

XSLT

Sang Shin

JPassion.com

“Learn with JPassion!”



Agenda

- Terminology: XSL, XSLT, XSL-FO
- Why Transformation?
- XSLT Operational Model
- Review on XPath
- **XSLT Stylesheet Language** (the focus of this presentation)
- XSLT vs. other Technologies

Agenda (continued)

- XSLT stylesheet language
 - > template
 - > value-of
 - > apply-templates
 - > for-each
 - > if
 - > when, choose, otherwise
 - > sort
 - > filtering

Terminology

XSL

- eXtensible Stylesheet Language
- A language for expressing **stylesheets**
- Made of two parts
 - > XSL Transformation (XSLT)
 - > XSL Formatting Objects (XSL-FO)

Transformation

- Transforming XML document into
 - > Another XML document
 - > XHTML
 - > WML
 - > HTML document
 - > Text
- XSLT
 - > W3C standard for XML transformation

Why Transformation?

Two Viewpoints of XML

- Presentation Oriented Publishing (POP)
 - > Useful for Browsers and Editors
 - > Usually used for data that will be consumed by Humans
- Message Oriented Middleware (MOM)
 - > Useful for Machine-to-Machine data exchange
 - > Business-to-Business communication an excellent example

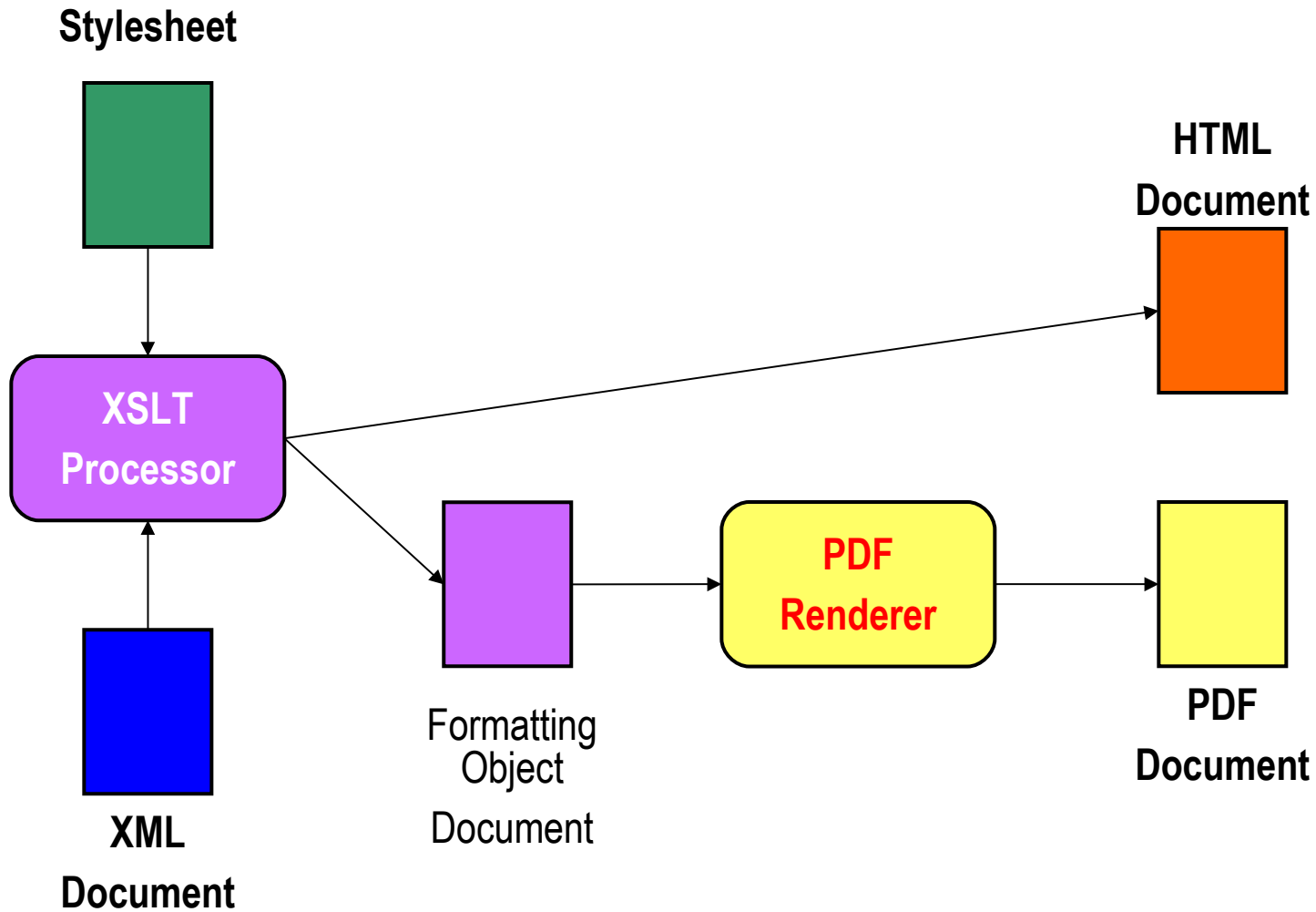
Importance of Transformation

- XSLT is **useful** in both POP and MOM
 - > transforming data into a viewable format in a browser (POP)
 - > transforming business data between content models (MOM)

XSLT in POP

- XML document typically represents only **content (data)** separated from **presentation** (unlike HTML)
- Transformations can be used to **style (render, present)** XML content into presentation
- A common styling technique presents XML in HTML format for desktop clients or WML format for mobile device client

