

<title>

</title>

XML

<subtitle>XSLT (cont)</subtitle>

<author>Prof. Dr. Christian Pape</author>

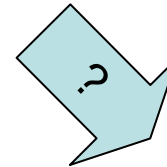
Content

- Expressions and Functions
- Copying nodes
- Using variables and parameters
- Conditional Processing

Expressions

- Arithmetic and Boolean when processing nodes
- Can be used in “value-of”

```
<math>
  <operand>12</operand>
  <operand>23</operand>
  <operand>45</operand>
  <operand>56</operand>
  <operand>75</operand>
</math>
```



```
12 + 25 = 37
23 + 25 = 48
45 + 25 = 70
56 + 25 = 81
75 + 25 = 100
```

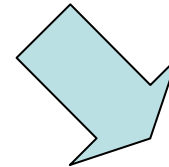
Expressions

- Arithmetic and Boolean when processing nodes
- Can be used in “value-of”

```
<xsl:template match="math">
  <xsl:apply-templates
    select="operand"/>
</xsl:template>
```

```
<xsl:template match="operand">
  <xsl:value-of select="."/>
  <xsl:text>+ 25 = </xsl:text>
  <xsl:value-of select=". + 25"/>
  <xsl:text>&#10;</xsl:text>
</xsl:template>
```

```
<math>
  <operand>12</operand>
  <operand>23</operand>
  <operand>45</operand>
  <operand>56</operand>
  <operand>75</operand>
</math>
```



```
12 + 25 = 37
23 + 25 = 48
45 + 25 = 70
56 + 25 = 81
75 + 25 = 100
```

Expressions

- Presence of operator causes value of select to be evaluated as an expression
- Content of operator (“.”) is converted into a number
- If operand is not a number, then result is NaN (not a number)
- Parenthesis allowed

```
<xsl:template match="operand">
  <xsl:value-of select="." />
  <xsl:text>+ 25 = </xsl:text>
  <xsl:value-of select=". + 25" />
  <xsl:text>&#10;</xsl:text>
</xsl:template>
```

Op.	Description
+	Addition
-	Subtraction
*	Multiplication
div	Division
mod	Remainder of division

Boolean Operators

- Use Boolean expressions in predicates to restrict resulting node list

```
<xsl:apply-templates
  select="operand[. > 40 ]">
</xsl:apply-templates>
```

```
<xsl:template
  match="operand[. > 40]">
  <xsl:value-of select="."/>
  <xsl:text> 25 = </xsl:text>
  <xsl:value-of select=". + 25"/>
  <xsl:text>&lt;br/></xsl:text>
</xsl:template>
```

```
45 + 25 = 70
56 + 25 = 81
75 + 25 = 100
```

Op.	Description
and	Boolean AND
or	Boolean OR
=	Equals
!=	Not equal
<	Less than (< is forbidden in attribute values)
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Boolean Expressions

- No negation operator or Boolean constants true or false
 - Negation is a function “not(boolean)”
 - “true()” and “false()” are functions, too
- Several other functions exist
 - XPath 1.0 defines 27
 - XSLT 1.0 adds 9

Boolean Functions

Functions	Description
boolean(object)	Converts an object to a Boolean value
false()	Returns false
not(boolean)	Returns negation of boolean argument
lang(string)	Returns true, if the argument matches the context-nodes xml:lang value
true()	Returns true

- xml:lang used for multiple-language element
- Inherited by child elements (cordially inherits xml:lang="de")

```
<greet>
  <greeting xml:lang="en">welcome</greeting>
  <greeting xml:lang="de">
    <cordially>Herzlich</cordially> willkommen
  </greeting>
</greet>
```

Boolean Functions

```
<greet>
  <greeting xml:lang="en">welcome</greeting>
  <greeting xml:lang="de">
    <cordially>Herzlich</cordially> willkommen
  </greeting>
</greet>
```

```
<xsl:template match="greet">
  <xsl:apply-templates
    select="greeting[lang('en')]"/>
</xsl:template>
```

```
<xsl:template match="greeting[lang('en')]">
  <xsl:text>English: &#10;</xsl:text>
  <xsl:value-of select="."/>
</xsl:template>
```



English:
welcome

String functions (some)

