ZENTRUM FÜR
INFORMATIONSMODELLIERUNG
AUSTRIAN CENTRE FOR
DIGITAL HUMANITIES

KARL-FRANZENS-UNIVERSITÄT GRAZ
UNIVERSITY OF GRAZ

Beyond TEI

Digital Editions with XPath & XSLT for the Web & in LaTeX

Sarah Lang

Harvard, April/May 2022

Overview

1. The workshop
2. Navigating XML using XPath

The workshop

Goals

1. get to know XPath & XSLT (and learn how to use it)
2. understand the role of XML/TEI, XPath and XSLT in Digital Editing
3. be able to use XSLT to generate HTML and LaTeX output from TEI
4. Two days isn't enough for you to master XSLT!

Schedule

Day 1, morning	XML, TEI and Digital Editing → repetition of the basics, making sure we're all on the same page, understanding why we're even learning XSLT.
Day 1, afternoon	Navigating XML documents using XPath, introduction to HTML (& Bootstrap) and LaTeX (& `reledmac`)
Day 2	Transforming XML documents into HTML & LaTeX output formats using XSLT

Single point of entry for all workshop-related materials: LaTeX Ninja blogpost & Github Repository (‘additional resources’ directory)

Introductions

Please introduce yourselves!

1. Name, pronouns, field/topic of study
2. Why did you come to this workshop?
3. Previous experience with Digital Humanities (DH) or editing?

Contact

@SarahALang_
@latex_ninja

sarahalang.com
latex-ninja.com

sarah.lang@uni-graz.at

Sarah Lang (she/they)

- originally from Germany, now in Graz (Austria)
- Studied Latin, French & History (teacher's education) in Graz & Montpellier (France), then Archaeology Bachelor, Master in Religious Studies & Philosophy
- got a DH certificate & started working at Zentrum für Informationsmodellierung (ZIM) / Centre for Information Modelling in Graz
  - Moral Weeklies/Spectators → *gams.uni-graz.at/mws*
  - Graz Repository of Ancient Fables (GRaF) → *gams.uni-graz.at/graf*
  - *PhD thesis:* Decoding alchemical *Decknamen* digitally. A Polysemantic Annotation and Machine Reasoning Algorithm for the Corpus of Iatrochymist Michael Maier (1568–1622)
- *Now:* teaching in Graz, Passau & Vienna; PostDoc in Graz. *Research interests:* history of science (alchemy), Neo-Latin, text mining and computer vision

Navigating XML using XPath

Slowly moving beyond TEI…

Why are we doing this workshop? The motivation from our abstract:

- ✔ But what happens after an edition is encoded in TEI?
- 👍 While it is an **ideal format for archiving digital data**, it is less than ideal for viewing and interacting with the edited text.
- 👍 The data transformation language XSLT allows editors to create multiple representations from their data encoded in XML, enabling the creation of both digital and print editions.

Goals for the next session

- ✔ understand the terms from the abstract
- ✔ everybody on the same page on XML/TEI for digital editing
- ✔ creating websites in HTML (& Bootstrap)
- ✔ creating PDFs & print(able) editions using LaTeX (& `reledmac`)
- ✖ ~~creating different representations from our data~~ → first: navigating XML documents automatically using XPath

Introduction to XPath i

XPath can be used to navigate in XML documents.
//persName = 'Give me all personal names / persons in the whole document.' → all findings will be highlighted in Oxygen. I have thus 'found' or 'addressed' them. I can 'select' them, for example to process them in XSLT.

Every occurence of Mina would result in the following once 'translated' to XPath:

1. All *<persName>* which have *@ref='#Mina'*, *in the whole document*
2. all *<persName>* in the whole document: *//persName*.
3. … having *@ref='Mina'*:

```
//persName[@ref='#Mina']
```

4. Try it out! (Use the XPath field in Oxygen – if you don't have any, look in the settings/preferences: Window → Tools: check 'XPath'.)

Introduction to XPath ii

How to find something using XPath:

1. Formulate a normal English sentence expressing what you wish to achieve, e.g. ‘I want all occurrences of the person Mina.’
2. ‘Translate’ this to the technical structure of the document: All those occurrences are encoded as `persName ref="#Mina"` – this gives me the means to address them.
3. Do I want to get the element nodes themselves (for further processing) or just their text contents? Or the result of a function? (‘Count all occurrences and give me their number.’)?
4. Are there conditions? Do I want just some specific elements (a subset of the above) or everything that is found (‘matched’) by my request?