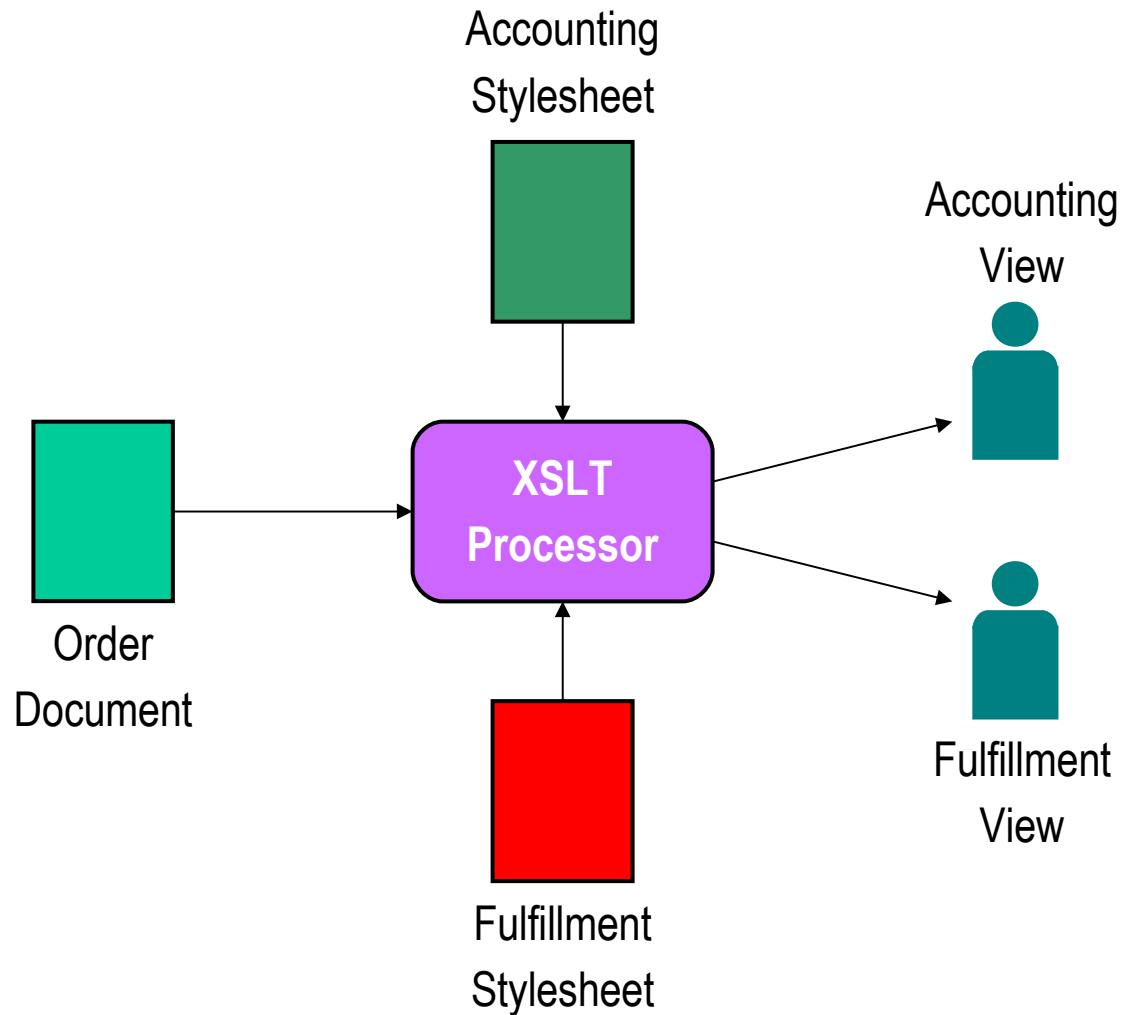
XSLT – in POP



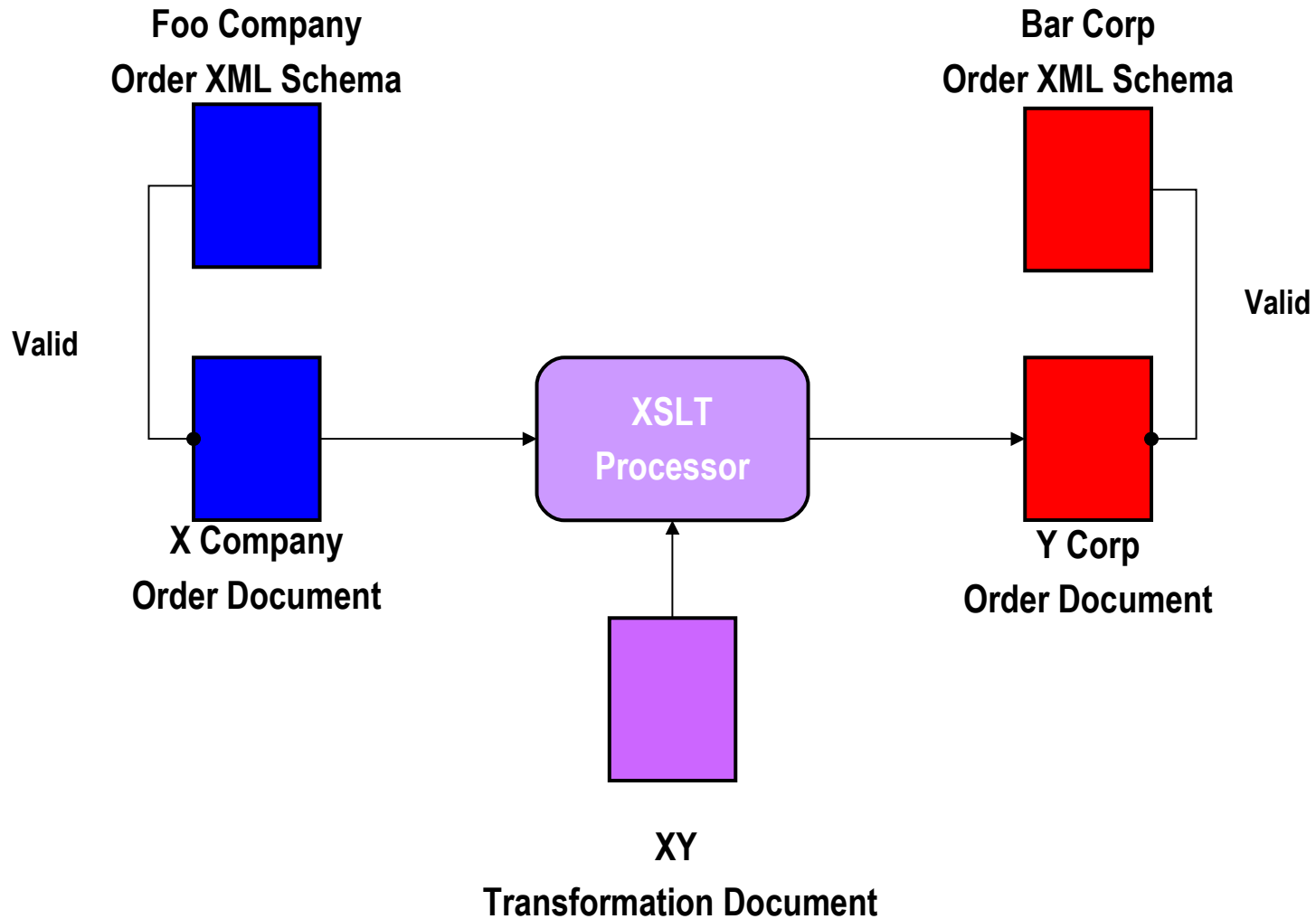
XSLT in MOM

- Important for eCommerce, B2B/EDI, and dynamic content generation
 - > Different content model
 - > Different structural relationship
 - > Different vocabularies

