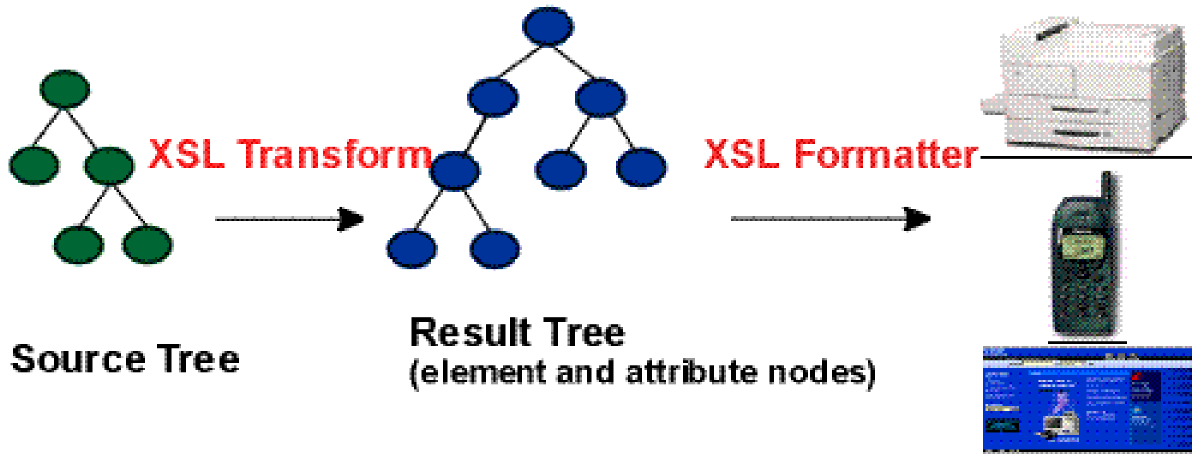
Introduction to XML

XSL (Extensible Stylesheet Language)

- Language for expressing style sheets
- Consists of two parts:
  - Transforming - XSLT
  - Formatting – XSL-FO



Result XML tree is the result of XSLT processing.

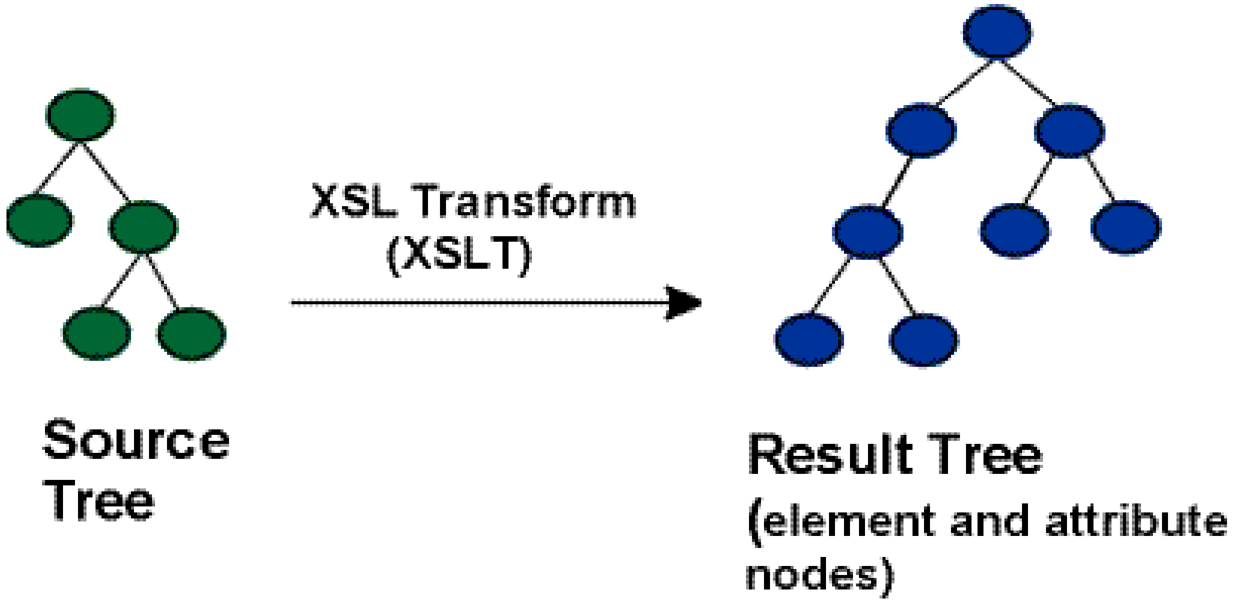
What is XSLT?

- Language for transforming XML documents into other ones
- Designed for use both as part of XSL and independently of XSL
    - E.g., use XSLT to create an output document that contains XSL Formatting Objects, and then feed this output into an XSL-FO processor that generates PDF, for example FOP from Apache (http://xmlgraphics.apache.org/fop/)
- Uses XPath for selecting nodes
    - Locate parts of the source tree document that match templates defined in the XSL stylesheet
- W3C Recommendation

How Tree Transformation Works?