paths & scope i

Depending on the query, I can search *in the whole document* or *in a certain scope*. To do that, I need to distinguish:

relative paths

Not every step in traversing the tree hierarchy of XML is listed but the query is run depending on the cursor's or programme's current location. Thus I need to know my current position.
This is useful because I only need to know about what's below that point in the tree (a limited scope). It's bad if I don't know where I am and get other results than expected. If you're not sure, prefer absolute paths!
In XSLT, we will need relative paths.

absolute paths

Every step in the hierarchy is stated explicitly. I only get the desired path (not accidentally matching something else).

```
//titleStmt/title
```

→ returns only the document title from the *<teiHeader>*, not other titles from bibliographical references.
Gets more complicated if the result comes from different places (use 'or' statements here: ||).

paths & scope ii

scope

It's possible that I continue working with a subset of all nodes in XSLT, for example following a `<xsl:for-each>` loop. Here I have to use relative paths because I can't know at the time of writing the code where such elements will be found.

A relative path can lead you to find unwanted things: maybe you get more or less than expected. Maybe you're not in the right place and the desired query doesn't yield any results in the current scope.

So how does XPath work? i

query result = set of nodes

A query returns a set of nodes. If you put a dot (`.`), you get just the text content of the current node.

`matches(.,'([0-9]{4})')` Does 'everything' (current node text content) contain a four letter number (i.e. data)? Answer: yes/no. → no direct access to the contents found, just a yes/no answer is the result!

`//*[matches(.,'([0-9]{4})')]` = 'Give me any node in the whole document which contains (nothing but) a four digit numer/date' = returns a set of nodes to which the answer to the question in the brackets (XPath condition) is 'yes' = 'true'.

Difference between strings & nodes

Once XPath is in the substring, it only has characters (strings) left to access, not nodes/elements. Same is true for dot (`.`). Some functions return an answer, some make changes (like *`translate()`*). You might get the error *`XPath failed due to: A sequence of more than one item is not allowed as the first argument of`*. This is because some functions only allow one element or one char-litteral as input, example:

```
contains(//t:origPlace/@ref, 'pleiades')
```

Tests results in error if there is more than one *`<origPlace>`* element in a document because many string operations only allow a char-literal or one element, not a set of nodes as input. The error can appear even when there actually is only one element where the node test is true (only one *`@ref='pleiades'`*). Solution:

```
//t:origPlace/@ref[contains(., 'pleiades')]
```

Also, you can only 'click into' (jump to) the results if it's a set of notes, not a string or yes/no answer.

So how does XPath work? ii

predicates [] = conditions

Predicates are conditions and noted [in brackets]. You can query the result of a function there (yes/no) and thus, make queries like: Give me all personal names if/where XY is true.

functions

Functions can go into predicates but also around the whole expression → in the latter case, you usually get an 'answer', not a set of nodes (which you probably wanted). Can result in the error explained on this slide. (Kind of) Like a function in math, you get just value as an answer – not a set of results.

```
function(//location, parameter)
```

Unlike in normal search an replace, we can now query conditions: "Give me all occurrences of the person Mina but only in chapters where Dracula doesn't appear."

So how does XPath work? iii

Absolute and relative paths

Try playing around with absolute and relative paths: To use a relative path, click into the *<titleStmt>* and search for just:

```
title
```

Try all these options. Are there differences?

```
title
/TEI/teiHeader//title
//titleStmt/title
//title
```

The last option will give you all titles, so if there's more than just the document title in the header, you will get all of them. *//* always give you the global scope.

```
//person//surname
//persName[@ref]
//persName/@ref
//persName[@ref="bla"]
//persName[contains(.,'bla')]
contains(.,'bla') = yes/no, no nodes
```

Negating queries

Negate the query with *!=* OR *not*. Example: A *<persName>* which doesn't have an *@ref* attribute:

```
//persName[@ref]
//persName[not(@ref)]
```

All persons except Mina:

```
//persname[@ref !='Mina']
```

Examples to get started

See exercise sheet.

```
1. //div[@type="chapter"]   -- count(//head)
2. //head -- //div[@type='chapter']/head
3. substring-after(//div[type=....],'Chapter')
4. distinct-values(//placeName/@xml:id)
  OR distinct-values(//placeName)
```

→ What's the difference?

```
5. distinct-values(//persName[@ref="#Mina"]
7. //p[contains(.,'Christmas')]
  OR //p[matches(.,[A-Z][a-z])]
8. //div[@type="chapter"][@n="4"]/head
  OR  //div[@type="chapter"][4]/head
9. //persName[@ref != "#Mina"]
  OR \\persname[not(@ref='#Mina')]
10. //div[@type='chapter'][1]/persName | //div[@type='chapter'][1]/placeName
OR //person | //place
```

XPath i

XML document = **tree structure** → navigate via paths

- root node (/) ≠ root element = child of the root node.
- Additionally: attribute node, text node, element node, (+ namespace + comment + processing instructions)
- Query = path, result = node set / value.
- part of the XSL language family (XPath, XSLT, XSL-FO, ...XQuery).

absolute (*/person/name*) OR
relative (*../../title*)

Syntax

- *axis name::nodetest[predicate]*
- axis = direction of movement
- node test: which node?
- predicate: condition.

