XSLT

Topics

- Terms: XSL, XSLT, XSL-FO
- Why Transformation?
- XSLT Operational Model
- A bit of Xpath
- XSLT Stylesheet Language
- Apache Xalan

Topics

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

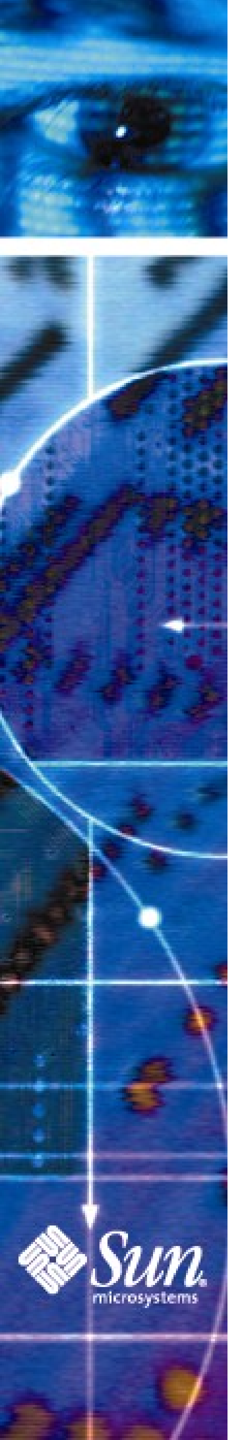
Terminology

XSL

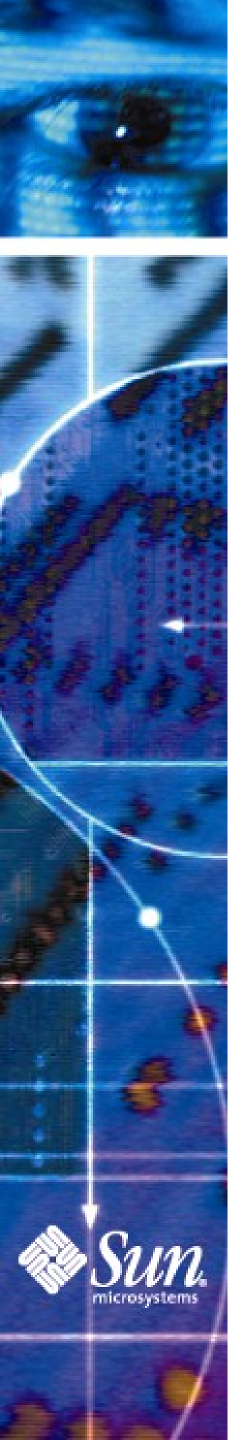
- eXtensible Stylesheet Language
- A language for expressing stylesheets
- Made of two parts
  - XSL Transformation (XSLT)
  - XSL Formatting Objects (XSL-FO)