- Style sheet describes rules for transforming a source tree into a result tree
- XSLT processor
  - Reads XML document & XSLT style sheet
  - Outputs a new XML document or fragment
- Transformation is achieved by associating patterns with templates
  - Pattern is matched against elements in source tree
  - Template is instantiated to create part of result tree

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree



In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

Data Model

- Data model of XSLT is the same as that used by XPath
- Tree structure of XML documents consists of 7 nodes:
  - The root node
  - Element nodes
  - Text nodes
  - Attribute nodes
  - Namespace nodes
  - Processing Instruction nodes
  - Comment nodes

Example: Periodic Table with two Atoms

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="eg.xsl"?>
<PERIODIC_TABLE>
      <ATOM STATE="GAS">
               <NAME>Hydrogen</NAME>
               <SYMBOL>H</SYMBOL>
               <ATOMIC_NUMBER>1</ATOMIC_NUMBER>
               <ATOMIC_WEIGHT>1.00794</ATOMIC_WEIGHT>
               <BOILING_POINT UNITS="Kelvin">20.28</BOILING_POINT>
               <MELTING_POINT UNITS="Kelvin">13.81</MELTING_POINT>
               <DENSITY UNITS="grams/cubic centimeter">
                              <!-- At 300K, 1 atm -->
                              0.0000899
               </DENSITY>
      </ATOM>
      <ATOM STATE="GAS">
               <NAME>Helium</NAME>
               <SYMBOL>He</SYMBOL>
               <ATOMIC_NUMBER>2</ATOMIC_NUMBER>
               <ATOMIC_WEIGHT>4.0026</ATOMIC_WEIGHT>
               <BOILING_POINT UNITS="Kelvin">4.216</BOILING_POINT>
               <MELTING_POINT UNITS="Kelvin">0.95</MELTING_POINT>
               <DENSITY UNITS="grams/cubic centimeter">
                              <!-- At 300K -->
                              0.0001785
               </DENSITY>
       </ATOM>
</PERIODIC_TABLE>
```

XSL Templates

- **Template rules** identify the nodes by using a **pattern**
  - Specified with the xsl:template element
- Content of xsl:template is instantiated
- match attribute is a pattern that identifies the source node or nodes to which the rule applies

  <xsl:template match = pattern >

  - The syntax for **patterns** is a subset of XPath expressions

The xsl:apply-templates element

- Tells a processor to compare each *child* element of the matched source element against the templates in the style sheet
  - If a match is found, output the template for the matched node
  - The template for the matched node may itself contain xsl:apply-templates elements to search for matches for its children

An XSLT style sheet (eg.xsl) for the periodic table

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl ="http://www.w3.org/1999/XSL/Transform">
          <xsl:template match="/">
                    <html>
                         <xsl:apply-templates/>
                    </html>
          </xsl:template>
          <xsl:template match="PERIODIC_TABLE">
                    <body>
                         <xsl:apply-templates/>
                    </body>
          </xsl:template>
          <xsl:template match="ATOM">
                    An Atom
          </xsl:template>
</xsl:stylesheet>
```

The result is:

```
<html>
    <body>
         An Atom
         An Atom
     </body>
</html>
```

Computing the value of a node with xsl:value-of

```
<xsl:template match="ATOM">
        <xsl:value-of select="NAME"/>
</xsl:template>
```

<u>The result is:</u>

```
<html>
   <body>
      Hydrogen
      Helium
   </body>
</html>
```

Values of Nodes in xsl:value-of

Node Type:	Value:
Root	The value of the root element 💬
Element	The concatenation of all parsed character data contained in the element, including character data in any of the descendants of the element 💬
Text	The text of the node
Attribute	The attribute value after entities are resolved and leading and trailing white space is stripped; does not include the name of the attribute, the equals sign, or the quotation marks
Namespace	The URI of the namespace
Processing instruction	The data in the processing instruction; does not include the processing instruction , <? or ?>
Comment	The text of the comment, <!-- and --> not included

Processing Multiple Elements with xsl:for-each

```
<xsl:template match="PERIODIC_TABLE">
    <xsl:for-each select="ATOM">
        <xsl:value-of select="."/>
    </xsl:for-each>
</xsl:template>
```

- The select="." in this template tells the processor to take the value of the *current element*, ATOM in this example

Patterns for Matching Nodes

- The match attribute of the xsl:template element supports a complex syntax that allows you to express exactly which nodes you do and do not want to match

- Matching the root node

  <xsl:template match="/">

Patterns for Matching Nodes

- Matching element names

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

- Matching children with /

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

<xsl:template match="PERIODIC_TABLE/*/SYMBOL">
```

- Matching descendants with //

```
<xsl:template match="PERIODIC_TABLE//NAME">
<xsl:template match="//ATOMIC_NUMBER">
```

- Matching by ID
  - Used to apply a particular style to a particular single element without changing all other elements of that type
  - id() selector: contains the ID value in single quotes

