NPRG039

Advanced Aspects and New Trends in XML (and Related) Technologies

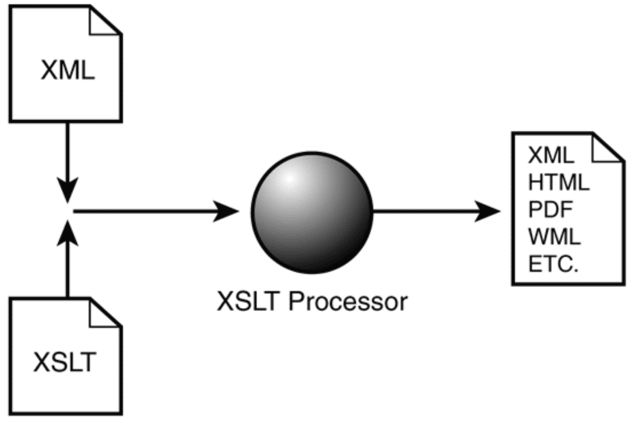
RNDr. Irena Holubová, Ph.D.

holubova@ksi.mff.cuni.cz

Lecture 9. Advances in XSLT

http://www.ksi.mff.cuni.cz/~svoboda/courses/171-NPRG039/

XSLT Processing



- Idea:
  - XSLT processor parses the input XML document and the input XSLT script
    - The root node of the document is stored into a context set
  - It applies suitable templates from the script to the context set until it is non-empty
    - The context set can change during application of a current template
      - New nodes can be added for processing
    - If there are multiple applicable templates, the one with the highest priority is applied
      - User-specified/implicit
    - If there is no suitable template an implicit template is used

XSLT Versions

- XSLT 1.0
  - W3C Recommendation 1999
  - http://www.w3.org/TR/xslt
- XSLT 2.0
  - W3C Recommendation 2007
  - http://www.w3.org/TR/xslt20/
- XSLT 3.0
  - Candidate Recommendation November 2015
  - http://www.w3.org/TR/xslt-30/
- Note:
  - Working groups
    - Working draft → candidate recommendation → proposed recommendation → recommendation
  - XML Prague conference

xmlprague
a conference on XML and data on the web

http://www.xmlprague.cz/

What's New in XSLT 2.0?

- Biggest change: XPath 1.0 $\rightarrow$ XPath 2.0
  - Works with XML Schema data types
  - Everything is a sequence
  - Supports loops and if clauses
  - Involves a huge set of built-in functions
  - …
- Output into multiple files
- Grouping of nodes
- User-defined functions
- Regular expressions
- XHTML output
- …

What’s New in XSLT 2.0?

XPath 2.0 For Loop

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
  <xsl:template match="/">
    <html>
      <head></head>
      <h1>Order items:</h1>
        <xsl:for-each select="order/item">
          <tr>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="number"/></td>
            <td><xsl:value-of select="price"/></td>
            <td><xsl:value-of select="number * price"/></td>
          </tr>
        </xsl:for-each>
        <tr>
          <th>Total:</th>
          <th colspan="3">
            <xsl:value-of select="sum(for $n in order/item
                                       return $n/price * $n/number)"/>
          </th>
        </tr>
      </table>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

What’s New in XSLT 2.0?

XPath 2.0 If Clause

```
...
<xsl:template match="item">
  <tr bgcolor="{if (position() mod 2 = 0)
                then '#FF8000'
                else '#FFC0C0'}">
    <xsl:apply-templates select="name|cathegory|price"/>
  </tr>
</xsl:template>
...
```

- In both cases usually more compact than the XSLT clauses **`xsl:for-each`** and **`xsl:if`** / **`xsl:choose`**
  - Note: There is no **`xsl:else`** clause for **`xsl:if`** in XSLT

What's New in XSLT 2.0?

