

XML and Databases

Chapter 15: XSLT

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Winter 2022/23

<http://www.informatik.uni-halle.de/~brass/xml22/>



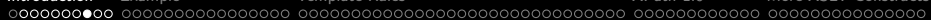
Introduction (2)

- Many browsers support CSS, which is normally used for HTML web pages, also for XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
                  href="mystyle.css"?>
<GRADES-DB>
...

```

- However, this has many restrictions:
 - With CSS, the elements are formatted in the order in which they are written,
 - and there is only very limited filtering.



Introduction (7)

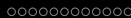
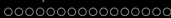
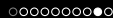
- For translating XML to HTML, XSLT can be used in two places:
 - Client: the web browser does the mapping,
 - Server: one uses an XSLT processor to translate XML to HTML, and publishes the HTML files.

Maybe in addition to the XML files. It is also possible that the HTTP server does the translation on demand: The web browser sends in the HTTP request a list of mime types it understands.

- It seems that browsers today still understand only XSLT 1.0 (which is based on XPath 1.0).

E.g. documentation of XSLT support in Mozilla Firefox:

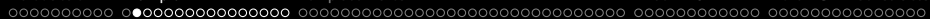
[\[https://developer.mozilla.org/en-US/docs/Web/XSLT\]](https://developer.mozilla.org/en-US/docs/Web/XSLT)



Introduction (8)

- Doing the XML to HTML mapping on Client or Server, continued:
 - If one does the translation in an intranet only for the employees of the company, one can at least rely on the knowledge which browser is used.
 - On the global internet, it might be that potential customers use old browsers which do not support XSLT or support it in incompatible ways.

One can still put the XML file on the server in addition to the HTML file, in order to support semantic web applications (like price comparison services).



Example XML File (1)

STUDENTS

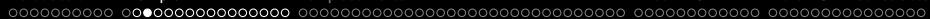
<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	David	Jones	NULL
103	Paul	Miller	...
104	Maria	Brown	...

EXERCISES

<u>CAT</u>	<u>ENO</u>	TOPIC	MAXPT
H	1	ER	10
H	2	SQL	10
M	1	SQL	14

RESULTS

<u>SID</u>	<u>CAT</u>	<u>ENO</u>	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7



Example XML File (2)

- Consider the grades DB with data in attributes:

```
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet type='text/xsl'
                 href='mystyle.xsl'?>
<GRADES-DB>
  <STUDENT SID='101'
           FIRST='Ann' LAST='Smith'
           EMAIL='smith@acm.org' />
  <STUDENT SID='102'
           FIRST='Michael' LAST='Jones' />
  ...

```


Example XSLT Stylesheet (8)

```
<xsl:template match="STUDENT">  
  <li>  
    <xsl:value-of select="@LAST" />,  
    <xsl:value-of select="@FIRST" />  
  </li>  
</xsl:template>  
  
</xsl:stylesheet>
```

- The result of the stylesheet is an HTML page which contains the student names, e.g. "Smith, Ann" as an unordered list.

`value-of` adds the value of the XPath-expression converted to a string.

Example XSLT Stylesheet (9)

```
<STUDENT SID='101'  
  FIRST='Ann' LAST='Smith'  
  EMAIL='smith@acm.org' />
```

```
<xsl:template match="STUDENT">  
  <li><xsl:value-of select="@LAST" /> ,  
    <xsl:value-of select="@FIRST" />  
  </li>  
</xsl:template>
```

```
<li>Smith,  
  Ann  
</li>
```


Stylesheets are XML (3)