```
<xsl:template match="id('e47')">
    <b><xsl:value-of select="."/></b>
</xsl:template>
```

- Matching attributes with @

```
<xsl:template match="MELTING_POINT">
        <xsl:apply-templates select="@UNITS"/>
</xsl:template>
<xsl:template match="@UNITS">
                <I><xsl:value-of select="."/></I>
</xsl:template>
```

```
The output is:

<I>kelvin</I>
<I>kelvin</I>
```

- Using the or operator |
  - Allows a template rule to match multiple patterns

<xsl:template match="ATOMIC_NUMBER|ATOMIC_WEIGHT">

- Testing with [ ]
  - Test for more details about the nodes that match a pattern using []
  - Perform many different tests including:
    - Whether an element contains a given child, attribute, or other node
    - Whether the value of an attribute is a certain string
    - Whether the value of an element matches a string
    - What position a given node occupies in the hierarchy

<xsl:template match="ATOM[MELTING_POINT]">

<xsl:template match="ATOM[DENSITY/@UNITS]">

<xsl:template match="ATOM[ATOMIC_NUMBER='10']">

The Built-in Template Rules

- Default rule for **elements**:

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

- Applies to element nodes and the root node:
- *|/ is XPath shorthand for "any element or root node"
- Ensure that all elements are recursively processed

- Default rule for **text nodes** and **attributes**:

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

  - Matches all text and attribute nodes and outputs the value of the node (<xsl:value-of select="."/>).
  - Ensures that at the very least an element's text is output, even if no rule specifically matches it
  - Copies attribute values (but not names)

- Default rule for **processing instructions** and **comments**:
    - Simply says to do nothing
    - *Drop* them from the output

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

Attribute Value Templates

- Copy data from input document to attribute values
- Attribute values in curly braces {} takes the place of the xsl:value-of element
  - Ex: To convert the periodic table into empty ATOM elements, the template can be written as:

```
<xsl:template match="ATOM">
    <ATOM NAME="{NAME}“
              ATOMIC_WEIGHT="{ATOMIC_WEIGHT}"
              ATOMIC_NUMBER="{ATOMIC_NUMBER}" />
</xsl:template>
```

```
<ATOM NAME="Vanadium"
        ATOMIC_WEIGHT="50.9415"
        ATOMIC_NUMBER="23" />
```

```
<xsl:template match="ATOM">
        <A HREF="{SYMBOL}.html">
           <xsl:value-of select="NAME"/>
        </A>
</xsl:template>
```

The output is of the form:

```
<A HREF=H.html>Hydrogen</A>
<A HREF=He.html>Helium</A>
```

Inserting Elements with xsl:element

- Element name is given by an attribute value template in the name attribute of xsl:element
  - Replace ATOM with GAS, LIQUID, and SOLID

```
<xsl:template match="ATOM">
        <xsl:element name="{@STATE}">
                <NAME><xsl:value-of select="NAME"/></NAME>
                   <!-- rules for other children -->
        </xsl:element>
</xsl:template>
```

Inserting Attributes with xsl:attribute

- Name: name attribute of the xsl:attribute element
- Value: contents of the xsl:attribute element
- Each xsl:attribute element is child of either an xsl:element element or a literal element

```
<xsl:template match="ATOM">
       <LI><A>
           <xsl:attribute name="HREF">
                    <xsl:value-of select="SYMBOL"/>.html
           </xsl:attribute>
          <xsl:value-of select="NAME"/>
      </A></LI>
</xsl:template>
```

The output is of the form:
<LI><A **HREF="H.html**">Hydrogen</A></LI>
<LI><A **HREF="He.html**">Helium</A></LI>

Generating PI with xsl:processing-instruction

- Target: required name attribute
- Content: contents of the xsl:processing-instruction

  - Replaces PROGRAM elements with a gcc processing instruction:

```
<xsl:template match="PROGRAM">
      <xsl:processing-instruction name="gcc"> -O4
      </xsl:processing-instruction>
</xsl:template>
```

Output:

```
    <?gcc -O4 ?>
```

Generating Comments with xsl:comment

- xsl:comment element
  - no attributes
  - contents are the text of the comment

```
<xsl:template match="ATOM">
    <xsl:comment>There was an atom here  once. </xsl:comment>
</xsl:template>
```

Output: Replaces ATOM nodes with this comment:

```
    <!--There was an atom here once.-->
```

Generating Text with xsl:text

- Inserts its contents into the output document

```
<xsl:template match="ATOM">
    <xsl:text>There was an atom here once.</xsl:text>
</xsl:template>
```

Output: replaces each ATOM element with the string:

There was an atom here once.

Copying the Context Node with xsl:copy

- Copies source node into output tree
- Child elements, attributes, and other content are not automatically copied
  - Ex: strips comments from a document

```
<?xml version="1.0"?>
<xsl:stylesheet version=“1.0”
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="*|@*|processing-instruction()|text()">
       <xsl:copy>
         <xsl:apply-templates select="*|@*|processing-instruction()|text()"/>
       </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