XPath 2.0

- Less restricted grammar
  - e.g., `/book/(chapter|appendix)/title`
- Functions can be applied as a part of path
  - e.g., `/catalogue/item/name/upper-case(.)`
- We can refer to an element/attribute in any namespace using `*:local_name`
  - e.g., `<xsl:template match="*:trkpt|*:wpt|*:rtept">`
- Quantifiers (`some` / `every`)
  - e.g., `if (every $j in (2 to $i – 1) satisfies $i mod $j ne 0) then $i else ()`
  - Extend the original = operator with implicit `some` quantifier
- Operator `is` for testing identity of two nodes
- Set operators (`union`, `except`, `intersect`)
- …

What’s New in XSLT 2.0?
Output into Multiple Files

- Element **`xsl:result-document`**
  - Attribute **`href`**
    - URL of output document
  - Attribute **`format`**
    - Format of the output document
    - Reference to an **`xsl:output`** element
- Element **`xsl:output`**
  - Attribute **`name`**
    - To enable referencing

What's New in XSLT 2.0?

Output into Multiple Files

XHTML type of output

```
<xsl:output name="orders-report-format" method="xhtml" .../>
<xsl:output name="order-format"         method="xml" ... />

<xsl:template match="/">
 <xsl:result-document href="orders-report.html"
                      format="orders-report-format">
  <html>
   <body><xsl:apply-templates /></body>
  </html>
 </xsl:result-document>

 <xsl:for-each select="document('orders.xml')//order">
  <xsl:result-document href="order{./@number}.html"
                       format="order-format">
   <xsl:apply-templates select="." />
  </xsl:result-document>
 </xsl:for-each>
</xsl:template>
```

other than the default input

What's New in XSLT 2.0?
Grouping of Nodes

- Grouping of nodes according to specific conditions
- Element **`xsl:for-each-group`**
  - Divides nodes into groups
  - Performs its body for each group
- Attributes:
  - **`select`** – like for **`xsl:for-each`**
  - **`group-by`** – XPath expression specifying values according to which we should group using value equivalence
    - No restriction for the XPath expression is mentioned in the specification
  - **`group-adjacent`** – same, but we group only adjacent nodes according to the value
  - **`group-starting-with`** – identifies groups according to starting nodes
    - A separate group is created for the nodes before the first matching node
  - **`group-ending-with`** – identifies groups according to ending nodes
    - A separate group is created for the nodes after the last matching node
- Compare with **`GROUP BY`** and **`HAVING`** in SQL

What’s New in XSLT 2.0?

Grouping of Nodes

- Functions:
  - **`current-group()`** – returns items in the current group
  - **`current-grouping-key()`** – returns the grouping key of the current group
    - i.e., the value equivalent for all members in the group
    - For **`group-by`** and **`group-adjacent`**

What’s New in XSLT 2.0?

Grouping of Nodes

```
<xsl:template match="quotations">
  <html>
    <head>
      <title>Quotations According to Authors</title>
    </head>
    <body>
      <xsl:for-each-group select="quotation" group-by="author">
        <xsl:sort select="current-grouping-key()" lang="cs"/>
        <h1><xsl:value-of select="current-grouping-key()"/></h1>
        <xsl:for-each select="current-group()">
          <xsl:sort select="text" lang="cs"/>
          <p>
            <xsl:value-of select="text"/>
          </p>
        </xsl:for-each>
      </xsl:for-each-group>
    </body>
  </html>
</xsl:template>
```

Example

```
<body>
  <h2>Introduction</h2>
  <p>XSLT is used to write stylesheets.</p>
  <p>XQuery is used to query XML databases.</p>
  <h2>What is a stylesheet?</h2>
  <p>A stylesheet is an XML document used to define a
transformation.</p>
  <p>Stylesheets may be written in XSLT.</p>
  <p>XSLT 2.0 introduces new grouping constructs.</p>
</body>
```

```
<chapter>
  <section title="Introduction">
    <para>XSLT is used to write stylesheets.</para>
    <para>XQuery is used to query XML databases.</para>
  </section>
  <section title="What is a stylesheet?">
    <para>A stylesheet is an XML document used to define a
transformation.</para>
    <para>Stylesheets may be written in XSLT.</para>
    <para>XSLT 2.0 introduces new grouping constructs.</para>
  </section>
</chapter>
```

