XSLT for Beginners

Jennifer Ramirez (ZB MED) | ramirez@zbmed.de

Declarative Amsterdam
November 12, 2026

About Me!

Jennifer Ramirez-Betancur

- BSc & MSc in Health Informatics
- Now: XML/XSL in Scientific Publishing
- International work:
  - BC CDC (Canada)
  - AKUH (Kenya)
  - cityEHR (Ukraine & UK)
- Teaching & Hobbies



How Today Runs

- One project: the conference programme, as XML
- Each section: a problem, a short explanation, a demo, then you write it
- Stuck? - there's people here who can help
- Questions: ask! - engagement welcome
- Break at 75 minutes

Quick Check

- Who has written XSLT before?
- Who works with XML regularly?

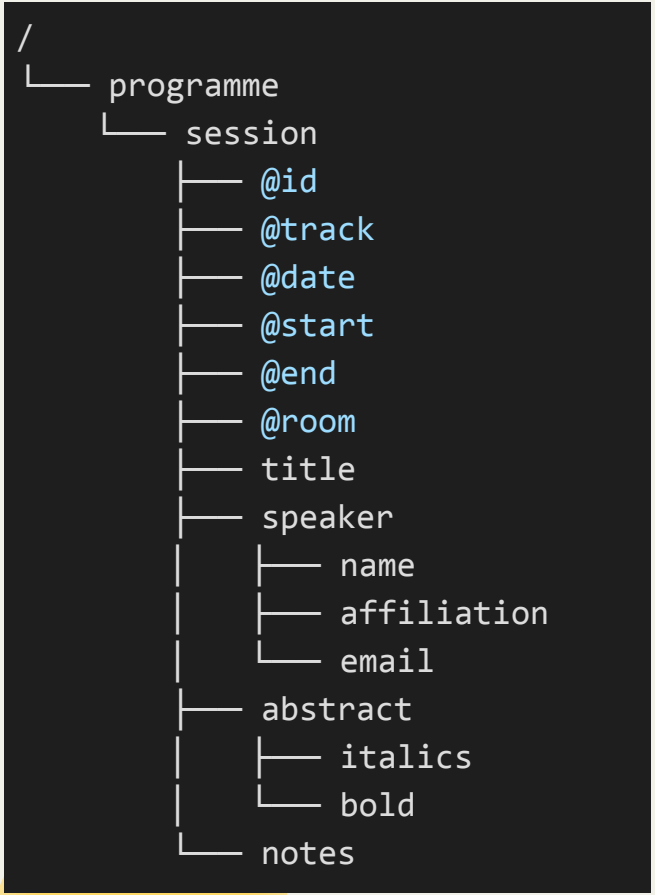
1.2 Programme File, XPath Basics & XML Refresher

The Conference Programme as XML

```
 <session id="s07" track="tutorial" date="2026-11-12" start="13:45" end="14:45"
room="L016">
    <title>CEM-ML declarative document lifecycle engine</title>
    <speaker>
      <name>Sasha Firsov</name>
      <email>sasha@example.org</email>
    </speaker>
    <abstract>What if the languages and tools in a <italics>web application</italics>
were not a tower of fixed…</abstract>
    <notes>Speaker needs a third screen.</notes>
  </session>
```

The Programme as a Tree

- Part of the processing builds a tree
- An XML tree contains Element nodes, Attribute nodes, Text nodes, etc.
- Documents are processed Top-down, Left-to-right



What is XML?

- XML marks up text so a program can tell what each part is: this is a title, this is a speaker's name
- You choose the element names. XML only sets the rules for writing them
    - Or you use a vocabulary someone else has defined, like JATS for journal articles or TEI for texts in the humanities
- Every XML document is a tree: one root element, with everything else nested inside it