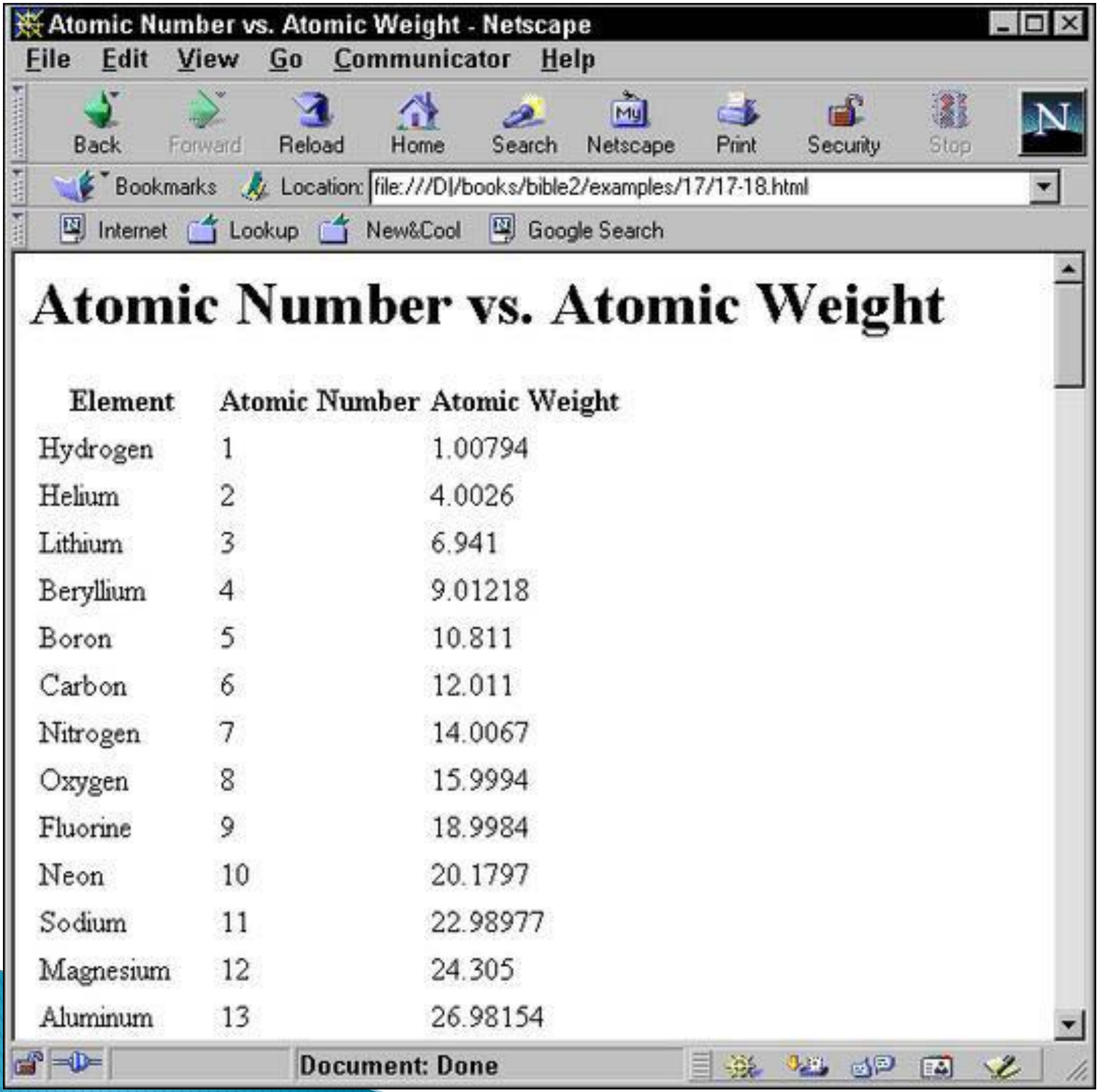
Sorting Output Elements

- xsl:sort element
  - Sorts output elements into a different order than they appear in the input
  - Appears as a child of xsl:apply-templates or xsl:for-each
  - The select attribute of xsl:sort defines the **key** used to sort the element's output