XSLT – in MOM Example #1



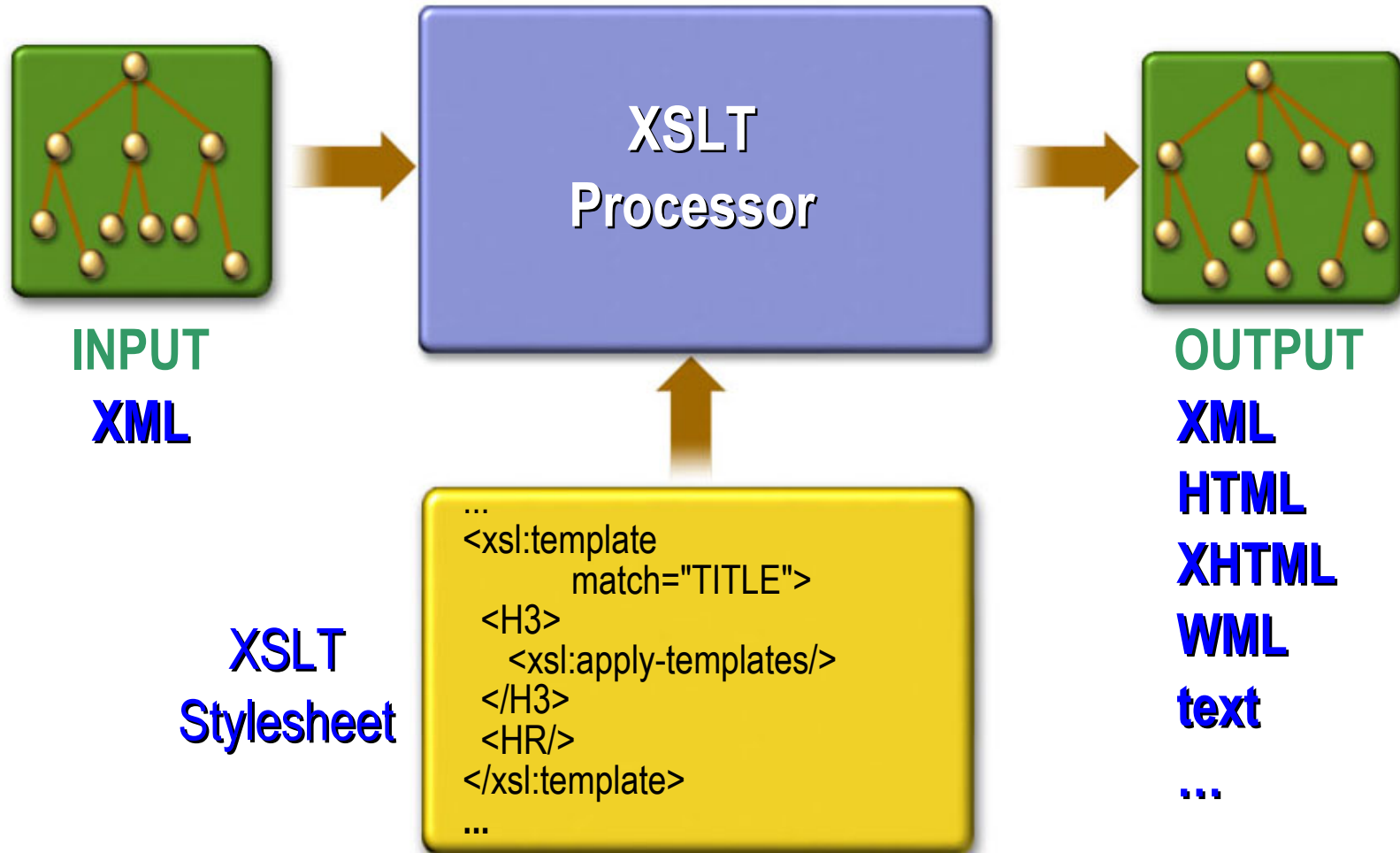
XSLT – in MOM Example #2



XSLT

Operational Model

XSLT Operational Model



What is XSLT Processor?

- Piece of software
 - > Reads an XSLT stylesheet and input XML document
 - > Converts the input document into an output document
 - > According to the instruction given in the XSLT stylesheet
- Called “stylesheet processor” sometimes

Examples of XSLT Processor

- Built-in within a browser
 - > Most browsers performs the XSLT transformation out of the box
- Built-in within web or application server
 - > Apache Cocoon
- Java packages
 - > JAXP
 - > Apache.org's Xalan

What is XSLT Stylesheet?

- Contains instruction of transformation
- Genuine XML document
- Root element typically is
 - > **stylesheet** or **transform**
 - > Both are defined in standard XSLT namespace
 - > <http://www.w3.org/XSL/Transform>
 - > xsl as customary prefix
 - > XSLT processor should understand both

Review on XPath

What is XPath?

- Used by XSLT (and by other XML technologies such as XPointer) for **referencing** elements and attributes of an XML document
- The “referencing” is done through **expression language (pattern)**
- Supports a **tree structure** expression
 - > Example: 7th child element of the third *person* element

XPath

- XPath expression results in a **node set**
 - > A node set of “*person*” elements under “*people*” element
- Various functions can be used on node sets, including:
 - > **not()** – eliminate a specific node
 - > **position()** – return the position within a node set
 - > **count()** – returns the number of nodes in a node set

XSLT Example 0

XML Example Document

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

Minimal (Empty) but Complete XSLT Stylesheet: example0.xsl

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

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

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


Result of XSLT Processing

```
<?xml version="1.0" encoding="utf-8"?>
```

```
Alan  
Turing
```

```
computer scientist  
mathematician  
cryptographer
```

```
Richard  
M  
Feynman
```

```
physicist  
Playing the bongoes
```

Explanation of the Result