Well-formed XML

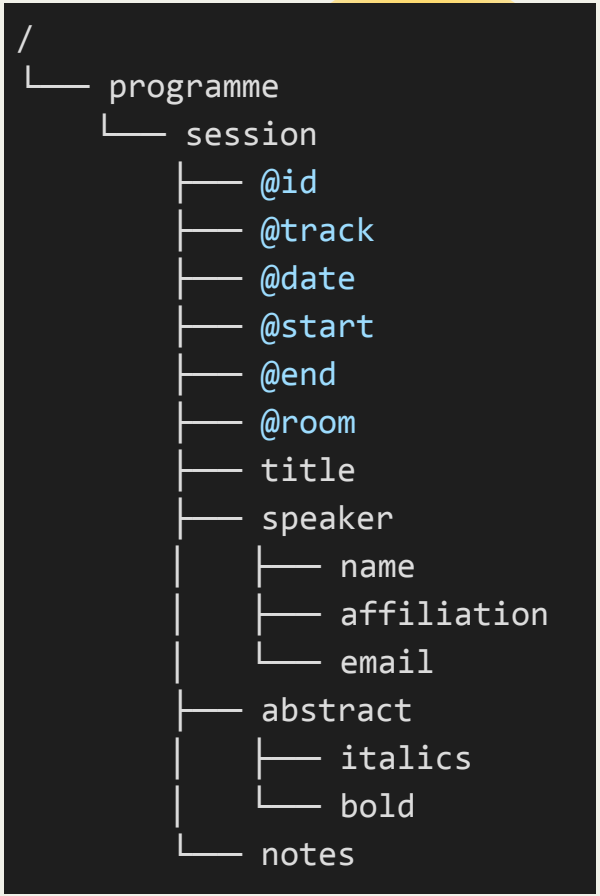
The Syntax Rules

- A single root element
- Elements must be properly nested & have a closing tag
- Tags are case-sensitive
- & written &, < written <
- Quoted attribute values

```
<programme>
   <session id="s02" track="symposium" date="2026-11-13" start="10:00"
end="10:30" room="Turing Hall">
     <title>Schemas & Schematron</title>
     <speaker>
       <name>Jane Doe</name>
       <email>jane@example.org</email>
     </speaker>
     <abstract>This talk is <bold><italics>very</bold></italics>
practical.</abstract>
   </session>
   <session id="s03" track="break" date="2026-11-13" start="11:00 end="11:30"
room="Foyer">
     <title>Coffee break</title>
   </session>
 </programme>
```

XPath

Path	What it selects
/programme/session	All the session elements that are children of programme, the root element
title	The title children of the current node (the context node). Inside a session template, that's the session's title
speaker/name	The name elements inside the speaker elements that are children of the current node
@start	The start attribute of the current node
//title	Every title in the document, at any depth



XPath: Predicates & Axes

```
<programme>
     <session id="s12" room="Turing Hall">
       <title>Saxon-Forms</title>
       <speaker>
         <name>John Dziurłaj</name>
         <affiliation>Hilton Roscoe
                      LLC</affiliation>
       </speaker>
       <speaker>
         <name>O'Neil Delpratt</name>
         <affiliation>Saxonica</affiliation>
       </speaker>
     </session>
</programme>
```

Path	Result
/programme/session/speaker[2]/name	<name>O'Neil Delpratt</name>
/programme/session[@room = 'Turing Hall']/title	<title>Saxon-Forms</title>
//speaker[affiliation = 'Saxonica']/name	?
//session[@room = 'L016']/title	?
//affiliation/ancestor::session/@id	?

[ ] is a predicate: a filter. Read it as "where".
:: names an axis: the direction to move. No axis means child.

Task: XPath in Oxygen

Open programme.xml

In Oxygen's XPath bar, try:

```
1.  //title
2.  //session[@room = 'L016']/title
3.  //session[@id = 's22']/speaker[2]/name
4.  //speaker[affiliation]/name
```

What are the results?

1.3 XSLT Basics

What is XSLT?

***E**xtensible **S**tylesheet **L**anguage **T**ransformations*

- Transform from:
  - XML format into another
  - XML format to non-XML format
  - non-XML format to an XML format
- **Today**: 3.0

Stylesheets

- <xsl:stylesheet> is the root element for all XSLT documents
- Namespace must be *"http://www.w3.org/1999/XSL/Transform"*
- Version 3.0 is best

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

  <xsl:template match="session">
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```

The Processor & How Processing Works

1. Start at the document node
2. Find the template that best matches
3. No match? A built-in rule applies
4. `xsl:apply-templates` moves on to the nodes it selects

Your First Stylesheet!