Ex: An XSLT style sheet that sorts by atomic number

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:template match="PERIODIC_TABLE">
          <html>
                <head>
                  <title>Atomic Number vs. Atomic Weight</title>
                </head>
                <body>
                    <h1>Atomic Number vs. Atomic Weight</h1>
                    <table>
                       <th>Element</th>
                       <th>Atomic Number</th>
                       <th>Atomic Weight</th>
                       <xsl:apply-templates>
                                  <xsl:sort data-type="number"
                                                       select="ATOMIC_NUMBER" order="ascending"/>
                       </xsl:apply-templates>
                    </table>
                </body>
          </html>
     </xsl:template>
     <xsl:template match="ATOM">
                <tr>
                   <td><xsl:apply-templates select="NAME"/></td>
                   <td><xsl:apply-templates select="ATOMIC_NUMBER"/></td>
                   <td><xsl:apply-templates select="ATOMIC_WEIGHT"/></td>
                </tr>
     </xsl:template>
</xsl:stylesheet>
```

Output: Atoms numerically sorted by atomic number


Atomic Number vs. Atomic Weight - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: file:///D|/books/bible2/examples/17/17-18.html
Internet Lookup New&Cool Google Search
Document: Done

Atomic Number vs. Atomic Weight

Element	Atomic Number	Atomic Weight
Hydrogen	1	1.00794
Helium	2	4.0026
Lithium	3	6.941
Beryllium	4	9.01218
Boron	5	10.811
Carbon	6	12.011
Nitrogen	7	14.0067
Oxygen	8	15.9994
Fluorine	9	18.9984
Neon	10	20.1797
Sodium	11	22.98977
Magnesium	12	24.305
Aluminum	13	26.98154

Modes

- Modes allow an element to be processed multiple times, each time producing a different result

- Both xsl:template and xsl:apply-templates have an optional mode attribute

Ex: An XSLT style sheet that uses modes to format the same data differently in two different p

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:template match="/PERIODIC_TABLE">
            <HTML>
                  <HEAD>
                       <TITLE>The Elements</TITLE>
                  </HEAD>
                  <BODY>
                        <H2>Table of Contents</H2>
                         <UL>
                             <xsl:apply-templates select="ATOM" mode="toc"/>
                        </UL>
                         <H2>The Elements</H2>
                                    </BODY> <xsl:apply-templates select="ATOM" mode="full"/>
            </HTML>
      </xsl:template>
      <xsl:template match="ATOM" mode="toc">
                  <LI><A>
                       <xsl:attribute name="HREF">#<xsl:value-of select="SYMBOL"/></xsl:attribute>
                        <xsl:value-of select="NAME"/>
                  </A></LI>
      </xsl:template>
      <xsl:template match="ATOM" mode="full">
                  <H3><A>
                        <xsl:attribute name="NAME"> <xsl:value-of select="SYMBOL"/> </xsl:attribute>
                        <xsl:value-of select="NAME"/>
                  </A></H3>
                  <P> <xsl:value-of select="."/> </P>
     </xsl:template>
</xsl:stylesheet>
```

Defining Constants with xsl:variable

- xsl:variable element defines a named string for use elsewhere in the style sheet *via an attribute value template*
  - Single attribute, name, which provides the variable name
  - Contents provide the replacement text

```
<xsl:variable name="copy01">
      Copyright 2001 Elliotte Rusty Harold
</xsl:variable>
```

Accessing the variable “copy01”

```
   <BLOCK COPYRIGHT="{$copy01}"> </BLOCK>
```

Named Templates

- xsl:template element has a name attribute by which it can be explicitly invoked
- xsl:call-template element has a required name argument that names the called template
- When processed, xsl:call-template is replaced by the contents of the xsl:template element it names

```
<xsl:template name="ATOM_CELL">
    <td>
        <font face="Times, serif" color="blue" size="2">
            <b>
                <xsl:value-of select="."/>
            </b>
          </font>
      </td>
</xsl:template>
```

Ex: Calling the named template **ATOM_CELL**

```
<xsl:template match="ATOMIC_NUMBER">
            <xsl:call-template name="ATOM_CELL"/>
</xsl:template>
```

Passing Parameters to Templates

- xsl:with-param element
  - The value of the parameter is specified in the same way as for xsl:variable and xsl:param

```
<xsl:template name="ATOM_CELL">
        <xsl:param name="file">index.html</xsl:param>
      <td>
          <font face="Times, serif" color="blue" size="2">
              <b><a href="{$file}"><xsl:value-of select="."/></a> </b>
          </font>
      </td>
</xsl:template>
```

Ex: Passing the parameter “file”

```
<xsl:template match="ATOMIC_NUMBER">
      <xsl:call-template name="ATOM_CELL">
         <xsl:with-param name="file">atomic_number.html</xsl:with-param>
      </xsl:call-template>
</xsl:template>
```

The Output is of the form:

```
<td>
    <font face="Times, serif" color="blue" size="2">
           <b>
              <a href="atomic_number.html">52</a>
           </b>
     </font>
</td>
```

Making Choices

- xsl:if
  - Provides a simple facility for changing output based on a pattern
  - test attribute of xsl:if contains an expression that evaluates to a boolean

Ex: A comma and a space is added after all names except the last name

```
<xsl:template match="ATOM">
      <xsl:value-of select="NAME"/>
      <xsl:if test="position()!=last()">, </xsl:if>
</xsl:template>
```

xsl:choose

- Selects one of several possible outputs
  - xsl:when: each condition and its associated output template
  - If none of the xsl:when elements are true, the xsl:otherwise child element is instantiated