- Applying empty stylesheet to any XML document
 - > Elements in the XML document are traversed sequentially
 - > Content of each element is put in output
 - > Attributes are NOT traversed
 - > Default behavior of “getting content of an element” is outputting the “text” of the traversed sub-elements
- Without any specific templates to apply to an element
 - > XSLT processor falls back to default behavior

xml-stylesheet Instruction

<?xml-stylesheet ..> Processing Instruction

- Can be included as part of XML document
- Tells **XML-aware browser** where to find associated stylesheet

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

```
<?xml-stylesheet
```

```
    type="text/xml"
```

```
    href="http://www.oreilly.com/styles/people.xsl"?>
```

```
<people>
```

```
....
```

Template

What does a Template do?

- Controls which output is created from which input
- `<xsl:template match="..">`
- `match=".."` attribute contains an **XPath expression**
 - > XPath expression identifies **input node set** it matches
- For each node in the node set, the **template contents** (things between `<xsl:template ..>` and `</xsl:template>` tags) are instantiated and inserted into the output

XSLT Example 1

Very Simple XSLT Stylesheet 1: example1.xsl

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

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

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

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

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

XML Example Document: people.xml (Source document)

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

Result of XSLT Transformation

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

Explanation of the Result

- `<xsl:template match="people">` in the XSLT stylesheet generates a result node set
 - > There is one `<people>` node in the result node set
- For each node in the node set, the **template contents** (things between `<xsl:template ..>` and `</xsl:template>` tags) are instantiated and inserted into the output
- The `<person>` node will be replaced with null since there is nothing between `<xsl:template>` .. and `</xsl:template>`