1. Open exercises/1.3-first.xsl and run its scenario
2. Add the first-session line from TODO 2, A and then B, and run each
3. Change [1] to [2] (or [18]), then try one of the paths from 1.2

1.4 Identity Transform, apply-templates, & Built-in Templates

The Identity Transform

- It copies everything: every attribute and every node!
- On its own that isn't very useful. It's a starting point: add a template for each part you want to change, and the rest passes through untouched
- This is one of the most used patterns in XSLT, for anything that changes a little and keeps the rest

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

xsl:mode

- XSLT 3.0's one-line version of the identity template
- A node with no template of its own is copied, and processing carries on inside it
- `xsl:mode` has more settings, such as `warning-on-no-match="yes"`, which asks the processor to warn when a node has no template of its own

```
<xsl:mode on-no-match="shallow-copy"/>
```

xsl:apply-templates

- Output for one speaker (with no email)

```
<speaker><name>Steven Pemberton</name><affiliation>CWI</affiliation></speaker>
```

- apply-templates says "process these next". Anything it doesn't select is never processed, so nothing comes out for it
- select picks which nodes. Leave it out and you get all the children (not the attributes)

```
<xsl:template match="speaker">
  <xsl:copy>
    <xsl:apply-templates select="name | affiliation"/>
  </xsl:copy>
</xsl:template>
```

Empty Templates

The other way to leave something out: let it be selected, and give it a template that outputs nothing

Email is more specific than `node()`, so this template wins over the identity transform template

```
<xsl:template match="email"/>
```

Built-in Templates

- Elements: process the children
- Text: copy it to the output
- Attributes: not visited unless you ask
- Comments and processing instructions: dropped

Task: A Public Copy

1. Open exercises/1.4-public.xsl. It's only a skeleton
2. Use the identity template to copy the file
3. Add an empty template for email | notes
4. Replace the identity template with xsl:mode

1.5 Literal elements, xsl:output, & The Context Node

Literal Result Elements

Elements without `xsl:` in front are literal result elements: they go to the output as they are

The `xsl:` instructions fill in the data, *e.g. the sessions*

The stylesheet is still XML, so write `<br/>`, not `<br>`. HTML output turns it back into `<br>`

```
<xsl:template match="/programme">
  <html>
    <head>
      <title>Declarative Amsterdam 2026</title>
    </head>
    <body>
      <h1>Programme</h1>
      <xsl:apply-templates select="session"/>
    </body>
  </html>
</xsl:template>
```

xsl:output

The xsl:output declaration controls how the result of an XSLT transformation is serialized.

Options for output formats are defined using xsl:output , including the target format type (@method) whether mark-up is indented, which encoding to use, and much more.

```
<!-- HTML output -->
<xsl:output method="html" html-version="5"
            indent="yes"
            encoding="UTF-8"/>

<!-- XML output -->
<xsl:output method="xml" indent="yes"
            encoding="UTF-8"
            omit-xml-declaration="yes"/>

<!-- Plain text output -->
<xsl:output method="text" encoding="UTF-8"
            media-type="text/plain"/>

<!-- JSON output -->
<xsl:output method="json" indent="yes"
            encoding="UTF-8"/>

<!-- Adaptive output (XSLT 3.0) -->
<xsl:output method="adaptive"
            indent="yes"/>
```

The Context Node

Inside this template, title means this session's title

{title} writes out its value as text. The long form is <xsl:value-of select="title" />

```
<xsl:template match="session">
  <div>
    <h2>{title}</h2>
    <xsl:apply-templates select="speaker"/>
  </div>
</xsl:template>
```

Task: The Web Page

1. Open exercises/1.5-html.xsl
2. Add `@date`, `@start`, `@end`, and, `@room` to each session. Try at least A and C from the examples and compare
3. Make the speaker template output the name in a `<p class="speaker">` .

1.6 Push and Pull

Pull

The program "pulls" the information it needs to process a document as it is needed. The most common paradigm that programmers are used to using.

```
<p class="abstract">{abstract}</p>

<p class="abstract"><xsl:value-of select="abstract"/></p>
```

Push

This is where the document to be processed is "pushed" through the program, so that the program processes information in the document as it is encountered.

```
<xsl:apply-templates select="abstract"/>

<xsl:template match="abstract">
  <p class="abstract"><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="bold">
  <b><xsl:apply-templates/></b>
</xsl:template>
```

xsl:for-each

- The pull loop: good for regular, predictable lists
- Inside it, the context node is the current session

```
<ul class="overview">
  <xsl:for-each select="session">
    <li>{@start}: {title}</li>
  </xsl:for-each>
</ul>
```