```
<!DOCTYPE xsl:stylesheet [  
    <!ENTITY Auml    "Ä">  
    <!ENTITY auml    "ä">  
    <!ENTITY Ouml    "Ö">  
    <!ENTITY ouml    "ö">  
    <!ENTITY Uuml    "Ü">  
    <!ENTITY uuml    "ü">  
    <!ENTITY szlig   "ß">  
    <!ENTITY nbsp    " ">  
]>
```

The numbers can be taken from the HTML DTD or the Unicode standard or [<http://www.w3.org/2003/entities/2007/w3centities-f.ent>].

Templates as Functions (1)

- One can view a template rule as a function with the following input:

- a current node,

The current node is the context node for the evaluation of XPath expressions within the template. It is the main input of the template rule. [<https://www.w3.org/TR/1999/REC-xslt-19991116#rules>]

- a current node list,

The current node list is only used for determining the context position and context size: The current node is always a member of the current node list. Its position is the context position (counted from 1).

The length of the current node list is the context size.

- possibly named parameters (see below).



Templates as Functions (4)

- Initially, `xs:template` is called with the root node of the input document as current node.

I.e. this is the call of the “main” procedure which initiates the stylesheet execution. The “current node list” contains only this node, i.e. the context position is 1 and context size is 1.

- When a template is executed, the contents of the `xs:template` element is evaluated.

This process is described in detail starting on Slide 38. The language can be seen as a functional language, i.e. the contents of the `xs:template` is basically a term which is evaluated to a result tree fragment based on the values of its subterms. There are variables (see Slide 73), but these can be assigned a value only once, i.e. they are constants.

Templates as Functions (7)

- Since the current node (the main input parameter to the template invocation) is so important, there is a special function `current()` which returns it.

There are several “additional functions” that can be used in XPath expressions embedded in XSLT, which are not part of the XPath core function library. [<http://.../REC-xslt-19991116#add-func>] [<http://.../REC-xslt-19991116#function-current>]

- When an XPath expression in the template is evaluated, the context node is first the current node.
- However, as the path expression navigates through the XML document, the context node changes, while the result of `current()` remains stable.

Templates as Functions (8)

- Note that a template invocation for a node n can call templates for any node in the input document, not only in the subtree rooted at n .
- Print sum of homework points for each student:

```

<xsl:template match="STUDENT">
  <li>
    <xsl:value-of select="@LAST" />:
    <xsl:value-of select="sum(//RESULT
      [@SID=current()/@SID] [@CAT='H']
      /@POINTS)" />
  </li>
</xsl:template>

```

Template Instantiation (1)

General Remarks:

- The contents of the `xs:template`-element is “instantiated” to give a result tree fragment (which usually becomes part of the output document).

When the “real” XSLT evaluation starts, there is an XDM tree for the input document, and one for the stylesheet (which is XML, too).

- I.e. to specify what XSLT does, one must define a function “instantiate” that takes a context C and a node n inside the template as input, and returns a sequence of nodes for the result tree.

Usually, it will call itself recursively for the children of node n .

Template Instantiation (2)

Literal Result Elements:

- Elements within the template that do not belong to the XSLT namespace are evaluated by creating the corresponding element in the output.

[<https://www.w3.org/TR/1999/REC-xslt-19991116#literal-result-element>]

- The content of the element in the template is evaluated to give the content of the constructed element.

Therefore, e.g. nested `xsl:value-of` or `xsl:apply-templates` are evaluated, too.

Template Instantiation (3)

Literal Result Elements, continued:

- Attributes of the element are treated as “attribute value templates”: They can contain XPath-expressions in “{...}” that are evaluated to give the attribute value of the constructed element.

This is similar to XQuery. If one needs literal “{” outside an XPath expression, one has to write “{{”, and the same for “}”. Note that this special interpretation of “{...}” happens only in attribute values of literal result elements or certain attributes of some XSLT elements. Within element content, “{...}” is not interpreted.

[<https://.../REC-xslt-19991116#dt-attribute-value-template>]

Template Instantiation (4)

Literal Text:

- Text nodes within the template are copied to the generated output, unless they contain only whitespace.

And unless an ancestor element of the text node contains the special attribute `xml:space="preserve"` (introduced in the XML standard).

[<https://.../REC-xslt-19991116#section-Creating-Text>]

[<https://.../REC-xslt-19991116#strip>]

- Note that adjacent text nodes (no matter how they are created) are merged, and empty text nodes are removed, as required by the data model.

Template Instantiation (5)

xsl:text:

- If one wants to generate e.g. a single space in the output document, one can write