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

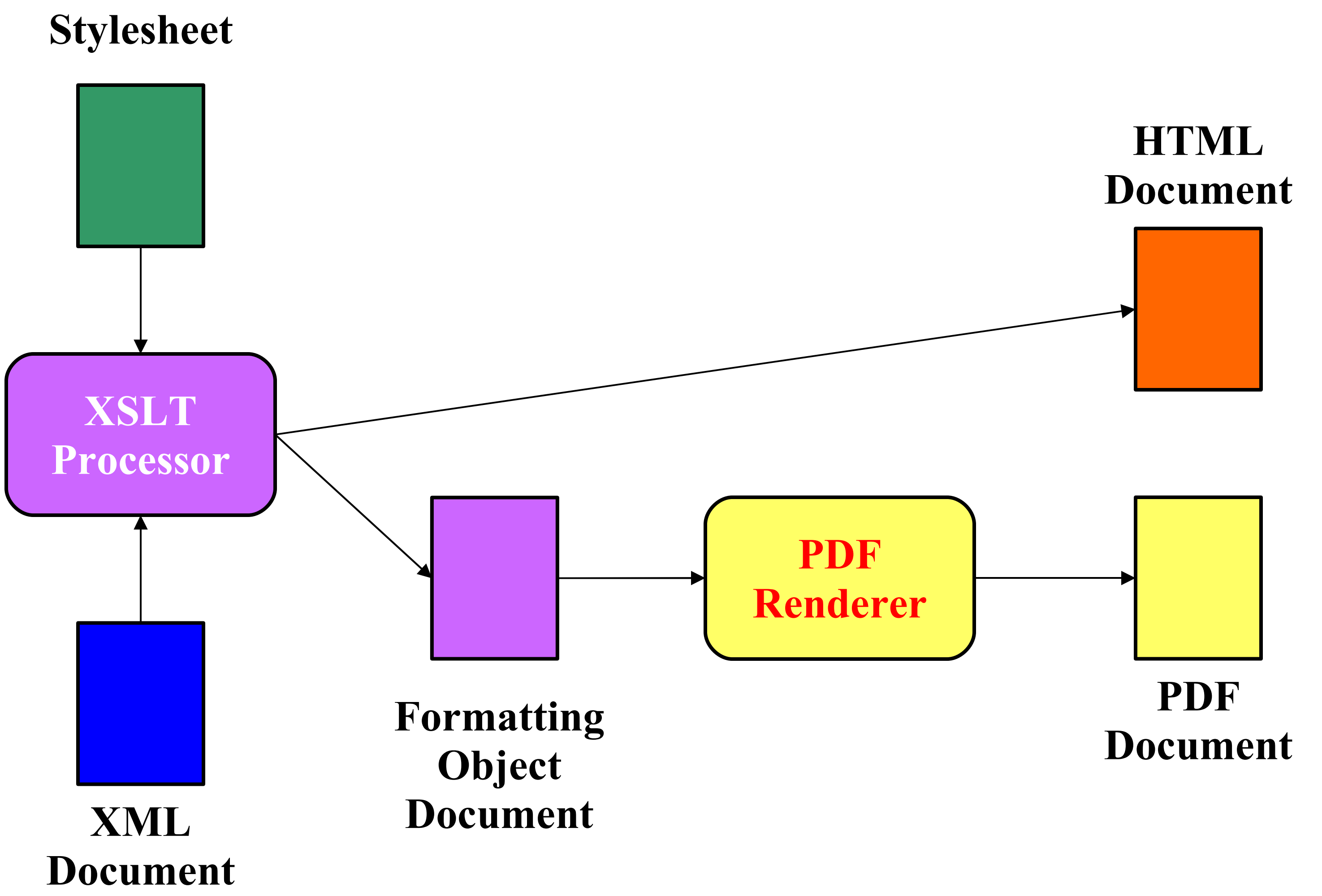
Importance of Transformation

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

XSLT in POP

- XML document separates content from presentation
- Transformations can be used to style (render, present) XML documents
- A common styling technique presents XML in HTML format

