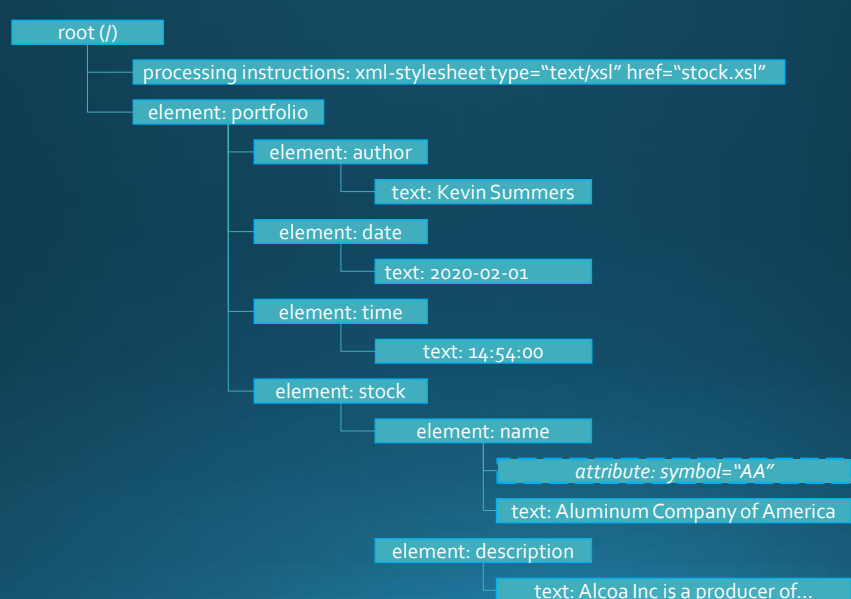
XSL Transformations

Reference Text: New Perspectives on XML, Comprehensive, 3rd Edition
Patrick Carey, Sasha Vodnik, 2015

1

Remember the Node Tree…



2

Remember XSLT Templates…

- A **template** is a collection of elements that define how a particular section of the source document should be transformed in the result document
- The **root template** sets up the initial code for the result document
- To create a template, the syntax is:

```
<xsl:template match="node">
   XSLT and Literal Result Elements
</xsl:template>
```

where **node** is either the name of a node from the source document's node tree, **or** an XPath expression that points to a node in the tree

3

Applying a Template to a Source Document

Use the following XSLT element
<xsl:apply-templates select="*XPath Expression*" />
where *XPath Expression* indicates the node template to be applied

```
<xsl:template match="/">
  <html>
    <head>
        <title>Stock Information</title>
        <link href="stock.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <div id="datetime"><b>Last Updated:</b>
            <xsl:value-of select="portfolio/date"/>  at
            <xsl:value-of select="portfolio/time"/>
        </div>
        <h1 class="title">Hardin Financial</h1>
        <h2 class="title">Stock Information</h2>
        <xsl:apply-templates select="portfolio/stock/name"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="name"/>
    <h3 class="name">
        <xsl:value-of select="."/>
    </h3>
</xsl:template>
```

The **<xsl:for-each>** block has been replaced with **<xsl:apply-templates>**

This is the template that will be applied

4

Sorting Nodes

By default, nodes are processed in document order, by their appearance in the document

To specify a different order, XSLT provides the ***<xsl:sort>*** element

This element can be used with either the ***<xsl:apply-templates>*** or the ***<xsl:for-each>*** element

The ***<xsl:sort>*** element contains several attributes to control how the XSLT process sorts the nodes in the source document

The ***select*** attribute determines the criteria under which the context node is sorted	The ***data-type*** attribute indicates the type of data (number, text)	The ***order*** attribute indicates the direction of the sorting (ascending or descending)

```
<xsl:apply-templates select="day">
    <xsl:sort data-type="number" order="descending"/>
</xsl:apply-templates>
```

5

Conditionally Creating Nodes

XSLT supports two kinds of conditional elements:

<xsl:if>	<xsl:choose>

To apply a format only if a particular condition is met , use the ***<xsl:if>*** element

To test for multiple conditions and display different outcomes, use the ***<xsl:choose>*** element