```
<xsl:text> </xsl:text>
```

Whitespace within this element is preserved.

- The element `xsl:text` can be used for generating any text node, but because literal text is copied, this is needed only in special situations.
- The content model of `xsl:text` is `#PCDATA`.
I.e. pure text without nested elements.

Template Instantiation (6)

disable-output-escaping:

- If one writes `<` or `<`, the internal representation of the style sheet contains the character “<”.
- Normally, on output this is escaped, i.e. written “`<`”, so that valid XML is produced.

The same is done for “&” and maybe other special characters.

- The elements `xsl:text` and `xsl:value-of` have an attribute `disable-output-escaping` which permits to suppress this translation.

[<https://.../REC-xslt-19991116#disable-output-escaping>]

Template Instantiation (7)

disable-output-escaping, continued:

- The default value of this attribute is "no".
- If `disable-output-escaping` is set to "yes", characters like `<` are printed without the translation to `<`.
- Note that in the stylesheet, one must write `<` (so that the stylesheet is valid XML).
- But the output will then be simply `<`.
- E.g., if one wants to generate `ä`:

```
<xsl:text disable-output-escaping="yes"
  >&amp;auml;</xsl:text>
```

Template Instantiation (8)

value-of:

- `<xsl:value-of select="e"/>` is replaced by the value of the XPath-expression `e`, converted to string.

The node, for which the template is applied, is the context node for evaluating `e`. The result of `value-of` is always a string (text node).

[<https://www.w3.org/TR/1999/REC-xslt-19991116#value-of>]

- If several nodes are selected, only the first (in document order) is chosen, and its string-value is taken.

As explained above for `xsl:text`, the attribute `disable-output-escaping` can be used to suppress the translation of characters like `<` to `<`.

Template Instantiation (9)

apply-templates:

- `<xsl:apply-templates select="e"/>` is replaced by the result of doing the “template rule” transformation recursively for all nodes that in the result of the XPath-expression `e`. See Slide 34.

- If `select` is missing, all child nodes are selected.

`xsl:apply-templates` can contain `xsl:sort` →81 and `xsl:with-param` →84. Furthermore, it has an attribute `mode` →83.

[<https://.../REC-xslt-19991116#section-Applying-Template-Rules>]

- Of course, a template can contain any number of calls to `xsl:apply-templates`, not only one.

Template Instantiation (10)

xsl:copy:

- `<xsl:copy>C</xsl:copy>` copies the current node (the node for which the template was called) and replaces its children by the result of evaluating `C`.

I.e. the node is copied, but not its children or attributes. The copied node can be any kind, it does not have to be an element node.

[<https://www.w3.org/TR/1999/REC-xslt-19991116#copying>]

- The identity transformation can be specified as:

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>
```

Template Instantiation (11)

xsl:copy-of:

- `<xsl:copy-of select="e"/>` copies the result of evaluating `e` into the result of the template, including all descendant nodes.

[<https://www.w3.org/TR/1999/REC-xslt-19991116#copy-of>]

- I.e. `xsl:copy-of` can be used to copy portions of the input XDM tree into the output.

The input XDM tree is subject to the removal of pure whitespace text nodes, as defined by `<xsl:strip-space elements="A B C"/>` (whitespace text nodes that are children of the named elements A, B and C will be removed), see [<https://www.w3.org/TR/1999/REC-xslt-19991116#strip>].

Template Instantiation (12)

xsl:element:

- `<xsl:element name="n">C</xsl:element>`
generates an element node with name *n* and content that is the result of evaluating *C*.

[\[http://.../REC-xslt-19991116#section-Creating-Elements-with-xsl:element\]](http://.../REC-xslt-19991116#section-Creating-Elements-with-xsl:element)
- One can use `{e}` in the attribute value of `name` to include XPath-expressions that are evaluated and replaced by the result (converted to a string).

I.e. the attribute value is interpreted as “attribute value template”. This possibility to compute the element name is probably the reason for using `xsl:element`, because otherwise one could have written simply the element itself.