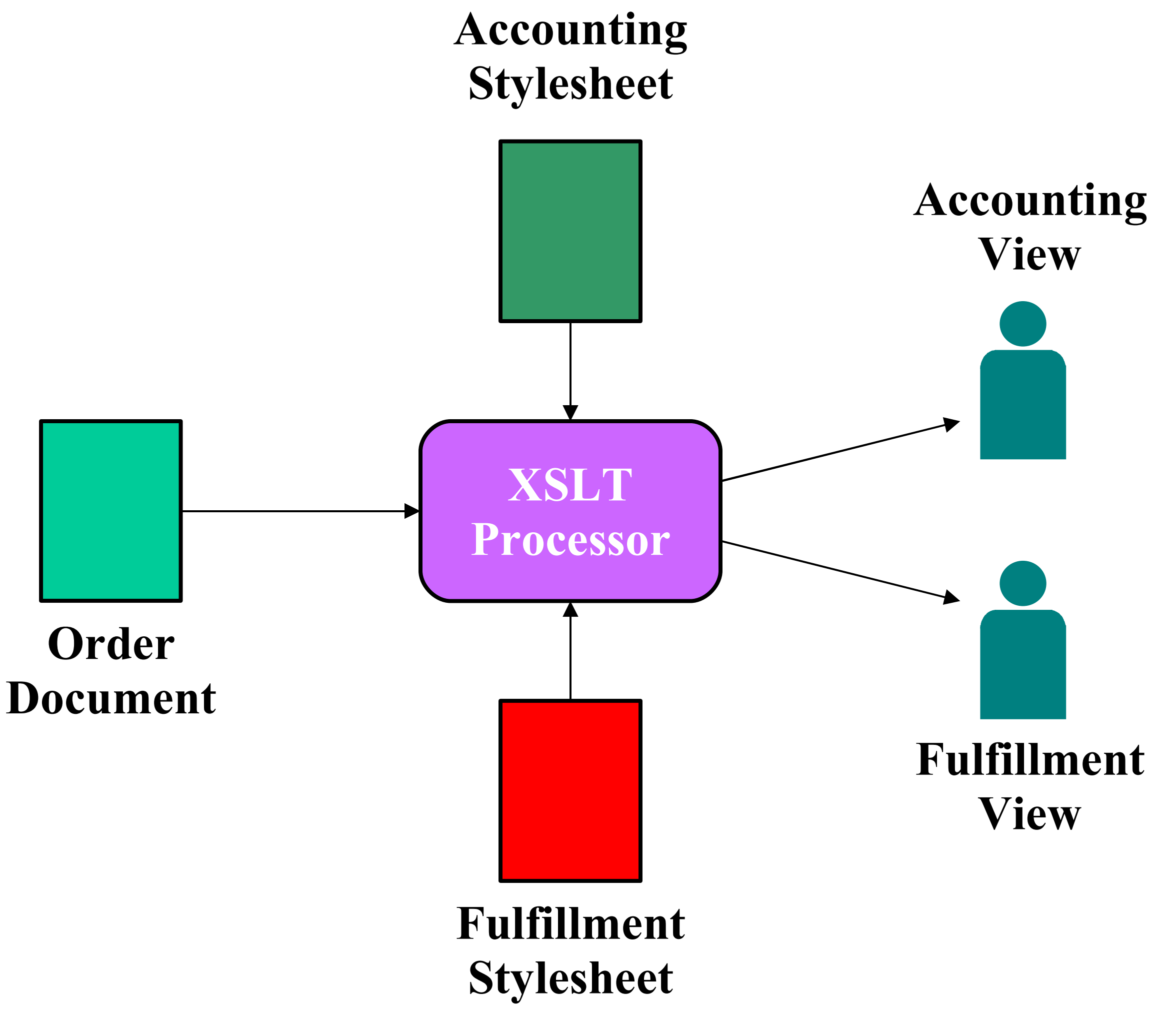
XSLT – in POP


Stylesheet
HTML Document
XSLT Processor
PDF Renderer
Formatting Object Document
PDF Document
XML Document

