

Einführung in XSLT mit Übungen

Wuppertal, 11.2.2020

Übersicht

- Was ist XSLT?
- Grundsätzliche Funktionsweise
- Aufbau einer Transformation
- Wichtige Befehle Elemente
- XSLT in oXygen
- Übungen

Was ist XSLT?

- XSLT ist eine Sprache, um XML-Dokumente in verschiedene Zielformate zu transformieren
- XSLT ist ein W3C-Standard seit 1999, aktuell: XSLT 3.0
- *XSLT heißt Extensible Stylesheet Language Transformation*
- XSLT ist selbst XML
- $XSL = XSLT + XSL-FO$

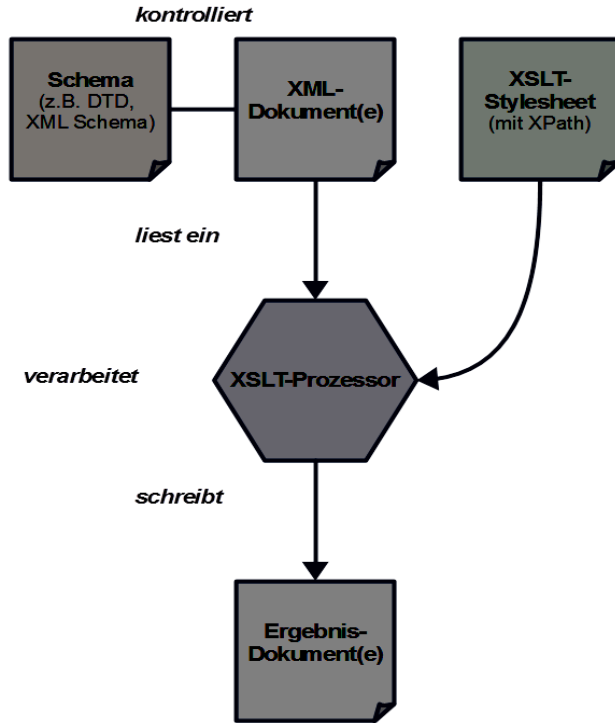
Wozu braucht man XSLT

Man braucht XSLT, um aus XML etwas (anderes) zu machen

- Reports
 - Z.B. Text
- Publikationen
 - Z.B. HTML/CSS
- Visualisierungen
 - Z.B. SVG, HTML/CSS/Javascript
- Exporte
 - Z.B. CSV, JSON, Datenbankformate
- XML
 - Daten verändern, anreichern, zusammenführen, extrahieren

... dies alles kann zusammenspielen!

Grundsätzliche Funktionsweise



- XSLT verwendet XPath
- XSLT wird von einem XSLT-Prozessor ausgeführt
- XML + XSLT → Zieldokument(e)

Grundsätzliche Funktionsweise

- Das Prinzip der drei Bäume:
 - Eingabebaum – Verarbeitungsbaum – Ausgabebaum
- Das Prinzip der Verarbeitung
 - Gehe durch den Eingabebaum und berücksichtige dabei die Anweisungen des Verarbeitungsbaums um den Ausgabebaum zu schreiben
- Das Prinzip der „Schablonen“
 - Wenn diese Schablone (ein XPath-Muster) passt, dann folge ihren Anweisungen

Grundsätzliche Funktionsweise

Deutsch → XSLT

Ich bin ein XML-Dokument

Ich bin ein XSLT-Stylesheet

Ich bin eine Schablone, ich passe auf ein Muster

Ich tue etwas, wenn das Muster im Ausgangsbaum gefunden wird. Meistens hole ich Informationen aus dem XML-Dokument, verarbeite sie und schreibe sie in das Zieldokument