6

Conditionally Creating Nodes - Example

```
<xsl:template match="today">
    <table class="today">
        <tr>
            <th class="today">Current</th>
            <th class="today">Open</th>
            <th class="today">High</th>
            <th class="today">Low</th>
            <th class="today">Volume</th>
        </tr>
        <tr>
            <td class="number">
                <xsl:choose>
                    <xsl:when test="@current &lt; @open">
                        <img src="down.gif"/>
                    </xsl:when>
                    <xsl:when test="@current > @open">
                        <img src="up.gif"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <img src="same.gif"/>
                    </xsl:otherwise>
                </xsl:choose>
                <xsl:value-of select="@current"/>
            </td>
            <td class="number"><xsl:value-of select="@open"/></td>
            <td class="number"><xsl:value-of select="@high"/></td>
            <td class="number"><xsl:value-of select="@low"/></td>
            <td class="number"><xsl:value-of select="@volume"/></td>
        </tr>
    </table>
</xsl:template>
```

node (element) to process

operator (see next slide)

attributes of the element

All the html elements are ***literals***

7

OPERATOR	DESCRIPTION	EXAMPLE
=	Tests whether two values are equal to each other	@symbol = "AA"
!=	Tests whether two values are unequal	@symbol != "AA"
<	Tests whether one value is less than another	day < 5
<=	Tests whether one value is less than or equal to another	day <= 5
>	Tests whether one value is greater than another	day > 1
>=	Tests whether one value is greater than or equal to another	day >= 1

Comparison Operators & Functions

8

Predicates

- ***Predicates*** are XPath expressions that test for a condition and create subsets of nodes that fulfill that condition
- The predicate can also indicate the position of the node in the node tree
- To select a specific position in the source document, use the ***position()*** function combined with any XPath expression

9

Predicate Example in a Root Template

```
<xsl:template match="/">
  <html>
    <head>
        <title>Stock Information</title>
        <link href="stock.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <div id="datetime"><b>Last Updated:</b>
            <xsl:value-of select="portfolio/date"/>  at
            <xsl:value-of select="portfolio/time"/>
        </div>
        <h1 class="title">Hardin Financial</h1>
        <h2 class="title">Stock Information</h2>

        <h2 class="category">Industrials</h2>
        <xsl:apply-templates select="portfolio/stock/[category='Industrials']">
            <xsl:sort select="name"/>
        </xsl:apply-templates>

        <h2 class="category">Utilities</h2>
        <xsl:apply-templates select="portfolio/stock/[category='Utilties']">
            <xsl:sort select="name"/>
        </xsl:apply-templates>

        <h2 class="category">Transportation</h2>
        <xsl:apply-templates select="portfolio/stock/[category='Transportation']">
            <xsl:sort select="name"/>
        </xsl:apply-templates>

    </body>
  </html>
</xsl:template>
```

Test for condition

10

Creating Elements & Attributes

- To create an element, XSLT uses the **<xsl:element>** tag
- The **name** attribute assigns a name to the element
- The **namespace** attribute provides a namespace
- The **use-attribute-sets** provides a list of attribute-sets

11

Creating an Element

To create the *<a>* element in the result document, use the *<xsl:element>* tag

```
<xsl:template match="name"/>
    <xsl:element name="a">
        <h3 class="name">
            <xsl:value-of select="."/>
            ( <xsl:value-of select="@symbol")/> )
        </h3>
    </xsl:element>
</xsl:template>
```

```
...
<stock>
  <name symbol="ORCL">Oracle Corporation</name>
  ..
  ..
</stock>
...
```

```
<a><h3 class="name">Oracle Corporation (ORCL)</h3></a>
```

12

Creating an Attribute

Attributes are created in XSLT by using the **<xsl:attribute>** element

The **name attribute** specifies the name of the attribute

The **namespace** attribute indicates the namespace

13

Creating an Attribute

To add the **href** attribute to the **<a>** tag, use the <xsl:attribute> element

```
<xsl:template match="name"/>
    <xsl:element name="a">
        <xsl:attribute name="href">
            <xsl:value-of select="../link"/>
        </xsl:attribute>
        <h3 class="name">
            <xsl:value-of select="."/>
            ( <xsl:value-of select="@symbol")/> )
        </h3>
    </xsl:element>
</xsl:template>
```

```
...
<stock>
  <name symbol="ORCL">Oracle Corporation</name>
  <link>http://theWebAddress</link>
  ..
</stock>
...
```

```
<a href="http://theWebAddress">
  <h3 class="name">Oracle Corporation (ORCL)</h3>
</a>
```

14

Creating Comments & Processing Instructions

Comment	The *<xsl:comment>* element creates the comment
Processing Instruction	You can create a processing instruction by using the *<xsl:processing-instruction>* element
Use	E.g. If you wanted to add a processing instruction to attach the result document to the *style.css* sheet, use the following code:

```
<xsl:processing-instruction name="xml-stylesheet">
    href="styles.css" type="text/css"
</xsl:processing-instruction>
```

