Week 02: XML transformations, XPath

Agenda

- Quick note on XML namespaces
- XPath
- XML transformations
  - XSLT
- Short demo
- Hands on: Iteration 02

Let's start!

XML Namespace

- logical spaces
- element name can be **reused** in different namespaces
- namespace elements **can have** a specific prefix
  - *no prefix* => element belongs to the default namespace
  - *explicit prefix* => element is from a different namespace, prefix overrides the default namespace
- namespace has a unique **name** defined by URI
- URI is seen **only as** a string => it is just a simple identifier
- URI often points at a namespace documentation => just a convention, not a rule
- **default** namespace vs. **prefixed** namespace

```
<!-- Default namespace -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <body>
     <h1>Heading</h1>
  </body>
</html>

<!-- Prefixed namespace -->
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <xhtml:body>
     <xhtml:h1>Heading</xhtml:h1>
  </xhtml:body>
</xhtml:html>
```

XPath

- query language, path expressions
- **selecting** nodes from document
- capable of computing values from the content
- based on the **tree representation** => walks the tree, selects nodes that satisfy given criteria
- can be used in **XSLT**
- doesn't follow **XML** syntax

Note: XPath syntax is similar to how we write paths in file systems.

Expression	Description
nodename	Selects all nodes with the name "nodename"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

Example: XPath

```xml
<bookstore>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

  <book category="web" cover="paperback">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

```xpath
<!-- select every `title` element from a `book` element
     from `bookstore` where `price` is less than 35.00 -->
/bookstore/book[price<35.00]/title

<!-- select the first `book` element from `bookstore` -->
/bookstore/book[1]

<!-- select every `title` element from the current node
    where `title` has an attribute `lang` and
    its value is equal to 'en' -->
//title[@lang='en']

<!-- select every `book` element
     that contains an `author` child element -->
//book[./author]
```

XSLT

- **XSL** transformations
- **XSL** => e**X**tensible **S**tylesheet **L**anguage
  - describes how elements are supposed to be transformed
- for instance, from XML to HTML, XML to a different XML
- conversions to Word, CI output conversions...
- **.xslt/.xsl** file extension

Example: XSLT for bookstore XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/bookstore">
    <library>
        <books>
            <xsl:apply-templates select="book"/>
        </books>
        <authors>
            <xsl:for-each select="book">
                <author>
                    <xsl:value-of select="author" />
                </author>
            </xsl:for-each>
        </authors>
    </library>
  </xsl:template>

  <xsl:template match="book">
    <book lang="{title/@lang}">
      <xsl:if test="price > 35">
        <xsl:attribute name="deposit">true</xsl:attribute>
      </xsl:if>
      <xsl:value-of select="title" />
    </book>
  </xsl:template>

</xsl:stylesheet>
```

Example: Resulting XML

```
<library>
   <books>
      <book lang="en">Harry Potter</book>
      <book lang="en" deposit="true">Learning XML</book>
   </books>
   <authors>
      <author>J K. Rowling</author>
      <author>Erik T. Ray</author>
   </authors>
</library>
```

XSL Templates

- template can **reuse** another template inside itself
- **xsl:import** or **xsl:include** to load a template (from a different file)
- **xsl:apply-templates** or **xsl:call-template** to use a template

Note: break a large template into smaller reusable ones, if possible => readability and maintainability are increased.

XSL Loops and conditions

- **xsl:for-each** => *for* loop
- **xsl:if** => *if* statement
- **xsl:choose**, **xsl:when**, **xsl:otherwise** => condition with more options

XSL Choose example

```
<xsl:variable name="standardPrice" select="20.99"/>
<xsl:variable name="actualPrice" select="@price" />

<xsl:choose>
  <xsl:when test="$standardPrice > $actualPrice">
    <xsl:attribute name="discount">true</xsl:attribute>
  </xsl:when>
  <xsl:otherwise>
    <xsl:attribute name="specialEdition">true</xsl:attribute>
  </xsl:otherwise>
</xsl:choose>
```

Functional vs. procedural approach?

- XSLT is based on functional programming ideas
- **xsl:for-each**?
  - does not behave exactly as in imperative lanugages
  - can be completely avoided

Functional vs. procedural apporach: foreach

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- library.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" indent="yes" />
    <xsl:template match="/">
        <library>
            <xsl:for-each select="bookstore/book">
                <book>
                    <xsl:value-of select="title" /> by <xsl:value-of select="author" />
                </book>
            </xsl:for-each>
        </library>
    </xsl:template>

</xsl:stylesheet>
```

Functional vs. procedural apporach: foreach with template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- library.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" indent="yes" />
    <xsl:template match="/">
        <library>
            <xsl:for-each select="bookstore">
                <xsl:apply-templates select="book" />
            </xsl:for-each>
        </library>
    </xsl:template>

    <xsl:template match="book">
        <book>
            <xsl:value-of select="title" /> by <xsl:value-of select="author" />
        </book>
    </xsl:template>

</xsl:stylesheet>
```

Functional vs. procedural apporach: only template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- library.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" indent="yes" />
    <xsl:template match="/">
        <library>
            <xsl:apply-templates select="bookstore/book" />
        </library>
    </xsl:template>

    <xsl:template match="book">
        <book>
            <xsl:value-of select="title" /> by <xsl:value-of select="author" />
        </book>
    </xsl:template>

</xsl:stylesheet>
```

Functional vs. procedural apporach: two files

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- library.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" indent="yes" />
    <xsl:include href="book.xsl" />

    <xsl:template match="/">
        <library>
            <xsl:apply-templates select="bookstore/book" />
        </library>
    </xsl:template>

</xsl:stylesheet>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- book.xsl -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

    <xsl:output method="xml" indent="yes" />
    <xsl:template match="book">
        <book>
            <xsl:value-of select="title" /> by <xsl:value-of select="author" />
        </book>
    </xsl:template>

</xsl:stylesheet>
```

Resulting XML

```
<library>
    <book>Harry Potter by J K. Rowling</book>
    <book>Learning XML by Erik T. Ray</book>
</library>
```

Note: without foreach, we could break the template into separate files and still achieve the same result. With an appropriate XPath expression and template application, foreach is not necessary.

Tools

- XPath online: freeFormatter XPath
- XSLT online: freeFormatter XSLT
- **xsltproc** in command line

Note: if you'll use xsltproc in command line, the command is as follows:

```
xsltproc -o output.html app.xsl data.xml
```

where names and types of files are chosen as necessary.

Note: Mac OS and some Linux distributions have xsltproc tool by default, for others, see options in Package manager or use Choco/WSL (for Windows).

Demo: Discord

Disclaimer: we will use a little bit of HTML, but in a very beginner friendly way. The only thing you need to know for now about HTML is that it has a very similar syntax to XML.

- take XML data from the previous week (but this time with some real image URLs)
- create XSL stylesheets to define how the data show up in the HTML
- take a look at the resulting HTML in the browser

Questions?

Hands on: Iteration 02

You can find the assignment in Gitlab Issues as well as in the Interactive syllabus.

Before you start:

- please check if your tutor has already accepted your MR
- if they have, make sure you merged your solution from the previous week

Note: if your tutor haven't got to your MR yet, it's completely ok. You don't have to have the previous iteration merged to be able to work on a new one - iterations are independent. However, if you have an accepted MR that is not merged, it's still open and not incorporated in your 'main' branch.