Lab:

**Exercise 1: Test “example0.xml”
stylesheet**

4343_ws_xml_xslt.zip

**(For the rest of the presentation,
try “examplex.xml” stylesheet)**



XSLT Example 2

Very Simple XSLT Stylesheet 2: example2.xsl

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

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

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

```
Folks in Brandeis XML class
```

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

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

Result

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

Folks in Brandeis XML class

XSLT Example 3

Very Simple XSLT Stylesheet 3: example3.xsl

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

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

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

```
A Person
```

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

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

XML Example Document

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

Result

```
<?xml version="1.0" encoding="utf-8"?>
```

A Person

A Person

- Whitespace outside of `<person>` element preserved
- `<person>` element is replaced by contents of template, `"A Person"`

Explanation of the Result

- `<xsl:template..>` in the XSLT stylesheet generates a result node set
 - > There are two `<person>` nodes in the result node set
- Each element in the input document is processed in sequence
 - > There is no template for `<people>` element, so default behavior is used
 - > There is a template for `<person>` element, so template is applied (instead of default behavior) - it will be replaced by the template content, which is "A Person"

XSLT Example 4

Very Simple XSLT Stylesheet 4: example4.xsl

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

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

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

A Person

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

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

- Same stylesheet with example 3 but with different input XML document (*people1.xml*, which is shown in the next slide)

New XML Example Document: people1.xml

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

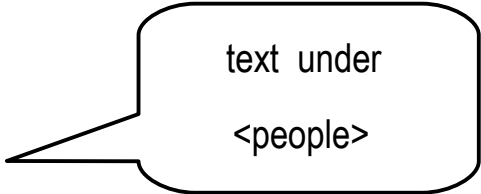
  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    ...
  </person>

  Some text here under people element!

  <clinton>
    Monica is under Clinton element!
  </clinton>

</people>
```



text under
<people>

Result

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

A Person



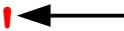
Template content

A Person



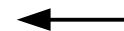
Template content

Some text here under people element!



Default behaviour
of <people>
element

Monica is under Clinton element!



Default behaviour
of <clinton>
element

Explanation

- Each element in the input document is processed in sequence
 - > There is no template for <people> element, so it is processed in default mode
 - > There is a template for <person>, so template is applied
 - > There is no template for <clinton> element, so it is processed in default mode

XSLT Example 5

A Simple XSLT Stylesheet: example5.xsl

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

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

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

```
<p>A Person</p>
```

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

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

Result

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<p>A Person</p>
```

```
<p>A Person</p>
```

- **Template content** contains tags (`<p>` in this example) and character data (*A Person* in this example)

<xsl-valueof ..>

What does `<xsl:value-of>` element do?

- Controls what gets generated as output (instead of generating text content as output as we've seen in the previous examples)
 - > Example: For each person element, display his/her first name in upper case

<xsl:value-of select=".."> element

- Extracts the **string value** of an element or an attribute and writes it to output
 - > string value - text content of the element after all the tags have been removed
- **select** attribute containing XPath expression identifies an element or an attribute
 - > It could be a node set, in which case, <xsl:value-of> will result in the string value of first node is taken

XSLT Example 6

example6.xsl

- For all <person> elements (node set), display string value of the <name> elements (node set)

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

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

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

```
    <p>
```

```
      <xsl:value-of select="name"/>
```

```
    </p>
```

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

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

For the sample XML Example Document

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

Result

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

```
<p>
```

Alan

Turing

```
</p>
```

```
<p>
```

Richard

M

Feynman

```
</p>
```

Explanation

- Create a node set A via *match="person"*
 - > Two *<person>* elements in the node set A
- For each *<person>* element in the node set A, create a node set B via *select="name"*
 - > One *<name>* element in each node set B
- Display string value of each node set B

```
<name>  
  <first_name>Alan</first_name>  
  <last_name>Turing</last_name>  
</name>
```

Alan
Turing

**XSLT Example 6a
(Use the same
example6.xml
stylesheet with
a new XML
document)**

example6.xsl (same one we saw before)