Ex: Changes the color of the output based on whether the STATE attribute of the ATOM element is SOLID or LIQUID

```
<xsl:template match="ATOM">
    <xsl:choose>
      <xsl:when test="@STATE='SOLID'">
              <P style="color: black"> <xsl:value-of select="."/> </P>
      </xsl:when>
      <xsl:when test="@STATE='LIQUID'">
             <P style="color: blue"> <xsl:value-of select="."/> </P>
      </xsl:when>
      <xsl:otherwise>
              <P style="color: green"> <xsl:value-of select="."/> </P>
      </xsl:otherwise>
     </xsl:choose>
</xsl:template>
```

Output Methods

- xsl:output element
  - Allows stylesheet authors to specify how they wish the result tree to be output
  - method attribute specifies which output method to use and normally has one of the three values (XML, HTML, Text)

```
<!-- Category: top-level-element -->
<xsl:output  method = "xml" | "html" | "text" />
```

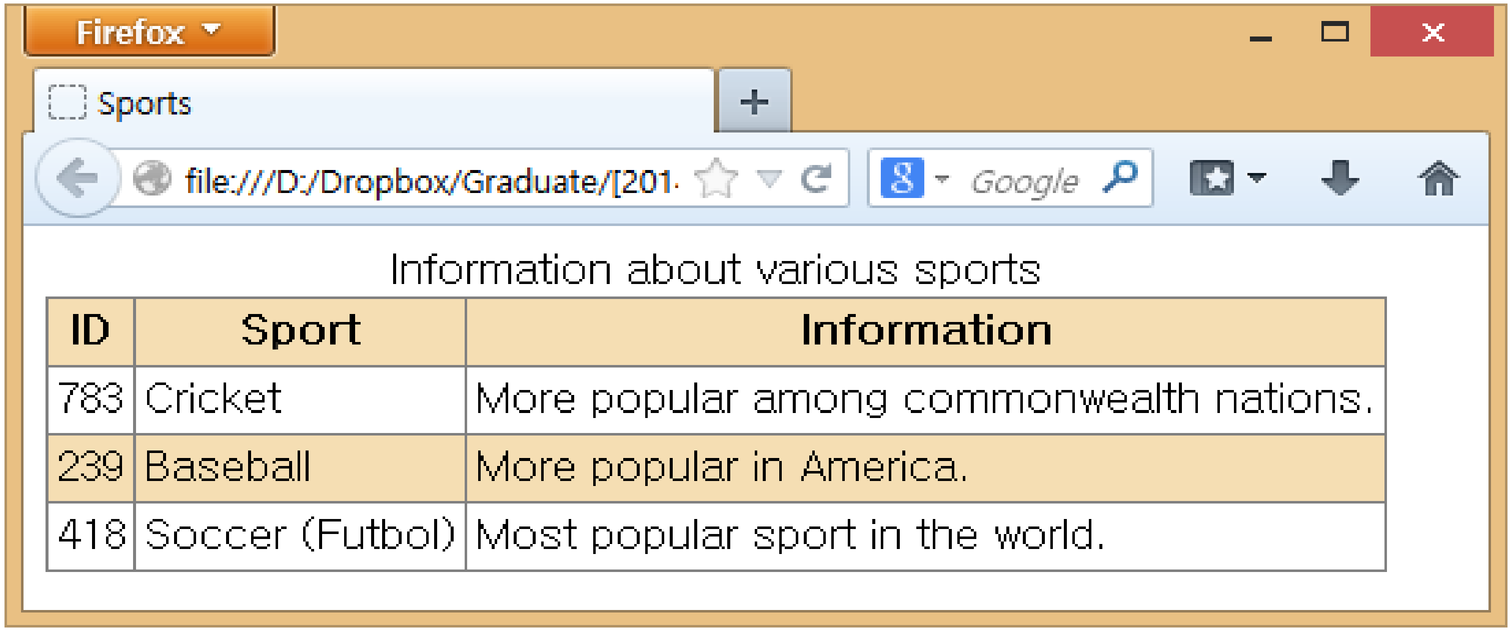
- XSLT does not analyze every node of the source tree
  - Selectively navigates the source tree using XPath's select and match attributes
- For XSLT to function, the source tree must be properly structured
- XSL style sheets can be connected directly to an XML document by adding an xml-stylesheet processing instruction to the XML document

- Two tree structures are involved in transforming an XML document using XSLT
  - source tree (the document being transformed)
  - result tree (the result of the transformation)
- XPath character ***/*** (a forward slash)
  - Selects the document root
  - In XPath, a leading forward slash specifies that we are using **absolute** addressing
  - An XPath expression with no beginning forward slash uses **relative** addressing

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

<!-- Fig. 15.18: sports.xml -->
<!-- Sports Database -->

<sports>
   <game id = "783">
      <name>Cricket</name>

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

   <game id = "239">
      <name>Baseball</name>

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

```

Fig. 15.18 | XML document that describes various sports. (Part 1 of 2.)