Function	Description
<code>concat(string, ..., string)</code>	Concatenates two or more string values
<code>contains(string, string)</code>	Returns true if first argument contains second argument
<code>normalize-space(string)</code>	Trims leading and trailing white spaces
<code>starts-with(string, string)</code>	Returns true, if the first arguments starts the second argument
<code>string-length(string)</code>	Returns the length of a string
<code>substring(string, number, number?)</code>	Returns the substring of the first argument starting with the second argument and optional length
<code>substring-after(string, string)</code> <code>substring-before(string, string)</code>	Returns the substring of first argument that follows second argument in first one

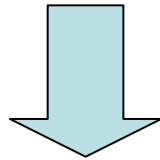
String functions (some)

Example	Result
<code>concat("Good", " ", "day")</code>	"Good day"
<code>contains("Have a nice day", "ave")</code>	true
<code>normalize-space(" eh? ")</code>	"eh?"
<code>starts-with("How about?", "How")</code>	true
<code>string-length("0123")</code>	4
<code>substring("Have a nice day", 5)</code>	"a nice day"
<code>substring("Have a nice day", 5, 1)</code>	"a"
<code>substring-after("Good day", "Good ")</code>	"day"
<code>substring-before("Good day", " ")</code>	"Good"

String Functions

```
<address>
  <name>Christian Pape</name>
  <street>Moltkestrasse</street>
  <house-number>30</house-number>
</address>
```

Source XML format



```
<Address>
  <FirstName>Christian</FirstName>
  <LastName>Pape</LastName>
  <Street>Moltkestrasse 30</Street>
</Address>
```

Target XML format

String functions (black board)

```
<xsl:template match="address">
  <Address>
    <xsl:apply-templates select="name"/>
    <Street>
      <xsl:value-of
        select="concat(street, ' ', house-number)"/>
    </Street>
  </Address>
</xsl:template>

<xsl:template match="name">
  <FirstName>
    <xsl:value-of select="substring-before(., ' ')/>
  </FirstName>
  <LastName>
    <xsl:value-of select="substring-after(., ' ')/>
  </LastName>
</xsl:template>
```

Content

- Expressions and Functions
- **Copying nodes**
- Using variables and parameters
- Conditional Processing

Copying nodes

- **copy (shallow copy)**
 - Outputs current node (element nodes only)
 - No output of attributes, text of current node, or child nodes
 - One optional attribute “use-attribute-set” to refer to a set of attributes
 - Copy may contain a template
- **copy-of (deep copy)**
 - Outputs current node (element and namespace nodes)
 - Copies attributes, text, and child nodes
 - Optional attribute “select” (similar to value-of element)

Copy

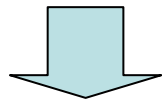
- European union consisting of (founding) member and candidate states

```
<eu> <!-- 2003 -->
  <member>
    <state>Austria</state>
    <state founding="yes">Belgium</state>
    <!-- 12 missing (5 founding states) -->
    <state>United Kingdom</state>
  </member>
  <candidate>
    <state>Bulgaria</state>
    <state>Cyprus</state>
    <!-- 10 missing -->
    <state>Turkey</state>
  </candidate>
</eu>
```

Copy

```
<eu> <!-- 2003 -->
  <member>
    <state>Austria</state>
    <state founding="yes">Belgium</state>
    <!-- 12 missing (5 founding states) -->
    <state>United Kingdom</state>
  </member>
</eu>
```

```
<!-- <xsl:template match="/eu/member/state[2]">
  <xsl:copy/>
</xsl:template>
```



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

No output of attributes, text of current node, or child nodes!

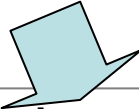
Copy

- Extracting all founding members

```
<xsl:template match="eu/member">
  <eu-members>
    <xsl:apply-templates select="state[@founding]">
  </eu-members>
</xsl:template>
```

```
<xsl:template match="state">
  <xsl:copy>
    <xsl:apply-templates>
  </xsl:copy>
</xsl:template>
```

Build-in rule puts out
elements text
(but not attribute values)

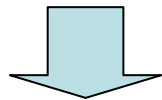


```
<?xml version="1.0" encoding="UTF-8">
<eu-members>
  <state>Belgium</state>
  ...
  <state>The Netherlands</state>
</eu-members>
```

Copy-of

```
<eu> <!-- 2003 -->
  <member>
    <state>Austria</state>
    <state founding="yes">Belgium</state>
    <!-- 12 missing (5 founding states) -->
    <state>United Kingdom</state>
  </member>
</eu>
```

```
<!-- <xsl:template match="/eu/member/state[2]">
  <xsl:copy-of/>
</xsl:template>
```



```
<?xml version="1.0" encoding="UTF-8">
  <state foundation="yes">Belgium</state>
```

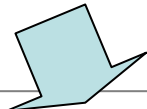
Output of attributes, text, and child nodes.