Template Instantiation (13)

xsl:attribute:

- `<xsl:attribute name="n">C</xsl:attribute>`
generates an attribute node with name *n* and the result of evaluating *C* as value.

One can use {*e*} in the attribute value of *name* (see `xsl:element` above).
The result of evaluating *C* must be a text node (e.g., element nodes cannot become an attribute value).

[<https://www.w3.org/TR/1999/REC-xslt-19991116#creating-attributes>]

- The generated attribute node is assigned to the enclosing element.

Attribute nodes must come first in the content, they cannot be added, e.g., after a text node.

Template Instantiation (14)

Summary:

- There are constructor elements for each node type:

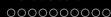
- `xsl:element`
- `xsl:attribute`
- `xsl:text`
- `xsl:processing-instruction`

[<http://.../REC-xslt-19991116#section-Creating-Processing-Instructions>]

- `xsl:comment`

[<https://.../REC-xslt-19991116#section-Creating-Comments>]

Obviously, these are similar to the computed constructors in XQuery. Literal elements, attributes and text in the template are copied to the result, this corresponds to the direct element constructors of XQuery.



Template Rule Selection (1)

- A template with XPath-expression p in the attribute “**match**” is applicable to a node n
 - if there is some ancestor a of n
 - such that n is an element of the nodes selected by p evaluated with context node a .
- For instance, “**/GRADES-DB/STUDENT**” matches
 - a **STUDENT**-node within the top **GRADES-DB** node,
 - not a **GRADES-DB** node with a **STUDENT** child node.
- There are priority rules if several templates match.

See below and

[<https://www.w3.org/TR/1999/REC-xslt-19991116#conflict>].

Template Rule Selection (2)

- Only a subset of XPath 1.0 is allowed as pattern, basically a |-union of path expressions using

- `/`, `//`,

At the beginning for absolute paths and as separators of the steps.

- `child::`, `attribute::`

Of course, the child axis does not have to be specified explicitly, and the attribute axis can be abbreviated to “@”.

- node tests (name and type tests) and
- predicates `[...]`.

Inside predicates, the full XPath 1.0 can be used.

[\[https://www.w3.org/TR/1999/REC-xslt-19991116#patterns\]](https://www.w3.org/TR/1999/REC-xslt-19991116#patterns)

Template Rule Selection (3)

- Each template has a priority.
- One can specify a priority explicitly:

```
<xsl:template match="STUDENT[1]" priority="2.0">
```

- If one does not specify a priority, the default is:
 - **-0.5** if it is just a node test without a name, e.g. `*` or `@*` or `text()`.
 - **-0.25** if it is a wildcard with a namespace, e.g. `abc:*`
 - **0** if it is a name, e.g. `STUDENT` or `@EMAIL`.
 - **+0.5** for all more complex patterns.

Template Rule Selection (4)

- If a pattern is composed with `|`, then the template is treated like several templates, one for each alternative.
- E.g. if the pattern is “`STUDENT|EXERCISE[1]`”, the template rule has priority
 - `0`, if applied to a `STUDENT`-element, and
 - `0.5`, if applied to the first `EXERCISE`-child of its parent.

Template Rule Selection (5)

- A stylesheet can be composed out of several files:

```
<xsl:import href="lib.xsl">
```

- Template rules from files that are imported earlier have lower “import precedence” than templates from later imported files.

Template rules in the main file have the highest “import precedence”.

`[xsl:apply-imports]` can be used for applying overridden rules.

- For template rule selection, “import precedence” is considered first, and priority second.

It is an error if that still leaves more than one template rule. However, the XSLT processor may recover by selecting the later rule.

Built-In Template Rules (1)

- Built-in template rule for descending in the tree if there is no other match:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

When `apply-templates` is specified without the attribute `select`, it processes all child nodes of the current node (element nodes, text nodes, comment nodes and PI nodes).

- Built-in rules have the lowest possible import precedence. Thus, if there is another rule matching the node, this other rule is chosen.

Built-In Template Rules (2)

- The following rule prints text nodes and values of attribute nodes when/if they are selected:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

Note that the above default rule applies the templates only to child nodes, not the attribute nodes.