15

Summary so far…

- Extensible Style sheet Language (or XSL), from an Enterprise integration perspective is composed of **XSLT** and **XPath**
- **XPath** language is used to reference a node
- **Templates** are used to format sections of the XML document and transform XML data into a variety of formats
- Nodes can be **sorted** in either alphabetical or numerical order
- **Comparison** elements allow changing the contents of the result document based on the values of the nodes in the source document
- **Predicates** are used to handle subsets of the source document's node tree
- You can **insert new elements and attributes** in the transformed document

16

Some More XSLT

17

The <xsl:number> Element

- Attributes:
  - **value**=expression: any XPath expression that evaluates to a number
  - **count**=pattern: specifies which nodes to count
  - **format**=pattern: pattern indicates number format
  - **grouping-size, grouping-separator**: indicate how digits are grouped and separator character
- Used to determine the integer **position** of the current node in the **source.**

18

XPath Functions

Used to *calculate numerical values* or *manipulate text strings*

The ***position()*** XPath function can be used along with the ***<xsl:number>*** element to output the numeric position of the current node in the result document

```
<xsl:for-each select="playlist/track">
        <xsl:number value="position()" format="1" />
        <xsl:value-of select="title" /><br />
</xsl:for-each>
```

19

FUNCTION	DESCRIPTION
ceiling(*number*)	Rounds *number* up to the nearest integer
count(*node_set*)	Counts the number of nodes in *node_set*
floor(*number*)	Rounds *number* down to the nearest integer
last(*node_set*)	Returns the index of the last node in *node_set*
position()	Returns the position of the context node within the processed node set
round(*number*)	Rounds *number* to the nearest integer
sum(*node_set*)	Calculates the sum of the values of *node_set*

XPath Numeric Functions

20

FUNCTION	DESCRIPTION
concat(*string1*, *string2*, *string3*, ...)	Combines *string1*, *string2*, *string3*, ... into a single text string
contains(*string1*, *string2*)	Returns the value true if *string1* contains the text string, *string2* and false if otherwise
starts-with(*string1*, *string2*)	Returns the value true if *string1* begins with the characters defined in *string2* and false if otherwise
string(*object*)	Converts *object* to a text string. If *object* is not specified, the string() function returns the string value of the context node.
string-length(*string*)	Returns the length of *string*. If *string* is not specified, the string-length() function returns the length of the string value of the context node.
substring(*string*, *start*, *length*)	Returns a substring from *string*, starting with the character in the *start* position and continuing for *length* characters. If no length is specified, the substring goes to the end of the original text string.
substring-after(*string1*, *string2*)	Returns a substring of *string1* that occurs after the characters defined in *string2*
substring-before(*string1*, *string2*)	Returns a substring of *string1* that occurs before the characters defined in *string2*

XPath Text Functions

21

OPERATOR	DESCRIPTION	EXAMPLE
+	Adds two numbers together	3 + 5
–	Subtracts one number from another	5 – 3
*	Multiplies two numbers together	5 * 3
div	Divides one number by another	15 divided by 3
mod	Provides the remainder after performing a division of one number by another	15 mod 3
–	Negates a single number	–2

XPath Mathematical Operators

22

Formatting Numbers
Number Format Symbols

XPath function *format-number*

Syntax: *format-number(value, format)*

Example: *format-number(56823.847, "#,##0.00")* displays *56,823.85*

SYMBOL	DESCRIPTION
#	Placeholder that displays an optional number of digits in the formatted number and is usually used as the leftmost symbol in the number format
0	Placeholder that displays required digits in the formatted number
.	Separates the integer digits from the fractional digits
,	Separates groups of digits in the number
;	Separates the pattern for positive numbers from the pattern for negative numbers
–	Shows the location of the minus symbol for negative numbers
%	Multiplies the number by 100 and displays the number as a percentage
‰	Multiplies the number by 1000 and displays the number as a per-mille value

23

The <xsl:decimal-format> Element

- Holds decimal formatting information
- Controls separator characters such as . and ,
- Can be named or default if un-named
- Named decimal format passed as argument to format-number

```
<xsl:decimal-format name="euro" decimal-separator="," grouping-separator="."/>
```

24

ATTRIBUTE	DESCRIPTION
name	Name of the decimal format. If you omit a name, the numbering scheme becomes the default format for the document.
decimal-separator	Character used to separate the integer and fractional parts of the number. The default is ".".
grouping-separator	Character used to separate groups of digits. The default is ",".
infinity	Text string used to represent infinite values. The default is "infinity".
minus-sign	Character used to represent negative values. The default is "-".
NaN	Text used to represent entries which are not numbers. The default is "NaN".
percent	Character used to represent numbers as percentages. The default is "%".
per-mille	Character used to represent numbers in parts per 1000. The default is "0/00".
zero-digit	Character used to indicate a required digit in the number format pattern. The default is "0".
digit	Character used to indicate an optional digit in the number format pattern. The default is "#".
pattern-separator	Character used to separate positive number patterns from negative number patterns in the number format. The default is ";".