Solution

```
<xsl:template match="body">
  <chapter>
    <xsl:for-each-group select="*" group-starting-with="h2">
      <section title="{self::h2}">
        <xsl:for-each select="current-group()[self::p]">
          <para><xsl:value-of select="."/></para>
        </xsl:for-each>
      </section>
    </xsl:for-each-group>
  </chapter>
</xsl:template>
```

Example

```
<doc>
  <page continued="yes">Some text</page>
  <page continued="yes">More text</page>
  <page>Yet more text</page>
  <page continued="yes">Some words</page>
  <page continued="yes">More words</page>
  <page>Yet more words</page>
</doc>
```

```
<doc>
  <pageset>
    <page>Some text</page>
    <page>More text</page>
    <page>Yet more text</page>
  </pageset>
  <pageset>
    <page>Some words</page>
    <page>More words</page>
    <page>Yet more words</page>
  </pageset>
</doc>
```

Solution

```
<xsl:template match="doc">
  <doc>
    <xsl:for-each-group
        select="*"
        group-ending-with="page[not(@continued='yes')]">
      <pageset>
        <xsl:for-each select="current-group()">
          <page><xsl:value-of select="."/></page>
        </xsl:for-each>
      </pageset>
    </xsl:for-each-group>
  </doc>
</xsl:template>
```

Example

Element in Multiple Groups

```
<titles>
  <title>A Beginner's Guide to <ix>Java</ix></title>
  <title>Learning <ix>XML</ix></title>
  <title>Using <ix>XML</ix> with <ix>Java</ix></title>
</titles>
```

```
<h2>Java</h2>
  <p>A Beginner's Guide to Java</p>
  <p>Using XML with Java</p>
<h2>XML</h2>
  <p>Learning XML</p>
  <p>Using XML with Java</p>
```

Solution

```
<xsl:template match="titles">
  <xsl:for-each-group select="title" group-by="ix">
    <h2><xsl:value-of select="current-grouping-key()"/></h2>
    <xsl:for-each select="current-group()">
      <p><xsl:value-of select="."/></p>
    </xsl:for-each>
  </xsl:for-each-group>
</xsl:template>
```

What's New in XSLT 2.0?
User-Defined Functions

- Element **`xsl:function`**
  - Attribute **`name`**
    - Name of function
  - Attribute **`as`**
    - Return value of function
  - Subelement **`xsl:param`**
    - Parameter of function
- Similar mechanism as named templates
- But we can use the functions in XPath expressions

What’s New in XSLT 2.0?

User-Defined Functions

```
<xsl:template match="/">
  <xsl:value-of select="my:factorial($number)"/>
  <xsl:text>&#xA;</xsl:text>
</xsl:template>

<xsl:function name="my:factorial" as="xs:integer">
  <xsl:param name="n" as="xs:integer"/>

  <xsl:choose>
    <xsl:when test="$n > 1">
      <xsl:sequence select="$n * my:factorial($n - 1)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:sequence select="1"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:function>
```

What’s New in XSLT 2.0?
Regular Expressions

- Enable simple processing of text
- XPath 2.0 Functions:
  - **`matches()`** – returns a boolean result that indicates whether or not a string matches a given regular expression
  - **`replace()`** – returns a string obtained by replacing all substrings that match a given regular expression with a replacement string
  - **`tokenize()`** – returns a sequence of strings formed by breaking a supplied input string at any separator that matches a given regular expression

What’s New in XSLT 2.0?