Task: Abstracts & an Overview List

1. Open exercises/1.6-push.xsl
2. Replace the {abstract} line with apply-templates and an abstract template
3. Using the bold template as a model, write a new template for italics
4. Add an overview list under the h1 with xsl:for-each : start time and title



1.7 Conditions: Declarative vs Procedural

Conditions: xsl:if & the XPath if Expression

xsl:if : output this block, or nothing

XPath if … then … else : pick one of two values

```
<xsl:template match="affiliation"> ({.})</xsl:template>

<xsl:if test="speaker">
  <p class="label">{if (count(speaker) eq 1) then 'Speaker:'
                   else 'Speakers:'}</p>
</xsl:if>
```

Procedural & Declarative

```
<xsl:for-each select="node()">
  <xsl:choose>
    <xsl:when test="self::bold"><b><xsl:value-of select="."/></b></xsl:when>
    <xsl:when test="self::italics"><i><xsl:value-of select="."/></i></xsl:when>
    <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

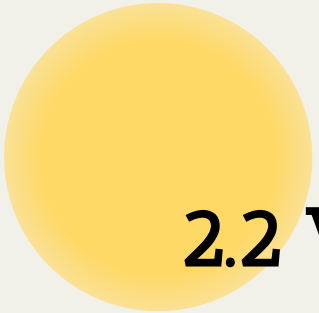
```
<xsl:template match="bold"><b><xsl:apply-templates/></b></xsl:template>
<xsl:template match="italics"><i><xsl:apply-templates/></i></xsl:template>
```

Task: Affiliations & Speaker Labels

1. Open exercises/1.7-conditions.xsl
2. Add a new affiliation template, applied after the speaker's name
3. Add a label "Speaker:" or "Speakers:" above the speakers, but only for sessions that have any

Checkpoint

- What a stylesheet is, and how the processor runs it
- XPath: finding things in the tree
- The identity transform and apply-templates: a public copy of the programme
- Built-in templates and xsl:mode
- Templates, the context node, value-of: a web page
- Push and pull
- Conditions, and when templates do the job better



2.2 Variables

xsl:variable

Used to store information during processing. Unlike a variable in a procedural programming language, it cannot change once set.

Declared with a name, used with a `$` in front of the name.

A good name makes the code say what it means: `$speaker-count` reads better than the expression behind it

```
<xsl:variable name="speaker-count"
              select="count(distinct-values(//speaker/name))"/>