<xsl:decimal-format> Attributes

25

Inserting Attribute Values

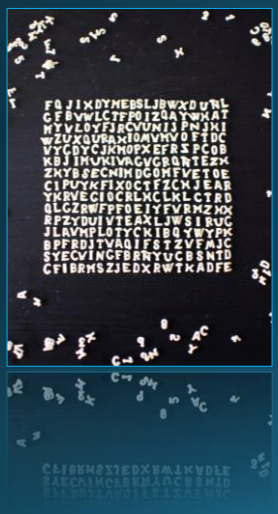
E.g. ***XSLT expression*** inserted into HTML attribute value

Syntax: *<tag attribute="{XSLT expression}">*

Example: *<td rowspan="{count(../../Items/Item)}">*

26

Text Nodes & White Space



- White space:
  - Space devoid of any printable character
    - Space, tab, new line, carriage return
- Adjacent <xsl:value-of> elements will have results combined to eliminate white space
- <xsl:text> can be used to create white space:
  - Syntax: *<xsl:text>Text</xsl:text>*
- Can only contain literal text

27

Working with White Space

- White Space Entities
  - Space -
  - Tab - 	
  - New line -

  - Carriage return -
- Stripping space:
  - Remove text nodes from the result document that contain only white space
  - Syntax: <xsl:strip-space elements="pattern">
  - Use * as pattern to match all nodes
- Preserving space:
  - Make sure that text nodes that contain only white space are not deleted
  - Syntax: <xsl:preserve-space elements="pattern">
  - Use * as pattern to match all nodes
- Normalize space:
  - Remove leading and trailing spaces
  - Syntax: normalize-space(text)

28

Variables

User-defined name that stores a particular value or object

Types of Variables:

XPath
- number
- text string
- boolean
- node set

Non XPath
- result tree fragment

29

Using Variables

- Syntax: *<xsl:variable name="name" select="value"/>*
  - Example: *<xsl:variable name="Months" select="12" />*

- Names are case-sensitive
- Value only set once upon declaration
- Enclose text strings in single-quotes
- *Value* can be an *XPath expression*
- Boolean type:
  - Set value to expression that is true or false
- Result tree fragment type:
  - Syntax:

```
<xsl:variable name="Logo">
    <img src="logo.gif" width="300" height="100" />
</xsl:variable>
```

30

To Reference a Variable

Syntax: *$variable-name*

Example: *$Months*

Referencing tree fragments:

Do not use ***$variable-name***

Use ***<xsl:copy>*** or ***<xsl:copy-of>*** to reference value – see next slide

31

Copying

<xsl:copy>

Syntax: *<xsl:copy />*

Shallow copy: only node itself is copied

<xsl:copy-of>

Syntax: *<xsl:copy-of select="expression"/>*

Deep copy: node and descendants are copied

32

Variable Scope

Global

Can be referenced from anywhere within the style sheet

Must be declared at the top level of the style sheet, as a direct child of the *<xsl:stylesheet>* element

Must have a unique variable name

Local

Referenced only within a template

Can share name with other local or global variable

33

Similar to variables, but:

Value can be changed after it is declared

Can be set outside of scope

Syntax:

<xsl:param name="name" select="value"/>

Example:

<xsl:param name="Filter" select="'C103'" />

To reference: *$param-name*

Parameters

34

Setting Parameter Values Externally

Depends on XSLT processor

Some work by appending parameter value to url

Command line processors allow external parameter setting:

MSXML
Saxon

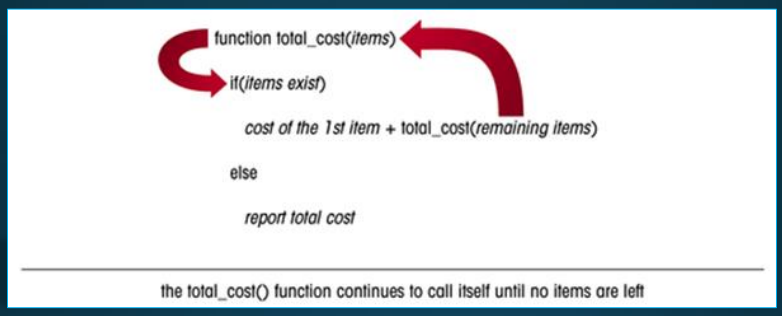
35

Template Parameters

- Local in scope
- Created inside **<xsl:template>** element
- To pass a parameter to template
  - place **<xsl:with-param>** element in **<xsl:apply-templates>** element
  - Syntax: ***<xsl:with-param name="name" select="value"/>***
  - No error if calling param name does not match template param name

```
<xsl:template name="print_grade">
        <xsl:param name="target_grade" />
        <xsl:param name="list" />
                ...
                <h1><xsl:value-of select="$target_grade" /></h1>
                ....
</xsl:template>
```

36



Using Recursion in XSL

Let's have a look at recursion in functional programming - it's a function that calls itself

39

Writing a Recursive Template

Templates that call themselves, usually passing along a new parameter value with each call

Needs to have a *stopping condition*

Expressed in an if statement or a choose statement

If missing, will call itself without end

Uses *<xsl:if>* syntax

```
<xsl:template name="template_name">
  <xsl:param name="param_name" select="default_value" />
  ...
  <xsl:if test="no_stopping_condition">
  ...
    <xsl:call-template name="template_name">
      <xsl:with-param name="param_name" select="new_value" />
    </xsl:call-template>
  ...
  </xsl:if>
</xsl:template>
```

40

Writing a Recursive Template

Uses ***<xsl:choose>*** syntax

```
<xsl:template name="template_name">
  <xsl:param name="param_name" select="default_value" />
  ...
  <xsl:choose>
      <xsl:when test="stopping_condition">
      ...
      </xsl:when>
      <xsl:otherwise>
        ...
       <xsl:call-template name="template_name">
         <xsl:with-param name="param_name" select="new_value" />
       </xsl:call-template>
       ...
      </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

41

Writing a Recursive Template

Example

Consider a list/set of ***item*** elements where the item has two attributes ***"qty"*** for the quantity of the item and ***"price"*** for the cost of the item.

Below is a template that will calculate the total cost for all the item elements…

Stopping condition

Recursive call

```
<xsl:template name="total_cost">
    <xsl:param name="list"/>
    <xsl:param name="total" select="0"/>
    <xsl:choose>
        <xsl:when test="$list">
            <xsl:variable name="first" select="$list[1]"/>
            <xsl:call-template name"total_cost">
                <xsl:with-param name="$list" select="$list[position() > 1]" />
                <xsl:with-param name="total" select="$first/@qty * $first/@price + $total" />
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="format-number($total, '€#,#00.00)" />
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

42

XML, XSD, XSL and *Java*

43

Executing XML validation and Transformation in *Java*

- So far we've used other applications to do the work
  - ✓ XML parsing/viewing using a browser (we did do some simple Java for this also however)
  - ✓ XML validation using Notepad++
  - ✓ XML transformation using Microsoft Edge

- Next we will consider these functions from a programming perspective
- In Java – we can load, parse, validate and transform XML
- To do this we require code libraries to help us achieve it
- Java API: Java API / Architecture for XML Processing (JAXP)
- Java comes with bundled classes to help with this *(Xerces for xml / Xalan for xslt)*

44

Java code to load an *XML File*

```
//Load the xml file...
File file = new File("shipment.xml");
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
```

Java code to load an *XML Schema File*

```
//Load the xml schema and create a validator for it...
SchemaFactory factory =
    SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
Schema schema = factory.newSchema(new File("shipment.xsd"));
```

Java code to *Validate an XML File*

```
//Validate the xml file against the schema...
Validator validator = schema.newValidator();
validator.validate(new DOMSource(doc));
```

45

```
try{

    //Load the xml file...
    File file = new File("shipment.xml");
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.parse(file);

    //Load the xml schema and create a validator for it...
    SchemaFactory factory =
        SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    Schema schema = factory.newSchema(new File("shipment.xsd"));

    //Validate the xml file against the schema...
    Validator validator = schema.newValidator();
    validator.validate(new DOMSource(doc));

}
catch (Exception e){
    System.out.println(e.getMessage());
}
```

Final Code to *load* and *validate* an *XML* file

46

```
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer;

transformer = tFactory.newTransformer( new StreamSource("shipment.xsl"));
transformer.transform(new StreamSource("shipment.xml"),
                      new StreamResult(new FileOutputStream("shipment.html")));
System.out.println("** The output is written in " + "shipment.html" + " **");
```

Java Code to transform an XML file using an XSLT Stylesheet (XSL file)

47