XSLT in MOM

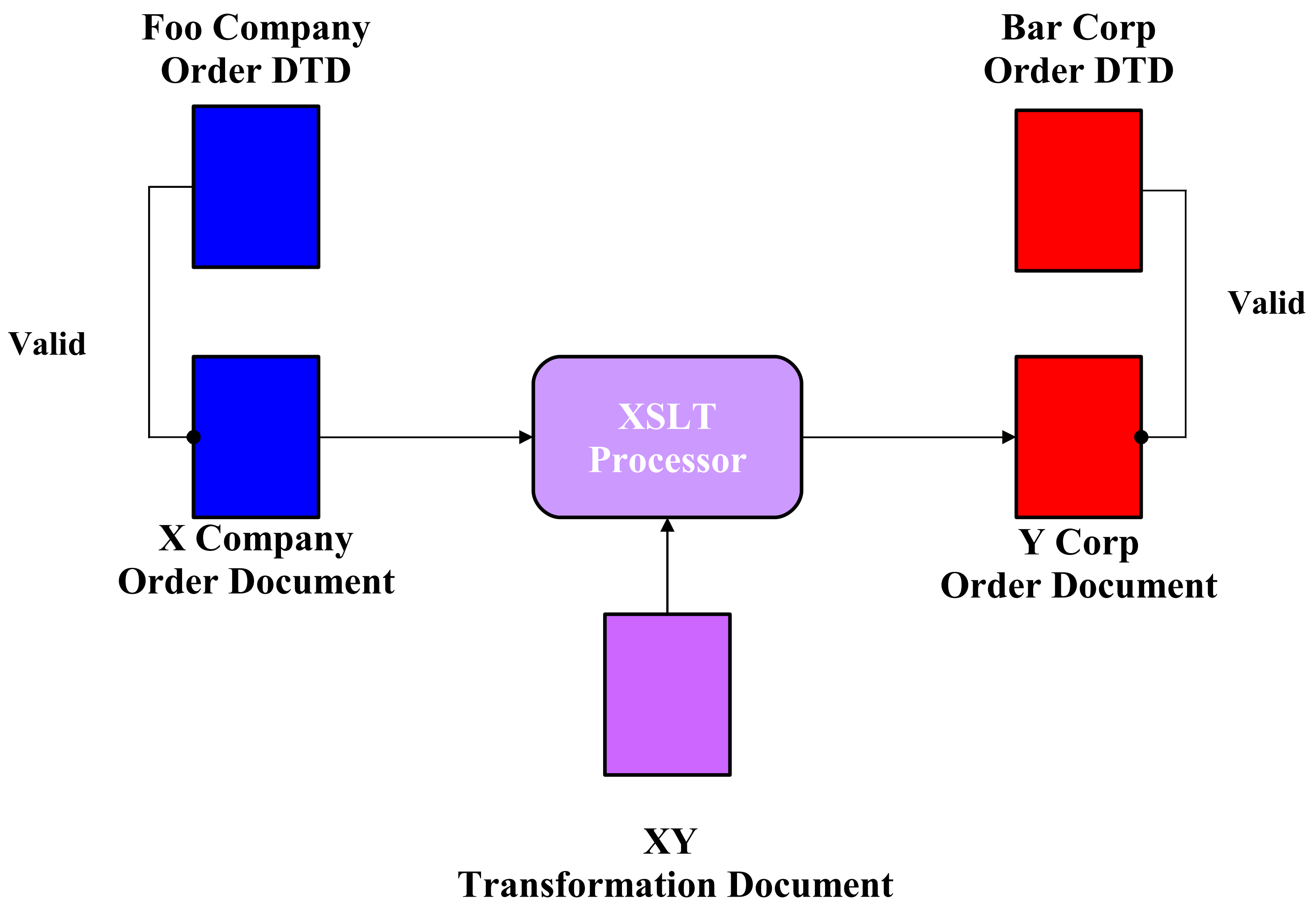
- Important for eCommerce, B2B/EDI, and dynamic content generation
  - Different content model
  - Different structural relationship
  - Different vocabularies