<p>{$speaker-count} speakers</p>
```

Task: Count the Sessions

1. Open exercises/2.2-variable.xsl
2. In the `/programme` template, declare `$sessions`
3. Show the number of sessions under the `h1`
4. Use `$sessions` in the overview list and in the session `apply-templates` too
5. Add `$talks:` select only the tutorials and symposium talks. Write the predicate yourself!

2.3 Parameters

xsl:param

Allows parameters to be passed to stylesheets - once defined, it can be used just like a variable. A value the stylesheet gets from whoever runs it: the Oxygen scenario, the command line, or another program

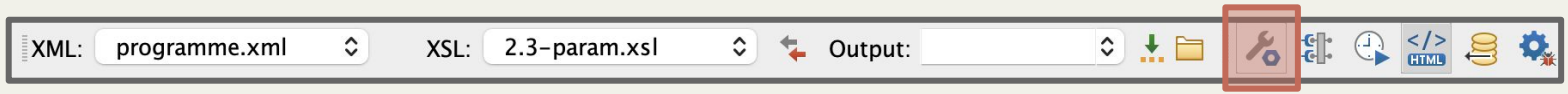
The select is only a default, used when no value is passed.

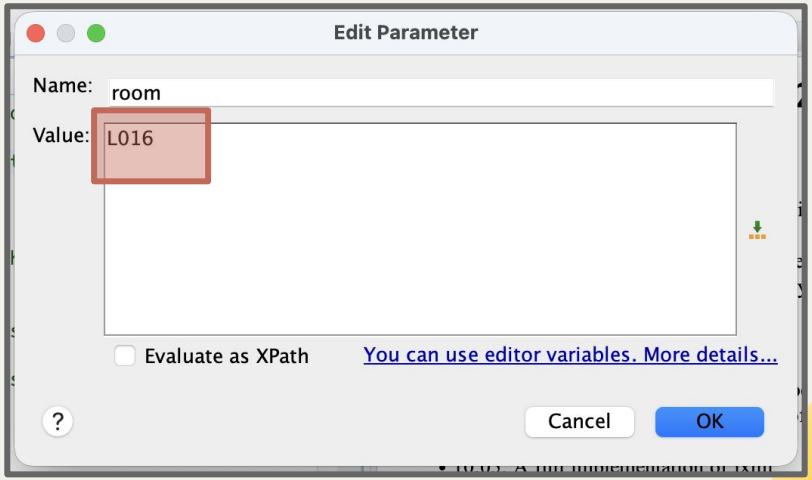
The quotes inside the quotes: `'all'` is a string

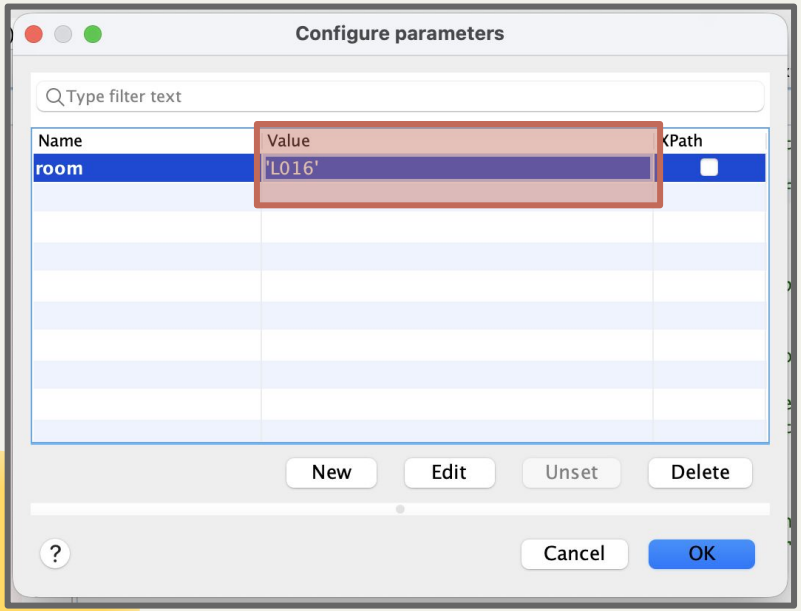
```
<xsl:param name="room" select="'all'"/>

<xsl:variable name="sessions"
    select="session[$room = 'all' or @room = $room]"/>
```

Setting a Parameter in Oxygen







Conditions on a Parameter

In 1.7, decisions about what a node is went into templates. This decision is about a setting from outside, and there's no node to match, so a condition can be the right tool.

The XPath **if** picks one of two values. **xsl:if** outputs a block or nothing. **||** joins strings

```
<h1>{if ($room = 'all') then 'Full programme'
                       else 'Programme for ' || $room}</h1>

<xsl:if test="$room = 'all'">
  <ul>...overview list...</ul>
</xsl:if>
```

Task: Room Printout

1. Open exercises/2.3-param.xsl
2. Declare the room param, default 'all'
3. Filter $sessions by room. Run it: nothing changes yet
4. Set room in the scenario's Parameters. Run it for L016, then Turing Hall
5. With an XPath if (or xsl:choose): "Full programme" or "Programme for [room]"
6. With xsl:if: show the overview list only on the full programme

2.4 Named Templates

Named Templates

A named template has a name instead of a match. It only runs when you call it via xsl:call-template

Calling it doesn't change the context node, so @start is still the start of the session you're in.

Its parameters come first, each with a default.

```
<xsl:template name="time-range">
  <xsl:param name="from" select="@start"/>
  <xsl:param name="to" select="@end"/>
  <xsl:text>{$from}-{$to}</xsl:text>
</xsl:template>

<p>{@date}, <xsl:call-template name="time-range"/>, at {@room}</p>
```

xsl:param & xsl:with-param

xsl:param is on the receiving side: it says what the template accepts, and what to use when nothing is sent

xsl:with-param is on the calling side: it sends a value. The two names have to match

Leave out xsl:with-param and the default is used

```
<!-- the call sends a value -->
<xsl:call-template name="time-range">
  <xsl:with-param name="from"
                  select="'08:30'"/>
</xsl:call-template>

<!-- the template receives it -->
<xsl:template name="time-range">
  <xsl:param name="from" select="@start"/>
  <xsl:param name="to" select="@end"/>
  <xsl:text>{$from}-{$to}</xsl:text>
</xsl:template>
```

Template Parameters with apply-templates

xsl:with-param works on apply-templates too.

The template that receives it declares it with xsl:param , like a named template.