- There is also a rule for processing instructions and comment nodes that returns an empty node set:

```
<xsl:template
  match="processing-instruction()|comment()"/>
```


Restrictions in XPath 1.0 (2)

- XPath 1.0 is not based on sequences as XPath 2.0. The available data types are:

- string
- number (floating point)

There is no integer type in XPath 1.0. Note that the type “number” contains an error value (NaN: “not a number”), positive and negative infinity, and a positive and a negative zero.

- boolean
- node set

Instead of sequences, XPath 1.0 has only node sets. Often the nodes must be ordered, this is done in document order. Some rules in XPath 2.0 become clearer if one remembers that the designers tried to remain compatible with the node sets from XPath 1.0.

Restrictions in XPath 1.0 (3)

- Variables can be defined only outside the path expression (in XSLT, not in XPath). Therefore, there is no **for**, **some**, **every**, **:=**.
- There is also no **if**.
- **except** und **intersect** are not available in XPath 1.0.
 - **union** is also not available, but **|** is (which does \cup).
- Since the type system is much more restricted (not based on XML Schema), there are no type tests or type conversions.

E.g. no instance of, **treat** as castable, **cast**.

Operators in XPath 1.0

Prio	Operator	Assoc.
1	or	left
2	and	left
3	=, !=	left
4	<, <=, >, >=	left
5	+, -	left
6	*, div, mod	left
7	- (unary)	right
8		left
9	/, //	left
10	[]	left

Functions in XPath 1.0 (2)

Node Set Functions, continued:

- `string local-name(node-set? x)`:

Node name without namespace prefix.

The first node in the input node set is taken (in document order). The argument is optional. If the function is called without argument, the local name of the context node is returned.

- `string namespace-uri(node-set? x)`:

Namespace URI of (first) argument node.

- `string name(node-set? x)`:

Name of (first) node in x with namespace prefix.

Functions in XPath 1.0 (3)

String Functions:

- `string string(object? x)`: Conversion to string.

For a node set, the string-value of the first node in the set is taken. If the argument is omitted, the context node is taken.

- `string concat(string s1, string s2, string* sn)`:
String concatenation.
- `boolean starts-with(string x, string y)`:
y is prefix of *x*.
- `boolean contains(string x, string y)`:
y is substring of *x*.

Functions in XPath 1.0 (4)

String Functions, continued:

- `string substring-before(string x, string y)`:
Prefix of x until first occurrence of y .

The empty string is returned if x does not contain y .

E.g. `substring-before("abcbc", "b") = "a"`.

- `string substring-after(string x, string y)`:
Suffix of x after first occurrence of y .

E.g. `substring-after("abcbc", "b") = "cbc"`.

- `string substring(string s, number p, number? l)`
Substring of s starting at position p with length l .

E.g. `substring("abcde", 2, 3) = "bcd"`. If l is omitted: entire rest.

Functions in XPath 1.0 (5)

String Functions, continued:

- **number** `string-length`(**string?** *s*):
Number of characters in *s*.
- **string** `normalize-space`(**string?** *s*):
s with sequences of whitespace characters reduced to a single space, and leading/trailing whitespace removed.
- **string** `translate`(**string** *s*, **string** *x*, **string:***y*)
s with the *i*-th character in *x* replaced by the *i*-th character in *y* (or removed if `string-length(y) < i`).

Functions in XPath 1.0 (6)

Boolean Functions:

- `boolean boolean(object x)`: Conversion to boolean.

See “effective boolean value”: E.g. node set is true iff it is not empty.

- `boolean not(boolean b)`: Negation.

- `boolean true()`: Constant value “true”.

- `boolean false()`: Constant value “false”.

- `boolean lang(string /)`:

Language of the context node is *l*.

E.g. if `xml:lang = "en-us"` is specified in an ancestor node of the context node (and no other language is specified in between), `lang("en")` is true.

If no language is specified, `lang` returns false.