- For every <person> element (in a node set returned from match="person", display string value of the <name> element of the node set returned from select="name")

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

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

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

```
    <p>
```

```
      <xsl:value-of select="name"/>
```

```
    </p>
```

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

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

New XML Example Document: people2.xml

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <name>
      <given_name>Alan2</first_name>
      <surname>Turing2</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

The 2nd <name> element
is added for the 1st
<person> element

Result (Same as Example 6)

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

```
<p>
```

```
  Alan
```

```
  Turing
```

```
</p>
```

```
<p>
```

```
  Richard
```

```
  M
```

```
  Feynman
```

```
</p>
```

Explanation

- The first `<person>` element has two `<name>` elements in the node set
- Displaying string value of a node set takes the first node in the set and display text value of it

```
<name>
  <first_name>Alan</first_name>
  <last_name>Turing</last_name>
</name>
<name>
  <given_name>Alan2</given_name>
  <surname>Turing2</surname>
</name>
```

Alan
Turing

Review of what we learned so far

- `<xsl:template match="element-name">`
 - > Create a node set using *match="element-name"*
 - > For each node in the node set, replace it with what is specified within the `<xsl:template ..>` and `</xsl:template>`
- `<xsl:value-of select="element-name"/>`
 - > Create a node set using *select="element-name"*
 - > Display the string value of each node in the node set

xsl:apply-templates

What does `xsl:apply-templates` do?

- Controls which child nodes of a context node a template is applied to

XSLT Example 7

<xsl:apply-templates> Example

- I would like the output to look like as following
 - > Last name then first name
 - > Only name not profession nor hobby

```
<?xml version="1.0" encoding="utf-8"?>
```

Turing

Alan

Feynman

Richard

Let's say we use example7.xsl

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

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

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

```
  <xsl:value-of select="last_name"/> ,
```

```
  <xsl:value-of select="first_name"/>
```

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

```
<!-- Something is missing here -->
```


For the sample XML Example Document..

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

  <person born="1912" died="1954">
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
    <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>

  <person born="1918" died="1988">
    <name>
      <first_name>Richard</first_name>
      <middle_initial>M</middle_initial>
      <last_name>Feynman</last_name>
    </name>
    <profession>physicist</profession>
    <hobby>Playing the bongoes</hobby>
  </person>

</people>
```

Result (Different from what we want)

```
<?xml version="1.0" encoding="utf-8"?>
```

Turing

Alan

computer scientist

mathematician

cryptographer



We don't want this

Feynman

Richard

physicist

Playing the bongoes



We don't want this

Explanation of result

- Each element in the input document is processed in sequence – any template applicable to each element is then applied
 - > There is no template for <people> and <person> elements – so they are processed in default mode
 - > There is a template for <name> element – so template is applied
 - > There is no template for other elements (<profession>, <hobby>) – so they are processed in default mode
- We do not want other elements (<profession>, <hobby>) to be processed
 - > This is where *<xsl:apply-templates>* can be useful

XSLT Example 8

<xsl:apply-templates> example8.xsl

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

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

```
<xsl:template match="name">  
  <xsl:value-of select="last_name"/>,  
  <xsl:value-of select="first_name"/>  
</xsl:template>
```

```
<!-- Apply templates only to name children of person -->
```

```
<xsl:template match="person">  
  <xsl:apply-templates select="name"/>  
</xsl:template>
```

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

Result

```
<?xml version="1.0" encoding="utf-8"?>
```

Turing

Alan

Feynman

Richard

Explanation of result

- Each element in the input document is processed in sequence – any template applicable to each element is then applied
 - > There is a template for <person> element
 - > The template of <person> element says “Search for a template for only <name> child element and apply it!”
 - > There is a template for <name> child element and it is applied

XSLT Example 8a

xsl:apply-templates: example8a.xsl – order of two templates are reversed

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

  <!-- Apply templates only to name children -->
  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

  <xsl:template match="name">
    <xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/>
  </xsl:template>

</xsl:stylesheet>
```

Result (Same result as with example8.xsl)

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

Turing,
Alan

Feynman,
Richard

Explanation of result

- Ordering of `<xsl:template>` within a stylesheet does not matter
- Elements of document instance are still processed from root to its child nodes – so order of elements in document instance matter