Copy-of

- Copying all candidate states

```
<xsl:template match="eu">
  <eu><xsl:apply-templates select="candidate"></eu>
</xsl:template>
```

```
<xsl:template match="candidate">
  <xsl:copy>
    <xsl:copy-of select="state"/>
  </xsl:copy>
</xsl:template>
```



```
<?xml version="1.0" encoding="UTF-8">
<eu>
  <candidate>
    <state>Bulgaria</state>
    ...
    <state>Turkey</state>
  </candidate>
</eu>
```

Content

- Expressions and Functions
- Copying nodes
- **Using Variables and Parameters**
- Conditional Processing

Variables and Parameters

- Variable
 - Binds name to a value
 - Value is immutable once bound to name
 - Very restricted form of variables when compared to other programming languages
- Parameter
 - Binds name to a default value
 - Value is mutable, can be passed by command line parameters or when invoking templates
 - Similar to method parameters in other programming languages
- Both
 - Global or local visibility (top-level or within template, definition in template hides top-level definition)
 - Refer to value with name and preceding “\$”, like, \$discount
 - No conflicts with elements of identical name

Variables

- Define variable name with “name” attribute
- Assign value with optional “select”
 - Location paths allowed
 - Expressions allowed
 - Defaults to empty string

```
<xsl:variable name="discount" select="50"/>
```

```
<xsl:variable name="discount" select="0.5 + 0.4"/>
```

```
<xsl:variable name="discount"/> <!-- empty string -->
```

```
<xsl:variable name="discount" select=""/> <!-- same -->
```

```
<xsl:variable name="discount" select="customer/discount"/>
```

```
<xsl:variable name="discount">
```

Variables

- Apply discount on price with XSLT

```
<?xml version="1.0" encoding="UTF-8">
<catalog>
  <item id="sc-00001">
    <maker>Scratchmore</maker>
    <description>Wool sweater</description>
    <size>L</size>
    <price>120.00</price>
    <currency>USD</currency>
  </item>
</catalog>
```

Variables

- Apply 10% discount on product catalog
 - add discount and reduced price

```
<?xml version="1.0" encoding="UTF-8">
<catalog>
  <item id="sc-00001">
    <maker>Scratchmore</maker>
    <description>Wool sweater</description>
    <size>L</size>
    <price>120.00</price>
    <discount>0.10</discount>
    <discount-price>108.00</discount-price>
    <currency>USD</currency>
  </item>
</catalog>
```

Variables (black board)

```
<xsl:variable name="discount" select="0.10"/>
<xsl:template match="catalog">
  <xsl:copy><xsl:apply-templates select="item"/></xsl:copy>
</xsl:template>
```

```
<xsl:template match="item">
  <xsl:copy>
    <xsl:attribute name="id">
      <xsl:value-of select="@id"/>
    </xsl:attribute>
    <xsl:copy-of select="make|description|size|price"/>
    <discount><xsl:value-of select="$discount"></discount>
    <discount-price>
      <xsl:value-of select="price - (price * $discount)"/>
    </discount-price>
    <xsl:copy-of select="currency"/>
  </xsl:copy>
</xsl:template>
```

Parameters

- Definition similar to variables
 - Use param element
- Given value should be considered as a default value
- Global parameters
 - values can be overwritten by command line parameters
- Local parameters
 - Values can be overwritten when invoking template

```
<xsl:param name="discount" select="50"/>
```

```
<xsl:param name="discount" select="0.5 + 0.4"/>
```

```
<xsl:param name="discount"/> <!-- empty string -->
```

```
<xsl:param name="discount" select=""/> <!-- same -->
```

```
<xsl:param name="discount" select="customer/discount"/>
```

```
<xsl:param name="discount">
```