Regular Expressions

- XSLT element **`xsl:analyze-string`**
  - Searches for substrings matching a regular expression
  - Attributes:
    - **`select`** – input string
    - **`regex`** – regular expression
    - **`flags`** – modes: case-insensitive (**`i`**), multi-line (**`m`**), remove-whitespaces (**`x`**), …
  - Subelements:
    - **`xsl:matching-substring`**
    - **`xsl:non-matching-substring`**
  - Functions:
    - **`regex-group()`** – returns N-th captured substring of the regular expression
      - Defined by parentheses (N-th left parenthesis)

What’s New in XSLT 2.0?
Regular Expressions

- Note:
  - XSLT 2.0 enables to read unparsed text
  - XSLT 2.0 enables to start with a named template
    - Can be specified as a parameter of the XSLT parser

$\Rightarrow$ XSLT 2.0 enables to transform non-XML input data into any textual format

- Including XML

Example

```
1164|Steve Jobs|2021
1168|Bill Gates|2021
1564|Mark Zuckerberg|2021
2021|Alan Turing|
```

```
<employees>
  <employee pn="1164">
    <name>Steve Jobs</name>
    <boss>2021</boss>
  </employee>
  <employee pn="1168">
    <name>Bill Gates</name>
    <boss>2021</boss>
  </employee>
  <employee pn="1564">
    <name>Mark Zuckerberg</name>
    <boss>2021</boss>
  </employee>
  <employee pn="2021">
    <name>Alan Turing</name>
  </employee>
</employees>
```

Solution – Part I.

```
<xsl:param name="file">employees.csv</xsl:param>

<!-- read file content into a variable -->
<xsl:variable name="csv"
  select="unparsed-text($file, 'windows-1250')"/>

<xsl:template name="csv2emp">
  <xsl:variable name="rows">
    <xsl:analyze-string select="$csv" regex="^.*$" flags="m">
      <xsl:matching-substring>
        <row>
          <xsl:value-of select="."/>
        </row>
      </xsl:matching-substring>
    </xsl:analyze-string>
  </xsl:variable>


  ...
```

reading
unparsed text

```
   ...

   <employees>
     <xsl:for-each select="$rows/row">
       <xsl:analyze-string
           select="." regex="^(\d+)\|(.+)\|(\d*)\s*$">
         <xsl:matching-substring>
           <employee pn="{regex-group(1)}">
             <name><xsl:value-of select="regex-group(2)"/></name>
             <xsl:if test="normalize-space(regex-group(3)) != ''">
               <boss><xsl:value-of
                   select="normalize-space(regex-group(3))"/></boss>
             </xsl:if>
           </employee>
         </xsl:matching-substring>
         <xsl:non-matching-substring>
           <xsl:message>
             <xsl:text>Error in input data: </xsl:text>
             <xsl:value-of select="."/>
           </xsl:message>
         </xsl:non-matching-substring>
       </xsl:analyze-string>
     </xsl:for-each>
   </employees>
</xsl:template>
```

Solution – Part II.

What's New in XSLT 2.0?

Extension Functions and Instructions

- Extension function = a function that is available for use within an XPath expression
  - Other than XPath core, XSLT extension, etc.
  - We can use functions defined in other languages
  - Function **`function-available()`** enables to test whether a function can be used
- Extension instruction = an instruction (element) which is not from XSLT
  - If a part of an instruction may be unknown, we can define a **`xsl:fallback`**
    - Its body is performed in case of an unknown instruction
- In both cases we must specify a particular namespace

What's New in XSLT 2.0?

Extension Functions and Instructions

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:date="java.util.Date"
                extension-element-prefixes="date">

<xsl:output method="text"/>

<xsl:template match="/">
  <xsl:value-of select="date:to-string(date:now())"/>
</xsl:template>

</xsl:stylesheet>
```

- Maps a Java class to a namespace prefix

What’s New in XSLT 2.0?

Extension Functions and Instructions

```
<xsl:choose>
  <xsl:when test="function-available('my:summary')">
    <xsl:value-of select="my:summary()"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text>Summary not available</xsl:text>
  </xsl:otherwise>
</xsl:choose>
```