Functions in XPath 1.0 (7)

Number Functions:

- `number number(object? x)`: Conversion to a number.
If a string has no numeric format, it is converted to the special floating point value “NaN” (“not a number”, error value).
- `number sum(node-set x)`:
Sum of the result of converting the string-value of each node in x to a number.
- `number floor(number x)`: Largest integer $\leq x$.
- `number ceiling(number x)`: Smallest integer $\geq x$.
- `number round(number x)`:
 x rounded to nearest integer (.5 is rounded up).

Variables (2)

- It is also possible to define the variable value in the content. This is processed as a template:

```
<xsl:variable name="table_headline">
  <tr><th>Student</th><th>Points</th></tr>
</xsl:variable>
```

- One can use this result tree fragment as follows:

```
<xsl:copy-of select="$table_headline"/>
```

- Note: `<xsl:variable name="n" select="2">` is not the same as `<xsl:variable name="n">2</xsl:variable>`.

In the first case, the value is the number 2, in the second case a node with value 2. E.g. `//student[$n]` works in the first case, but in the second, one must write `//student[position()=$n]`.

Variables (4)

- A defined variable cannot be assigned a new value (i.e. it should be understood like a constant).

A global variable can be shadowed by a local variable in a template.

- This permits very different evaluation algorithms, e.g. on parallel hardware.

The language is declarative (functional), not imperative.

- Of course, a variable within a template gets a new value for each template invocation:

```
<xsl:template match="STUDENT">  
  <xsl:variable name="no" select="position()"/>  
  ...  
</xsl:template>
```



Repetition: for-each (1)

- With the **for-each** construct, one can embed a template directly into another template:

```
<xsl:template match="GRADES-DB">
  <ul>
    <xsl:for-each select="STUDENT">
      <li>
        <xsl:value-of select="@LAST"/>,
        <xsl:value-of select="@FIRST"/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

[<https://www.w3.org/TR/1999/REC-xslt-19991116#for-each>]



Sorting (1)

- One can specify that `apply-templates` and `for-each` should construct the result not in document order of the selected nodes, but in a specific sort order.
- E.g. print students alphabetically sorted by last name, and by first name if last names are equal:

```
<xsl:template match="GRADES-DB">
  <ul>
    <apply-templates select="STUDENT">
      <sort select="@LAST"/>
      <sort select="@FIRST"/>
    </apply-templates>
  </ul>
</xsl:template>
```




References

- James Clark (Editor): XSL Transformations (XSLT), Version 1.0 W3C Recommendation, 16 November 1999
[\[https://www.w3.org/TR/1999/REC-xslt-19991116\]](https://www.w3.org/TR/1999/REC-xslt-19991116)
- Michael Kay (Editor): XSL Transformations (XSLT), Version 2.0 W3C Recommendation, 23 January 2007
[\[http://www.w3.org/TR/xslt20/\]](http://www.w3.org/TR/xslt20/)
- Michael Kay (Editor): XSL Transformations (XSLT), Version 3.0 W3C Candidate Recommendation, 19 November 2015
[\[http://www.w3.org/TR/xslt-30/\]](http://www.w3.org/TR/xslt-30/)
- Michael Kay: XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer) Wiley, 4th Ed. (June 3, 2008), ISBN-10: 0470192747, 1376 pages.
- Wikipedia (English): XSLT
[\[https://en.wikipedia.org/wiki/XSLT\]](https://en.wikipedia.org/wiki/XSLT)
- Robert Tolksdorf: Vorlesung XML-Technologien (Web Data and Interoperability), Kapitel 6: XSLT: Transformation von XML-Dokumenten. Freie Universität Berlin, AG Netzbasierte Informationssysteme, 2015.
[\[http://blog.ag-nbi.de/wp-content/uploads/2015/05/06_XSLT.pdf\]](http://blog.ag-nbi.de/wp-content/uploads/2015/05/06_XSLT.pdf)
- w3schools: XSLT Element Reference.
[\[http://www.w3schools.com/xml/xsl_elementref.asp\]](http://www.w3schools.com/xml/xsl_elementref.asp)