XSLT – in MOM



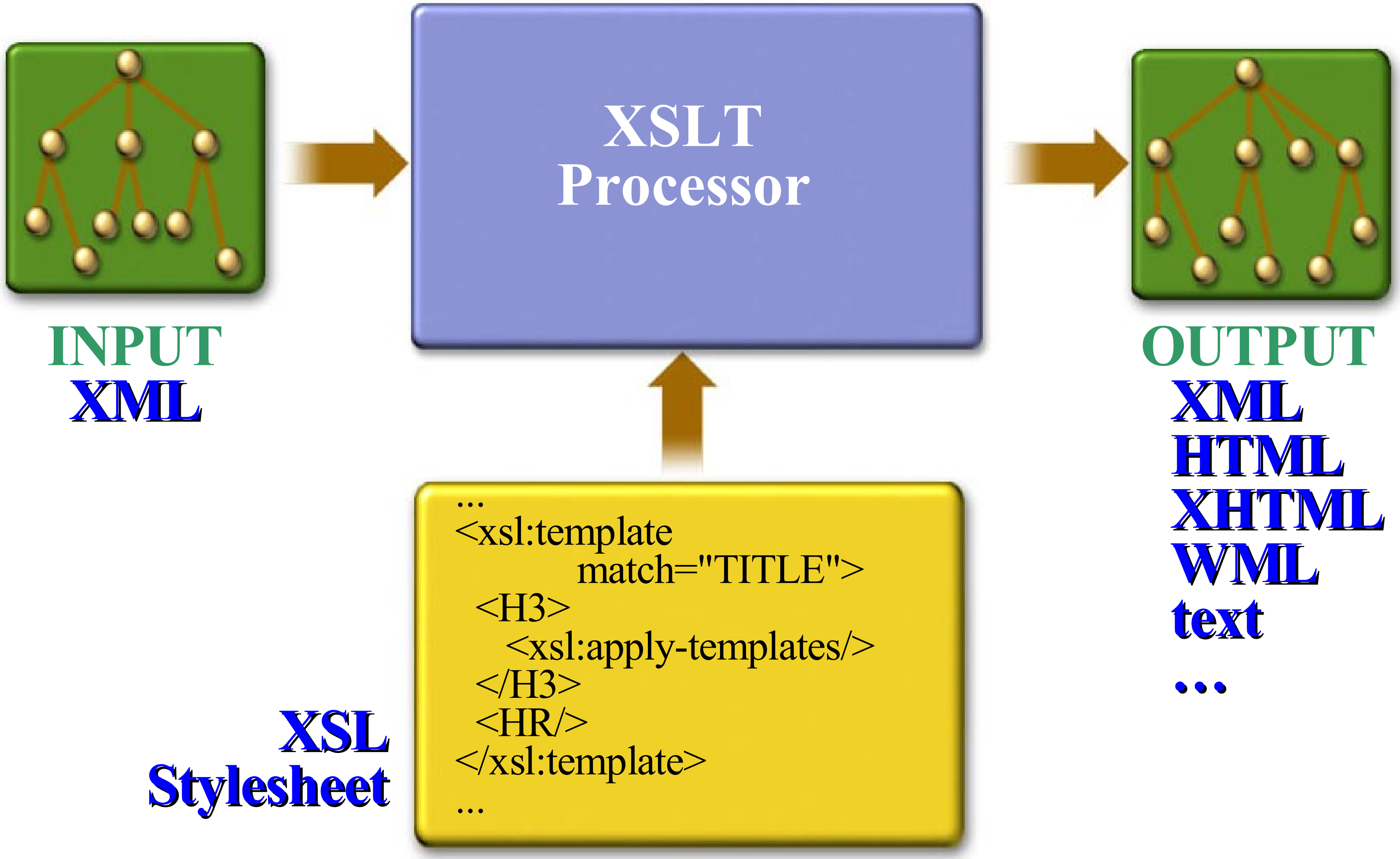
Accounting Stylesheet
Accounting View
XSLT Processor
Order Document
Fulfillment View
Fulfillment Stylesheet

XSLT – Data Transformation



XSLT Operational Model

XSLT Operational Model



```
...
<xsl:template
           match="TITLE">
  <H3>
    <xsl:apply-templates/>
  </H3>
  <HR/>
</xsl:template>
...
```

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 stylesheet
- Called stylesheet processor sometimes

Examples of XSLT Processor

- Built-in within a browser
  - IE 5.5 (not compatible to XSLT standard)
- Built-in within web or application server
  - Apache Cocoon
- Standalone
  - Michael Kay's SAXON
  - Apache.org's Xalan