XSLT Example 8b

xsl:apply-templates: example8b.xsl

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

  <xsl:template match="name">
    <xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/>
  </xsl:template>

  <xsl:template match="profession">
    My profession!
  </xsl:template>

  <!-- Apply templates only to name children -->
  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>

</xsl:stylesheet>
```

Result

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

Turing,
Alan

Feynman,
Richard

Explanation of result

- The template of <profession> does not get applied because template of <person> says only the template of <name> should be applied

XSLT Example 8c

xsl:apply-templates: example8c.xsl

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

```
  <xsl:template match="name">  
    <xsl:value-of select="last_name"/>,  
    <xsl:value-of select="first_name"/>  
  </xsl:template>
```

```
  <xsl:template match="profession">  
    My profession!  
  </xsl:template>
```

```
  <!-- Apply templates to all child nodes of person -->  
  <xsl:template match="person">  
    <xsl:apply-templates/>  
  </xsl:template>
```

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

Result

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

Turing,
Alan

My profession!

My profession!

My profession!

Feynman,
Richard

My profession!

Playing the bongoes

Explanation of result

- The templates of both <name> and <profession> get applied because the templates of all child nodes of <person> are to be applied
- Since there is no template for <hobby>, default behavior is taken

XSLT Example 9

example9.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="people">
    <html>
      <head><title>Famous Scientists</title></head>
      <body> <xsl:apply-templates/> </body>
    </html>
  </xsl:template>
  <xsl:template match="person">
    <xsl:apply-templates select="name"/>
  </xsl:template>
  <xsl:template match="name">
    <p><xsl:value-of select="last_name"/>,
    <xsl:value-of select="first_name"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Result

```
<html>  
<head><title>Famous Scientists</title></head>  
<body>  
  
  <p>Turing,  
    Alan</p>  
  
  <p>Feynman,  
    Richard</p>  
  
</body>  
</html>
```

Explanation of result

- Replace every *people* element with *html* element
- The template of `<person>` says to apply the template of `<name>` for all person nodes
- The template of `<name>` says to display last name and first name

Attributes

Attributes: example10.xsl

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

  <xsl:template match="people">
    <html>
      <head><title>Famous Scientists</title></head>
      <body>
        <dl>
          <xsl:apply-templates/>
        </dl>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="person">
    <dt><xsl:apply-templates select="name"/></dt>
    <dd><ul>
      <li>Born: <xsl:apply-templates select="@born"/></li>
      <li>Died: <xsl:apply-templates select="@died"/></li>
    </ul></dd>
  </xsl:template>

</xsl:stylesheet>
```

Attributes

- Default rule does not apply
 - > *apply-templates* has to be present in order to output values of attributes

Result

```
<html>
  <head>
    <title>Famous Scientists</title>
  </head>
  <body>
    <dl>

      <dt>
        Alan
        Turing
      </dt>
      <dd>
        <ul>
          <li>Born: 1912</li>
          <li>Died: 1954</li>
        </ul>
      </dd>
      ..
    </dl>
  </body>
</html>
```

XSLT vs. other Technologies

XSLT and DOM

- Most XSLT engine uses DOM internally
 - > Reason for slow performance and high memory requirement
- DOM could be used for transformation as well
 - > DOM does NOT provide any ready-to-use XPath functionality however
 - > XSLT is completely declarative
 - > XSLT is more portable than DOM

XSLT vs. Programming

- Programming is useful when you do more than transformation
- Examples
 - > Interpreting certain elements as database queries
 - > Inserting the query results into output document
 - > Asking users questions in the middle of transformation

Summary

Summary

- XSLT is useful to both POP and MOM
- XSLT Stylesheet Language
 - > `<xsl:template match="XPath">`
 - > `<xsl:value-of select="XPath"/>`
 - > `<xsl:apply-templates select="XPath"/>`

References

- “XML in a Nutshell” written by Elliotte Rusty Harold & W. Scott Means, O’Reilly, Jan. 2001(1st Edition), Chapter 8 “XSL Transformation”
- Apache.Org, Xalan
- JAXP 1.1

Learn with Passion!
JPassion.com