```
<xsl:template match="catalog/cd">
   <xsl:loop select="title">
    ...
     <xsl:fallback>
       <xsl:for-each select="title">
         <xsl:value-of select="."/>
       </xsl:for-each>
     </xsl:fallback>
   </xsl:loop>
 </xsl:template>
```

Requirements and Use Cases for New XSLT Version (2.1)

- http://www.w3.org/TR/xslt-21-requirements/
  - Requirements
  - Real-world scenarios
  - Tasks
- Priorities are still being decided
  - XSLT 3.0 is not finished
- Key requirements:
  1. Enabling streamable processing
  2. Enhancement to sorting and grouping
  3. Enhancement to schema awareness
  4. Combining **`group-starting-with`** and **`group-ending-with`**
  5. Default initial template
  6. Setting initial template parameters

XSLT 3.0

- To be used in conjunction with XPath 3.0
- Main extensions:
  1. Streaming mode of transformations
     - Neither the source document nor the result document is ever held in memory in its entirety
     - Motivation: we do not want to load the entire document in memory
  2. Higher order functions
  3. Extended text processing
  4. Improves modularity of large stylesheets
  5. …

http://www.w3.org/TR/xslt-30/

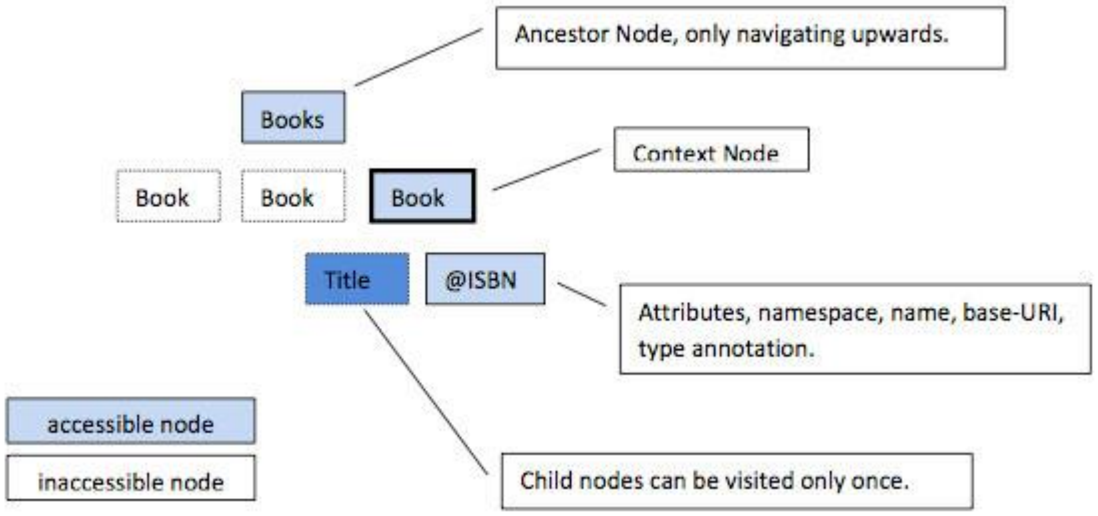
XPath 3.0 Extensions

- W3C Proposed Recommendation 22 October 2013
- Main extensions:
  - Dynamic function call
    - e.g., `$f(2, 3)`, `$f[2]("Hi there")`
  - A mapping operator `'!'`
    - e.g., `child::div1/child::para/string() ! concat("id-", .)` = selects string values of all elements `para` and prepends them with `"id-"`
    - e.g., `$emp ! (@first, @middle, @last)` = selects the three attributes for element in the given variable
- Other emphasized changes:
  - Inline function expressions – anonymous functions
    - e.g., `function($a as xs:double, $b as xs:double) as xs:double { $a * $b }`
  - Support for union types from XML Schema
  - Support for literal URLs in names (instead of prefixes)
  - A string concatenation operator `'||'`

http://www.w3.org/TR/xpath-30/

XSLT 3.0 and Streaming

- Restrictions to be aware of:
  - We have access only to the current element attributes and namespace declaration
  - Sibling nodes and ancestor siblings are <u>not</u> reachable
  - We can visit child nodes <u>only once</u>



Ancestor Node, only navigating upwards.
Books
Book
Book
Book
Context Node
Title
@ISBN
Attributes, namespace, name, base-URI, type annotation.
accessible node
inaccessible node
Child nodes can be visited only once.

"A processor that claims conformance with the streaming option offers a guarantee that an algorithm will be adopted allowing documents to be processed that are orders-of-magnitude larger than the physical memory available."

Example 1. Streamability

- Task: Split the input document so that each chapter is copied to a separate document `outer/chapterN.xml`
  - The input document is too large to fit into memory
  - Each chapter subtree fits into memory