XSLT Stylesheet

- 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

XPath

XPath

- Used by XSLT (and by other XML technologies such as XPointer) for referencing elements and attributes internal to an XML document
- Defines expression language (pattern) for referencing
- 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
<?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 but Complete XSLT Stylesheet

```
<?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 are traversed sequentially
  - Content of each element is put in output
    - Attributes are NOT traversed
  - Default behavior
- Without any specific templates
  - XSLT processor falls back to default behavior
- Need for templates

xml-stylesheet Instruction

xml-stylesheet Processing Instruction

- Included as part of XML document
- Tells XML-ware 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

Templates

- Controls which output is created from which input
  - *xsl:template* element form
  - *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 tags) are instantiated and inserted into the output tree

XSLT
Example 1

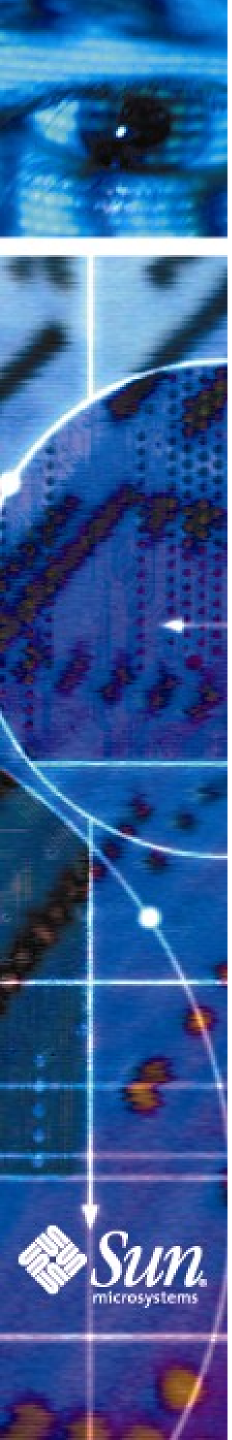
Very Simple XSLT Stylesheet 1

```
<?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>
```

- Simplest form of XPath pattern is a name of a single element



XML Example Document

```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">
    <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"?>
```

Explanation of the Result

- There is one node in the result node set – there is only one <people> element
- For the node, it will be replaced by the template content, which is “null”

XSLT
Example 2

Very Simple XSLT Stylesheet 2

```
<?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

```
<?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>
```

- Literal data characters - text copied from the stylesheet into the output document

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

XSLT Example 4

Very Simple XSLT Stylesheet 4

```
<?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

New 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">
    ...
  </person>

  Some text here under people element!

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

</people>
```

Result

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


  A Person                           <--- template content

  A Person                           <--- template content

  Some text here under people elelemt!  <-- default


  Monica is under Clinton element!      <-- default
```

XSLT
Example 5

A Simple XSLT Stylesheet

```
<?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>
```

- **Literal result elements** - elements copied from stylesheet to output document

Result

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


  <p>A Person</p>

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

- Template content contains tags and character data

xsl-valueof

xsl:value-of element

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

XSLT
Example 6

Example Stylesheet

- Extract names of all the people

```
<?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>
```

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
        Feyman
     </p>
```

xsl:apply-templates

xsl:apply-templates

- XSLT processor reads (traverses) the input XML document sequentially from top to bottom
- Templates are activated in the order they match elements encountered
  - Template for a parent will be activated before the children

xsl:apply-templates

- The order of the traversal can be changed by apply-templates
  - It can specify which element or elements should be processed next
  - It can specify an element or elements should be processed in the middle of processing another element
  - It can prevent particular elements from being processed

xsl:apply-templates

- *xsl:apply-templates* lets you make your choice of processing order explicit
- *select* attribute contains XPath expression telling the XSLT processor which nodes to process in the input tree
  - The *apply-templates* with no *select* attribute means all elements relative to the current element (context node) should be matched

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

    Feyman
        Richard
```

xsl:apply-templates Example

```
<?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 -->

</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"?>

   Turing
      Alan

  computer scientist
  mathematician
  cryptographer

   Feyman
      Richard

  physicist
  Playing the bongoes
```

Explanation

- Two <name> elements in the node set
- The <xsl:value-of> contents of the two <name> elements will be in the output tree
- Other elements are displayed in default mode

XSLT Example 8

xsl:apply-templates Example

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

</xsl:stylesheet>
```

xsl:apply-templates Example

```
<?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>
```

- Order of templates does not matter

Result

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

    Turing
      Alan

    Feyman
      Richard
```

xsl:apply-templates

- Also useful when child elements have templates of their own