XPath ii

/child::person[attribute::gender]/child::name ..... long
/person[@gender]/name ..... shorthand

axes

::self ..... current node itself (shorthand: .)
::ancestor ..... all ancestors of current node
attribute:: ..... attributes of current node (**shorthand: @**)
child:: ..... direct children of current node
descendant:: ..... ancestors of current node (**shorthand: //**)
parent:: ..... parent of current node (**shorthand: ../**)
preceding-sibling:: ..... preceding siblings of current node
following-sibling:: ..... following siblings of current node

Navigating XML

XPath expressions are evaluated in their context ● absolute & relative paths are possible (*dynamic & static context* ● namespaces ● functions, variables

XPATH basics

`/` ..... root node / one step further down the tree

`::*` ..... ::* = all types of elements irrespective of name

`@` ..... attribute

`[number]` ..... rank in result set

`text()` ..... text content of element

`::elementtype` ..... condition for element type (node test)

`//` ..... at any depth

`[text() = 'text content']` ..... condition (predicate)

`.` ..... self

`(/../` ..... jump one hierarchical level regardless of name

`path//subpath` ..... arbitrary depth

XPath predicates

Conditions which define subsets = predicates

```
//person[@gender="female"]
//person[firstname = "Stefan"]
//person[firstname != "Tanja"]
//person[1]
//person[last()]/firstname
//person[position() = 2]
/participants/person[position()=5]/firstname
/participants/person[last()]/firstname
count(child::*)
```

XPath functions

functions – set of nodes

`position()` ..... position of current node, return: number

`count()` ..... number of nodes in given node set

`last()` ..... last node in selected node set

string functions

`substring-before(value, substring)` .... substring of text node (*value*) coming before the indicated *substring*

`substring-before(., 'mina')` ..... e.g.

`substring-after(value, substring)` ..... like before

`substring(value, start, length)` ..... with start pos & length

`string-length(.)` ..... length of current text node

`concat(value1, value2, ...)` ..... concatenate

`concat(surname, ', ', forename)` ..... e.g.

`translate(target, string, replacement)` ..... replace

`noramlize-space(target)` ..... removes trailing whitespace

XPath functions

Boolean functions

`starts-with(value, substring)` ..... return: true/false

`starts-with(., 'D')` ..... true/false

`contains(value, substring)` ... if the current text node contains the string, returns yes

`not(comparison)` ..... returns yes if not XY

Functions with regex support

`matches(target, 'RegEx')` .. like `contains()` but only exact matches

`replace()` ..... like `translate()` but supports regex

`tokenize()` ..... tokenize text into single words

Troubleshooting i

1. **[Nothing works]** In case of doubt it's always the namespace! Or you're somewhere other than you expected (using relative paths) → try an absolute path!
2. **[Nothing found]** XML is **hierarchical**! Am I in the right spot? Can I reach the degree of **depth** that I wanted? In doubt, include `/../` to be more independent of hierarchy.
3. **[Not found enough?] Jumps in path depth:** A path (without `path//subpath`) in between will not jump any levels, only go one step deeper! Are there in-between layers you forgot? Like another nested `<div>`? Try putting `//` but then, depth is completely arbitrary (maybe also not the wanted outcome) – if you want a particular depth, like 3, try `/../../../`.
4. **[Path works but doesn't do what I wanted]** Formulate your query as precise as possible. Do I find all I want? Does my query match more than I wanted?
5. XSL's standard processing behaviour causes text content to be printed as is unless it's explicitly changed or suppressed.

Troubleshooting ii

Preparing XSL transformations

First query all elements used in your file (once each using the `distinct-values()` function) to be sure that you have everything covered or learn what you need to write template rules for.

`distinct-values(//@*/name())` ..... What types of attributes are there?
`distinct-values(//@*)` ..... What attribute values are in my document?
`distinct-values(//@rend)` ..... What values does `@rend` take?
`distinct-values(//name())` ..... What elements are there?

XPath practice!

Start working on the XPath/XSLT exercise sheet (resources/materials folder); XPath 1.1 & 1.2. It has lots of explanations. Feel free to use the slides and cheatsheet for help. Read the info on the worksheet!