```
<?xml version="1.0"?>
<wrapper>
  <chapter id="1" name="a_chapter_1">
    <p>S the first element of the list.</p>
    <p>Ele.</p>
    <p>He first element of the list, passing the rema.</p>
  </chapter>
  <removed/>
  <chapter id="2" name="a_chapter_2" removed="yes">
  …
```

```
<xsl:stylesheet version="2.0" xmlns:xsl="...">

   <xsl:template match="/wrapper">
    <xsl:for-each select="chapter">
      <xsl:result-document href="chapter{position()}.xml">
        <xsl:value-of select="."/>
      </xsl:result-document>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet version="2.1‟ xmlns:xsl="...">

  <xsl:mode streamable="yes"/>

  <xsl:template match="/wrapper">
    <xsl:for-each select="chapter">
      <xsl:result-document href="chapter{position()}.xml">
        <xsl:-of select="."/>
      </xsl:result-document>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Each template is streamable (conforms to restrictions)

Example 2. Streamability

- Task: The same one, but with nested data (we want the top-level chapters)

```
<?xml version="1.0"?>
<wrapper>
  <chapter id="1" name="chapter_1">
    <p>S the first element of the list.</p>
    <p>Ele.</p>
    <chapter id="2" name="chapter_2">
      <p>Element of the list, pao the syst.</p>
    </chapter>
    <p>He first element of tht, passing the rema.</p>
  </chapter>
  <set>
    <chapter id="3" name="chapter_3">
      <p>A.</p>
      <chapter id="4" name="chapter_4" removed="yes">
        <p>.</p>
        ...
```

```
<xsl:stylesheet version="2.0" xmlns:xsl="…">

  <xsl:template match="/wrapper">
    <xsl:for-each select="//chapter[not(ancestor::chapter)]">
      <xsl:result-document href="chapter{position()}.xml">
        <xsl:copy-of select="."/>
      </xsl:result-document>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

We cannot access ancestors when streaming

```
<xsl:stylesheet version="2.1" xmlns:xsl="…">

  <xsl:mode streamable="yes"/>

  <xsl:template match="/wrapper">
    <xsl:for-each select="outermost(//chapter)"/>
      <xsl:result-document href="chapter{position()}.xml">
        <xsl:copy-of select="."/>
      </xsl:result-document>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

Some functions are specified as streamable

Example 3. Streamability

- Task: Do the inverse of example 1, i.e., join the chapters and create a flat collection

```
<xsl:stylesheet version="2.0" xmlns:xsl="…">

  <xsl:param name="last-doc"/>

  <xsl:template name="main">
    <wrapper>
      <xsl:for-each select="1 to $last-doc">
        <xsl:copy-of select="document(concat('chapter', ., '.xml'))"/>
      </xsl:for-each>
    </wrapper>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet version="2.1"xmlns:xsl="…">

  <xsl:param name="last-doc"/>

  <xsl:template name="main">
    <wrapper>
      <xsl:for-each select="1 to $last-doc">
        <xsl:stream href="{concat('chapter', ., '.xml')}">
          <xsl:copy-of select="."/>
        </xsl:stream>
      </xsl:for-each>
    </wrapper>
  </xsl:template>
</xsl:stylesheet>
```