```
<xsl:template match="people">
  <html>
   <head><title>Famous Scientists</title></head>
    <body>
      <xsl:apply-templates select=“person”/>
    </body>
  </html>
</xsl:template>
```

xsl:apply-templates

- Replace every *people* element with *html* element
- Process all *person* children of the current *people* element
- Insert the output of any matched templates into the output document’s *body* element

Example

```
<?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>
```

Attributes

Attributes

```
<?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>
<dt>
            Richard
            M
            Feynman

         </dt>
         <dd>
            <ul>
               <li>Born: 1918</li>
               <li>Died: 1988</li>
            </ul>
         </dd>


      </dl>
   </body>
</html>
```

Modes

Modes

- Same input content needs to appear multiple times in the output document formatted according to different template
  - Titles of chapters
    - Table of contents
    - In the chapters themselves
- mode attribute
  - xsl:template
  - xsl:apply-templates

Example with mode attribute

```
<?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>
    <ul><xsl:apply-templates select="person" mode="toc"/></ul>
      <xsl:apply-templates select="person"/>
   </body>
  </html>
 </xsl:template>

 <!-- Table of Contents Mode Templates -->
 <xsl:template match="person" mode="toc">
  <xsl:apply-templates select="name" mode="toc"/>
 </xsl:template>

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

 <!-- Normal Mode Templates -->
 <xsl:template match="person">
  <p><xsl:apply-templates/></p>
 </xsl:template>

</xsl:stylesheet>
```

Result

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

      Alan
      Turing

    computer scientist
    mathematician
    cryptographer
  </p>
```

```
<p>

      Richard
      M
      Feynman

    physicist
    Playing the bongoes
  </p>
</body>
</html>
```

Filtering

Filtering

- So far we either process all the elements relative to a node or one element
- We need a way to filter out elements as well
- This is done with an XPath control structure

Example of Filtering

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


  <xsl:template match="person">
    <xsl:apply-templates select="*[not(self::hobby)]"/>
  </xsl:template>


</xsl:stylesheet>
```

- The *self* keyword is needed to inform the XSLT processor that the node following is a child of the current one

Result

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




   Alan
   Turing
 computer scientistmathematiciancryptographer


   Richard
   M
   Feynman
 physicist
```

xsl:for-each

xsl:for-each

- iterating through a node set
- **<xsl:for-each></xsl:for-each>**

Example of xsl:for-each

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

  <xsl:template match="people">
      <xsl:for-each select="person">
          <xsl:value-of select="name"/>
          <xsl:value-of select="@born"/>
      </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Result

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

      Alan
      Turing
    1912
      Richard
      M
      Feynman
    1918
```

xsl-if

xsl:if

- We can test content for certain values with XSL:
  - `<xsl:if test=`*`criteria`*`></xsl:if>`
- The *test* attribute is required and will either be true or false

Example of xsl:if

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

   <xsl:template match="people">
       <xsl:for-each select="person">
         <xsl:value-of select="name"/>
         <xsl:if test="@born='1912'">
             Died in
             <xsl:value-of select="@died"/>
         </xsl:if>
       </xsl:for-each>
   </xsl:template>

</xsl:stylesheet>
```

Result

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

      Alan
      Turing

           Died in
       1954
      Richard
      M
      Feynman
```

xsl:choose

xsl:choose, xsl:when, xsl:otherwise

- We can also select content using:

```
<xsl:choose>
   <xsl:when test=criteria>
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
</xsl:choose>
```

- The **test** attribute works in the same fashion as **xsl:if**

xsl:choose, xsl:when, xsl:otherwise

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


  <xsl:template match="people">
     <xsl:for-each select="person">
       <xsl:value-of select="name"/>
       <xsl:choose>
          <xsl:when test="@born='1912'">
             Died in <xsl:value-of select="@died"/>
          </xsl:when>
          <xsl:otherwise>
             Did not die in 1912
          </xsl:otherwise>
       </xsl:choose>
    </xsl:for-each>
  </xsl:template>


</xsl:stylesheet>
```