Parameters

```

<xsl:param name="discount" select="0.10"/>
<xsl:template match="catalog">
  <xsl:copy><xsl:apply-templates select="item"/></xsl:copy>
</xsl:template>

```

xalan -p discount '0.20' ...



```

<xsl:template match="catalog">
  <?xml version="1.0" encoding="UTF-8">
  <catalog>
    <item id="SC-00001">
      <maker>Scratchmore</maker>
      <description>wool sweater</description>
      <size>L</size>
      <price>120.00</price>
      <discount>0.20</discount>
      <discount-price>96.00</discount-price>
      <currency>USD</currency>
    </item>
  </catalog>
</xsl:template>

```

Parameters

- Invoking templates with parameters
 - Define parameter in template
 - Use “with-param” in “apply-templates”

```
<xsl:template match="catalog">
  <xsl:copy>
    <xsl:apply-templates select="item">
      <xsl:with-param name="discount" select="'0.15'"/>
    </xsl:apply-templates>
  </xsl:copy>
</xsl:template>
```

```
<xsl:template match="item">
  <xsl:param name="discount"/>
  <xsl:copy>
    <!-- same as before -->
  </xsl:copy>
</xsl:template>
```

Content

- Expressions and Functions
- Copying nodes
- Using Variables and Parameters
- **Conditional Processing**

Conditional Processing

- **if**
 - Inner elements only applied if condition (test) is true
 - no else

```
<xsl:if test="population > 10000000">  
  <xsl:text>(over 10M)</xsl:text>  
</xsl:if>
```
- **choose**
 - Several conditions with “when” elements
 - Inner element of first “when” element with “true” test condition
 - Optional “otherwise” if no “when” is applicable

```
<xsl:choose>  
  <xsl:when test="population > 10000000">  
    <xsl:text>(over 10M)</xsl:text>  
  </xsl:when>  
  <xsl:when test="population &lt;= 10000000">  
    <xsl:text>(&lt;= 10M)</xsl:text>  
  </xsl:when>  
</xsl:choose>
```

Conditional Processing

- Filter all products between and minimum and maximum price (excluding)
 - Two parameters minimum, maximum

```
<?xml version="1.0" encoding="UTF-8">
<catalog>
  <item id="sc-00001">
    ...
    <price>120.0</price>
    ...
  </item>
  <item id="sc-00121">
    ...
  </item>
</catalog>
```

Conditional Processing

```
<xsl:param name="minimum" select="50.0"/>
<xsl:param name="maximum" select="200.0"/>

<xsl:template match="catalog">
  <xsl:copy>
    <xsl:apply-templates select="item"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="item">
  <xsl:variable name="price" select="price"/>
  <xsl:if test="($minimum < $price)
    and ($price < $maximum) ">
    <xsl:copy-of/>
  </xsl:if>
</xsl:template>
```

Conditional Processing

- Given a price threshold t
- Put out all products without original price, but with “< t ” or “ $\geq t$ ” inside price element respectively
- If no price is given put out “no price”
- Threshold given as parameter
- Omit currency

```
<?xml version="1.0" encoding="UTF-8">
<catalog>
  <item id="SC-00001">
    <maker>Scratchmore</maker>
    <description>wool sweater</description>
    <size>L</size>
    <price>120.00</price>
    <currency>USD</currency>
  </item>
</catalog>
```

With $t = 120$

```
<?xml version="1.0" encoding="UTF-8">
<catalog>
  <item id="SC-00001">
    <maker>Scratchmore</maker>
    <description>wool sweater</description>
    <size>L</size>
    <price>< 100</price>
  </item>
</catalog>
```

```
<xsl:param name="threshold" select="90.0"/>

<xsl:template match="item">
  <xsl:variable name="price" select="price"/>
  <price>
    <xsl:copy-of select="make|description|size"/>
    <xsl:choose>
      <xsl:when test="($price &lt;= $threshold)">
        <xsl:text>&lt;; </xsl:text>
        <xsl:value-of select="$threshold"/>
      </xsl:when>
      <xsl:when test="($price &gt; $threshold)">
        <xsl:text>&gt;; </xsl:text>
        <xsl:value-of select="$threshold"/>
      </xsl:when>
    </xsl:choose>
  </price>
</xsl:template>
```

root-element, processing instructions
and template for catalog missing

Rest of lecture

- Java and XML
 - SAX
 - DOM