Processes the content of the document in a streaming manner

Example 4. Streamability

- Task: Given two 1GB documents with flat structure, create a single 2GB file, that contains first all the chapter children from the first file, then all the chapter children from the second file
- Difference from the previous case: the two input documents are too large to fit into memory

```
<xsl:stylesheet version="2.0" xmlns:xsl="…">
  <xsl:param name="doc1"/>
  <xsl:param name="doc2"/>

  <xsl:template name="main">
    <wrapper>
      <xsl:copy-of select="document($doc1)/wrapper/chapter"/>
      <xsl:copy-of select="document($doc2)/wrapper/chapter"/>
    </wrapper>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet version="2.1" xmlns:xsl="…">
  <xsl:mode streamable="yes"/>
  <xsl:param name="doc1"/>
  <xsl:param name="doc2"/>
  <xsl:template name="main">
    <wrapper>
      <xsl:stream href="{$doc1}">
        <xsl:copy-of select="wrapper/chapter"/>
      </xsl:stream>
      <xsl:stream href="{$doc2}">
        <xsl:copy-of select="wrapper/chapter"/>
      </xsl:stream>
    </wrapper>
  </xsl:template>
</xsl:stylesheet>
```

Processes the concatenation in a streaming manner

Processes the content of the document in a streaming manner

XSLT 3.0 and Higher-Order Functions

- Higher order functions = functions that either take functions as parameters or return a function
- XSLT 3.0 introduces the ability to define anonymous functions
  - Enables meta-programming using lambda expressions
- Example:
  - $(x, y) \rightarrow x*x + y*y$ … lambda expression that calculates the square of two numbers and sums them
  - $x \rightarrow (y \rightarrow x*x + y*y)$ … equivalent expression that accepts a single input and as output returns another function that in turn accepts a single input

XSLT 3.0 and Higher-Order Functions

```
<?xml version='1.0'?>
<xsl:stylesheet
     version="3.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($x as xs:integer)
        as (function(xs:integer) as xs:integer) {
        function ($y as xs:integer) as xs:integer {
          $x * $x + $y * $y
          }
      } "/>
    <xsl:value-of select="$f1(2)(3)"/>
  </xsl:template>
</xsl:stylesheet>
```

Variable f1 is assigned to an anonymous function that takes an integer and returns a function that takes an integer and returns an integer

XSLT 3.0 and Higher-Order Functions

- Support for common lambda patterns
  - map – applies the given function to every item from the given sequence, returning the concatenation of the resulting sequences
  - filter – returns items from the given sequence for which the supplied function returns true
  - fold-left – processes the supplied sequence from left to right, applying the supplied function repeatedly to each item, together with an accumulated result value
  - fold-right – respectively
  - map-pairs – applies the given function to successive pairs of items taken one from sequence 1 and one from sequence 2, returning the concatenation of the resulting sequences

http://www.w3.org/TR/xpath-functions-30/

XSLT 3.0 and Higher-Order Functions

Folding that sums only positive numbers from a list

```
<?xml version="1.0"?>
<xsl:stylesheet  version="3.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="list" select="(10,-20,30,-40)"/>

  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($accumulator as item()*, $nextItem as item()) as item()*
        {
        if ($nextItem &gt; 0) then
           $accumulator + $nextItem
        else
           $accumulator
        }"/>
        <xsl:value-of select="fold-left($f1, 0, $list)"/>
     </xsl:template>
</xsl:stylesheet>
```

References

- Jiri Kosek – XSLT 2.0: http://www.kosek.cz/xml/xslt/xslt2.html
- Requirements and Use Cases for XSLT 2.1
  - http://www.w3.org/TR/xslt-21-requirements/
- XSL Transformations (XSLT) Version 3.0
  - http://www.w3.org/TR/xslt-30/
- XPath and XQuery Functions and Operators 3.0
  - http://www.w3.org/TR/xpath-functions-30/
- XSLT and XQuery Serialization 3.0
  - http://www.w3.org/TR/xslt-xquery-serialization-30/