Result

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

      Alan
      Turing

              Died in 1954
      Richard
      M
      Feynman

              Did not die in 1912
```

xsl:sort

xsl:sort

- XSLT provides a nice way to sort documents by element contents
- The construct to use is:

```
<xsl:sort select=selection></xsl:sort>
```

- Sorting can only be done in the following constructs:
  - `<xsl:apply-templates…/>`
  - `<xsl:for-each …/>`

Example of xsl:sort ("Ascending")

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

   <xsl:template match="people">
     <xsl:apply-templates>
         <xsl:sort select="name"/>
     </xsl:apply-templates>
   </xsl:template>

</xsl:stylesheet>
```

Result

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

      Alan
      Turing

    computer scientist
    mathematician
    cryptographer

      Richard
      M
      Feynman

    physicist
    Playing the bongoes
```

Example of xsl:sort ("Descending")

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

   <xsl:template match="people">
     <xsl:apply-templates>
        <xsl:sort select="name"
   order="descending" />
     </xsl:apply-templates>
   </xsl:template>

</xsl:stylesheet>
```

Result

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

      Richard
      M
      Feynman

    physicist
    Playing the bongoes

      Alan
      Turing

    computer scientist
    mathematician
    cryptographer
```

xsl:copy

xsl:copy

- Used for creating an XML Document
- The copying is done using this construct:

```
<xsl:copy></xsl:copy>
```

- We will also specify to the processor that our output should be XML instead of HTML

```
<xml:output method="xml"/>
```

Example of xsl:copy

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

  <xsl:output method="xml"/>

  <xsl:template match=”people">
    <xsl:copy>
      <xsl:apply-templates>
        <xsl:sort select="name"/>
      </xsl:apply-templates>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Result

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

      Richard
      M
      Feynman

    physicist
    Playing the bongoes



      Alan
      Turing

    computer scientist
     mathematician
    cryptographer

</people>
```

Example 2 of xsl:copy

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

   <xsl:output method="xml"/>

   <xsl:template match="*">
     <xsl:copy>
       <xsl:apply-templates>
          <xsl:sort select="name"/>
       </xsl:apply-templates>
     </xsl:copy>
   </xsl:template>

</xsl:stylesheet>
```

Result

```
<?xml version="1.0" encoding="UTF-8"?>
<people><person>
    <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><person>
    <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
    </name>
    <profession>computer scientist</profession>
     <profession>mathematician</profession>
    <profession>cryptographer</profession>
  </person>
</people>
```

Apache Xalan

Apache Xalan

- Implements XSLT 1.0 and Xpath 1.0
- Can be run from both the command line and within application code
- Support scripting extension
- Command line syntax:

```
java org.apache.xalan.xslt.Process
-IN <input document>
-XSL <stylesheet>
-OUT <output document>
```

Xalan Demo

- Class materials
- Xalan built-in demos

Xalan in Application

- Applet wrapper
- Can be used in a servlet, JSP
- EJB code

Programming API

- Input (Source tree)
  - File, Character stream, Byte stream
  - DOM
  - SAX input stream
- Output (Result tree)
  - File, Character stream, Byte stream
  - DOM
  - SAX events

Programming API using Xalan

```
// Have the XSLTProcessorFactory obtain a interface to a
// new XSLTProcessor object.
XSLTProcessor processor =
    XSLTProcessorFactory.getProcessor();

// Have the XSLTProcessor processor object transform
// "foo.xml" to System.out, using the XSLT instructions
// found in "foo.xsl".
processor.process(new XSLTInputSource("foo.xml"),
                  new XSLTInputSource("foo.xsl"),
                  new XSLTResultTarget(System.out));
```

Programming API using JAXP 1.1

```
TransformerFactory tf
= TransformerFactory.newInstance();
Transformer transformer =
= tf.newTransformer(new StreamSource(“foo.xsl”);

transformer.transform(
        new StreamSource(“foo.xml”),
        new StreamSource(“bar.xml”));
```

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
  - 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
- Apache Xalan

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