```
<xsl:apply-templates select="$sessions">
  <xsl:with-param name="show-abstract" select="$room = 'all'"/>
</xsl:apply-templates>

<xsl:template match="session">
  <xsl:param name="show-abstract" select="true()"/>
  ...
  <xsl:if test="$show-abstract">
    <xsl:apply-templates select="abstract"/>
  </xsl:if>
</xsl:template>
```

Named Templates, Modes, & Functions

How you call it	Use it for
`<xsl:call-template name="time-range" />`	A piece of output you want in several places
`<xsl:apply-templates select="." mode="time-range" />`	The same nodes, processed a second way
`{my:time-range(.)}`	A value you calculate inside XPath

Task: time-range Template

1. Open exercises/2.4-named.xsl
2. Create the time-range template with from and to params
3. Call it in the session block
4. Call it in the overview list

2.5 Sorting

xsl:sort

Allows you to sort the data before transforming it.

It must be the first child of either:

- xsl:apply-templates
- xsl:for-each
- xsl:perform-sort

```
<xsl:apply-templates select="$sessions">
  <xsl:sort select="@date"/>
  <xsl:sort select="@start"/>
</xsl:apply-templates>
```

Task: Sort the Programme

1. Open exercises/2.5-sort.xsl
2. Sort the session blocks by date, then start time
3. Try sorting by title, then by speaker: `speaker[1]/name`
4. Try `order="descending"`
5. Sort the overview list the same way

2.6 Outputs (again..)

Text output

- Text between instructions that's only whitespace is ignored
- xsl:text keeps it.
 is a line break

```
<xsl:output method="text"/>

<xsl:template match="session">
  <xsl:value-of select="@start"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="title"/>
  <xsl:text>&#10;</xsl:text>
</xsl:template>
```

Task: Plain-text Programme

1. Open exercises/2.6-text.xsl
2. Output the start time and the title. Where did the space go?
3. Fix it with `xsl:text`, and end each line with `
`
4. Write the whole line as one `xsl:text` with `{…}`

JSON output

XSLT 3.0 can write JSON in two ways: build XML in a fixed format and convert it with `xml-to-json()`, or build Maps and Arrays and let `method="json"` write them out

You can read JSON too: `parse-json()` and `json-doc()` give you Maps, and `json-to-xml()` gives you XML (you can apply templates to)

```
<!-- 1. XML in a fixed format, then xml-to-json() -->
<map xmlns="http://www.w3.org/2005/xpath-functions">
  <string key="title"><xsl:value-of select="title"/></string>
</map>
```

```
<!-- 2. a map, to be written out by method="json" -->
map { 'title' : string(title) }
```

2.7 Wrap-up

What We Covered (a lot)

What you used	What it made
XPath, templates, apply-templates, built-in rules	A public copy of the programme
Literal elements, push and pull	The web page, with abstracts and an overview
Templates instead of tests, conditions	Affiliations and speaker labels
Variables and parameters	Counts, and a printout for each room door
Named templates	One time range in two places
Sorting, text and JSON output	Time order, the email and the app

Tools for Afterwards

- **Editors:** Oxygen XML Editor, Visual Studio Code (with an XSLT extension), Altova XMLSpy, Stylus Studio, Eclipse
- **Processors:** XSLT 3.0, 2.0, and 1.0: Saxon (Java/JavaScript/C/Python)
  XSLT 1.0: Xalan (Java), xsltproc (C)
- **Command line with Saxon for Java:** java net.sf.saxon.Transform -s:source -xsl:stylesheet -o:output
- **Command line with SaxonJS on Node:** npm install xslt3, then xslt3 -s:source -xsl:stylesheet -o:output
- Your Oxygen licence from today works until **19 November**

Thank you!

Do you have any questions?

Declarative

Amsterdam
Jennifer Ramirez (ZB MED) | ramirez@zbmed.de

November 12, 2026