conditional statement
otherwise condtional
root element **stylesheet**
StreamSource
template element
text-based document
when conditional
XPath
XPath expression
XSLT (XSL Transformation Language)
XSLT processor

SELF-REVIEW EXERCISES

D.1 State whether the following are *true* or *false*. If *false*, explain why.

a) XSLT uses XHTML to match nodes for transforming an XML document into a different document.
b) In its most current specification, XSLT does not allow for iteration through a node set returned by an XPath expression.
c) By using XSLT, XML documents can easily be converted between formats.
d) Like element **`choose`**, element **`if`** provides a single conditional test.
e) XSLT allows for modularity in stylesheets, which enables XSLT documents to import other XSLT documents.
f) The document resulting from an XSLT transformation may be in the format of an XML document, XHTML/plain text or any other text-based document.
g) XSLT does not provide default templates; all templates must be custom built by the programmer.
h) XSLT provides elements to perform conditional processing, such as **`if`** elements.
i) XSLT does not provide variables.

D.2 Fill in the blanks in each of the following statements.
a) Attribute _____ defines the XSLT specification used in an XSLT document.
b) An XSLT document is an XML document with a root element _____.
c) XSLT provides the _____ element to allow alternative conditional statements.
d) The letter T in XSLT stands for _____.
e) Templates of an XSLT document can be applied to specific nodes of an element by using element _____.
f) The **`template`** element matches specific _____ of an XML document.
g) Attribute _____ has values **`ascending`** and **`descending`**.
h) Element _____ includes other XSLT documents in the current XSLT document.

ANSWERS TO SELF-REVIEW EXERCISES

D.1 a) False. XSLT uses XPath to match nodes when transforming an XML document into a different document. b) False. XSLT allows for iteration through a node set returned by an XPath expression. c) True. d) False. Unlike element **`choose`**, element **`if`** provides a single conditional test. e) True. f) True. g) False. XSLT provides several default templates for the nodes of an XML document. h) True. i) False. XSLT provides variables.

D.2 a) **`version`**. b) **`stylesheet`**. c) **`choose`**. d) Transformation. e) **`apply-templates`**. f) nodes. g) **`order`**. h) **`include`**.