```
      <game id = "418">
         <name>Soccer (Futbol)</name>

         <paragraph>
            Most popular sport in the world.
         </paragraph>
      </game>
   </sports>
```



Information about various sports

ID	Sport	Information
783	Cricket	More popular among commonwealth nations.
239	Baseball	More popular in America.
418	Soccer (Futbol)	Most popular sport in the world.

Fig. 15.18 | XML document that describes various sports. (Part 2 of 2.)

```
<?xml version = "1.0"?>
<!-- Fig. 15.19: sports.xsl -->
<!-- A simple XSLT transformation -->

<!-- reference XSL style sheet URI -->
<xsl-stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

   <xsl:output method = "html" doctype-system = "about:legacy-compat" />
   <xsl:template match = "/"> <!-- match root element -->

   <html xmlns = "http://www.w3.org/1999/xhtml">
      <head>
         <meta charset = "utf-8"/>
         <link rel = "stylesheet" type = "text/css" href = "style.css"/>
         <title>Sports</title>
      </head>

      <body>
         <table>
         <caption>Information about various sports</caption>
            <thead>
               <tr>
```

Fig. 15.19 | XSLT that creates elements and attributes in an HTML5 document. (Part 1 of 2.)

```
                  <th>ID</th>
                  <th>Sport</th>
                  <th>Information</th>
               </tr>
            </thead>

            <!-- insert each name and paragraph element value -->
            <!-- into a table row. -->
            <xsl:for-each select = "/sports/game">
               <tr>
                  <td><xsl:value-of select = "@id"/></td>
                  <td><xsl:value-of select = "name"/></td>
                  <td><xsl:value-of select = "paragraph"/></td>
               </tr>
            </xsl:for-each>
         </table>
      </body>
   </html>

   </xsl:template>
</xsl:stylesheet>
```

Fig. 15.19 | XSLT that creates elements and attributes in an HTML5 document. (Part 2 of 2.)

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

<!-- Fig. 15.20: sorting.xml -->
<!-- XML document containing book information -->
<book isbn = "999-99999-9-X">
   <title>Deitel&apos;s XML Primer</title>

   <author>
      <firstName>Jane</firstName>
      <lastName>Blue</lastName>
   </author>

   <chapters>
      <frontMatter>
         <preface pages = "2" />
         <contents pages = "5" />
         <illustrations pages = "4" />
      </frontMatter>

```

Fig. 15.20 | XML document containing book information. (Part 1 of 2.)

```
         <chapter number = "3" pages = "44">Advanced XML</chapter>
         <chapter number = "2" pages = "35">Intermediate XML</chapter>
         <appendix number = "B" pages = "26">Parsers and Tools</appendix>
         <appendix number = "A" pages = "7">Entities</appendix>
         <chapter number = "1" pages = "28">XML Fundamentals</chapter>
      </chapters>

      <media type = "CD" />
   </book>
```

Fig. 15.20 | XML document containing book information. (Part 2 of 2.)

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

<!-- Fig. 15.21: sorting.xsl -->
<!-- Transformation of book information into HTML5 -->
<xsl:stylesheet version = "1.0"
   xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

   <!-- write XML declaration and DOCTYPE DTD information -->
   <xsl:output method = "html" doctype-system = "about:legacy-compat" />

   <!-- match document root -->
   <xsl:template match = "/">
      <html>
         <xsl:apply-templates/>
      </html>
   </xsl:template>

```

Fig. 15.21 | XSL document that transforms `sorting.xml` into HTML5. (Part 1 of 5.)

```
    <!-- match book -->
    <xsl:template match = "book">
       <head>
          <meta charset = "utf-8"/>
          <link rel = "stylesheet" type = "text/css" href = "style.css"/>
          <title>ISBN <xsl:value-of select = "@isbn"/> -
             <xsl:value-of select = "title"/></title>
       </head>

       <body>
          <h1><xsl:value-of select = "title"/></h1>
          <h2>by
             <xsl:value-of select = "author/lastName"/>,
             <xsl:value-of select = "author/firstName"/></h2>

          <table>

             <xsl:for-each select = "chapters/frontMatter/*">
                <tr>
                   <td>
                      <xsl:value-of select = "name()"/>
                   </td>

```

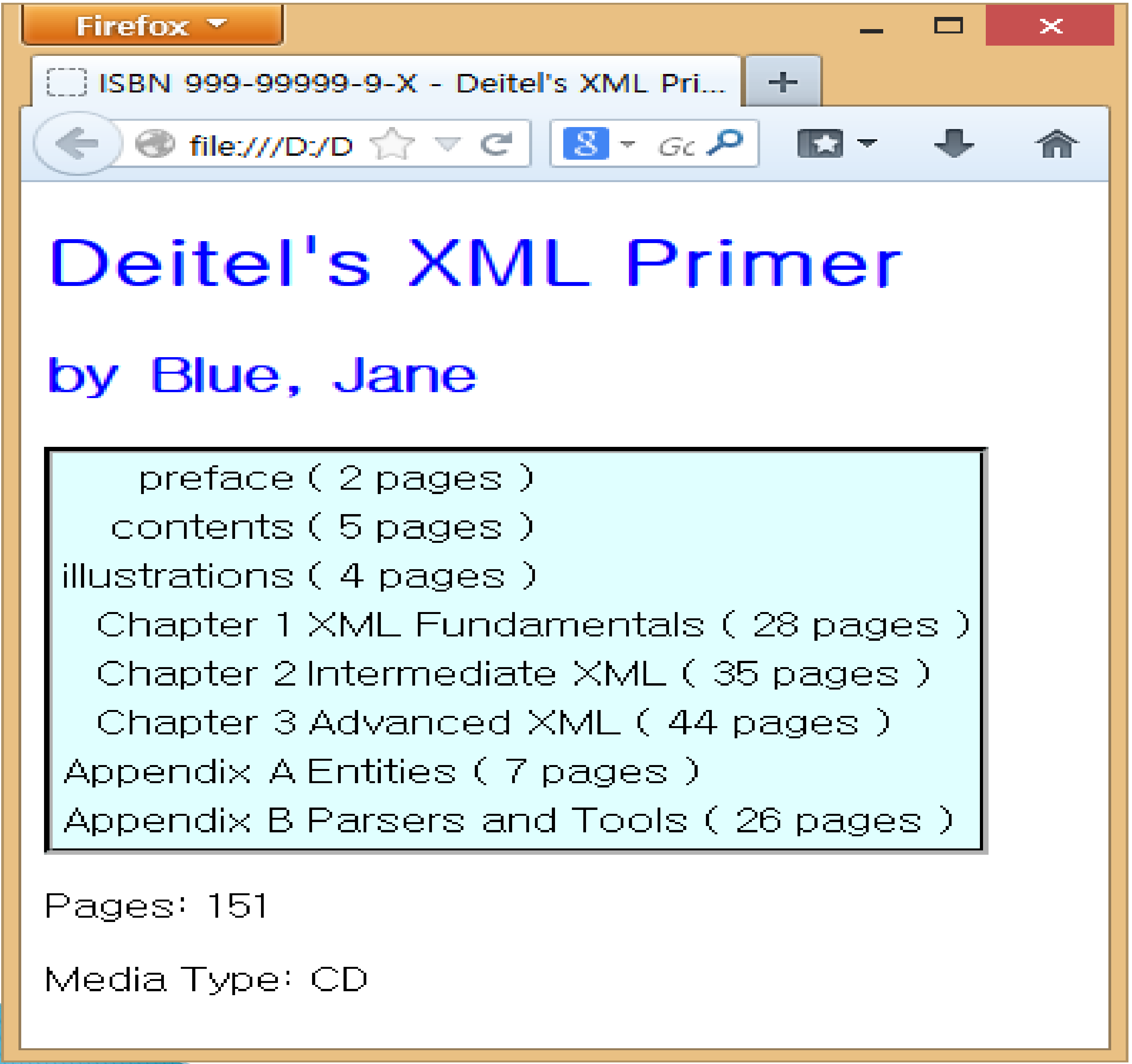
Fig. 15.21 | XSL document that transforms sorting.xml into HTML5. (Part 2 of 5.)

```
                  <td>
                     ( <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>

            <xsl:for-each select = "chapters/chapter">
               <xsl:sort select = "@number" data-type = "number"
                    order = "ascending"/>
               <tr>
                  <td>
                     Chapter <xsl:value-of select = "@number"/>
                  </td>

                  <td>
                     <xsl:value-of select = "text()"/>
                     ( <xsl:value-of select = "@pages"/> pages )
                  </td>
               </tr>
            </xsl:for-each>

```

Fig. 15.21 | XSL document that transforms `sorting.xml` into HTML5. (Part 3 of 5.)

```
                  <xsl:for-each select = "chapters/appendix">
                     <xsl:sort select = "@number" data-type = "text"
                        order = "ascending"/>
                     <tr>
                        <td>
                           Appendix <xsl:value-of select = "@number"/>
                        </td>

                        <td>
                           <xsl:value-of select = "text()"/>
                           ( <xsl:value-of select = "@pages"/> pages )
                        </td>
                     </tr>
                  </xsl:for-each>
               </table>

               <p>Pages:
                  <xsl:variable name = "pagecount"
                     select = "sum(chapters//*/@pages)"/>
                  <xsl:value-of select = "$pagecount"/>
               <p>Media Type: <xsl:value-of select = "media/@type"/></p>
            </body>
         </xsl:template>
      </xsl:stylesheet>
```

Fig. 15.21 | XSL document that transforms `sorting.xml` into HTML5. (Part 4 of 5.)



Deitel's XML Primer

by Blue, Jane

preface (2 pages)
contents (5 pages)
illustrations (4 pages)
Chapter 1 XML Fundamentals (28 pages)
Chapter 2 Intermediate XML (35 pages)
Chapter 3 Advanced XML (44 pages)
Appendix A Entities (7 pages)
Appendix B Parsers and Tools (26 pages)

Pages: 151

Media Type: CD