Ich bin eine Schablone, ich passe auf ein Muster

Grundsätzliche Funktionsweise

Deutsch → XSLT

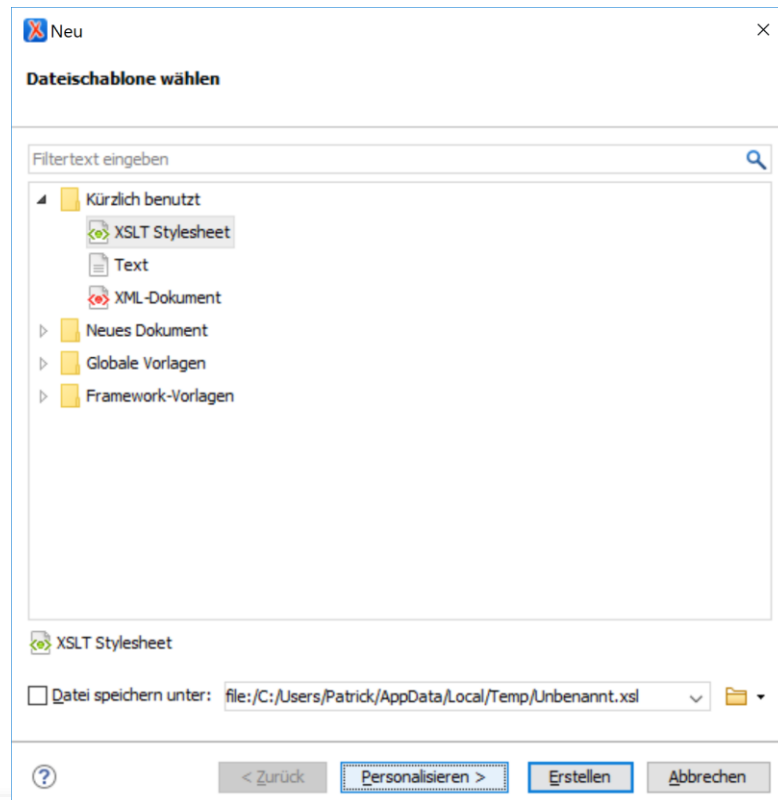
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  version="2.0">
  <xsl:template match="/">
    <h1>
      <xsl:value-of select="//titel" />
    </h1>
    <xsl:text>Hallo Welt</xsl:text>
  </xsl:template>
  <xsl:template match="lb"><br/></xsl:template>
</xsl:stylesheet>
```


Ein XSLT-Stylesheet in oXygen

- Strg+n

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
exclude-result-prefixes="xs"
version="2.0">

</xsl:stylesheet>
```



Transformationsszenarien

Wie bringt man XML, XSLT und den Prozessor zusammen?

- a) Dieses XML-Dokument soll mit jenem XSLT-Dokument verarbeitet werden
- b) Dieses XSLT-Dokument soll jenes XML-Dokument verarbeiten
- c) Lieber Prozessor, verarbeite jenes XML-Dokument mit jenem XSLT-Dokument
→ oXygen „Transformationsszenario“

oXygen-Transformationsszenario




Transformation-Szenarios konfigurieren (Strg+Umschalt+C)



Transformation-Szenarios anwenden (Strg+Umschalt+T)




oXygen-Transformationsszenario




 Neues Szenario

Name:

Speicherort: ☒ Projekt-Optionen ☐ Global-Optionen


XSLT FO-Prozessor **Ausgabedatei**

XML URL:   

XSL URL:   

[Mehr über \\${currentFileURL} lesen](#)


☐ Verwenden des Stylesheets der "xml-stylesheet" Deklaration


Transformator: 

Parameter (0)

Erweiterungen (0)

Zusätzliche XSLT-Stylesheets (0)



 Neues Szenario



Name:

Speicherort: ☒ Projekt-Optionen ☐ Global-Optionen

XSLT FO-Prozessor **Ausgabedatei**



Ausgabedatei

☐ Nach Datei fragen

☒ Datei speichern unter  

☒ In Browser-/Systemanwendung öffnen

☒ Gespeicherte Datei

☐ Andere Position  



☒ Im Editor öffnen


In der Ergebnis-Ansicht anzeigen als

☒ XML

☐ SVG

☐ XHTML

Grafik-URLs sind relativ zu dieser URL:  



Einige ~~Befehle~~ Elemente

Vorab:

- Nochmal zu XSLT 1.0, XSLT 2.0, XSLT 3.0 ...
- Ca. 50 Elemente (in XSLT 2.0)
- Von denen Sie ca. 10-20 brauchen (um schon mal sehr weit zu kommen)
 - Erklärung: ein großer Teil der Magie findet in XPath statt
- Einfaches Tutorial (etwas veraltet): https://www.w3schools.com/xml/xsl_intro.asp
- Nicht gut (nur XSLT 1.0): https://www.w3schools.com/xml/xsl_elementref.asp
- Eine etwas bessere Referenz (mit XSLT 2.0): <https://www.data2type.de/xml-xslt-xslfo/xslt/xslt-referenz/alphabetische-referenz-der-xslt-elemente/>

Einige Elemente

[xsl:stylesheet](#)

ist das Wurzelement eines XSL-Dokumentes.

[xsl:template](#)

enthält Instruktionen, deren Ausgabe direkt in das Ergebnisdokument eingetragen werden. Das Attribut `match(XPath)` sagt, auf welches Muster die Schablone passen soll

[xsl:apply-templates](#)

ruft das Template auf, dessen `match`-Attribut auf ein Kindknoten des aktuellen Kontextknotens passt.

[xsl:value-of](#)

wandelt eine im `select`-Attribut angegebene Sequenz in Werte um, die als Output in das Ergebnisdokument kopiert werden.

Einige Elemente

[xsl:for-each](#)

Tue für jedes ... etwas. Menge wird im Attribut `select` bestimmt.

[xsl:for-each-group](#)

dient dazu, eine im `select`-Attribut angegebene Eingabesequenz anhand eines Kriteriums zu gruppieren.

[xsl:sort](#)

sortiert eine (Knoten-)Sequenz anhand eines durch das `select`-Attribut ausgewählten Sortierkriteriums.

[xsl:if](#)

die klassische IF-Abfrage, ohne jedoch eine alternative ELSE-Möglichkeit zuzulassen.

[xsl:choose](#)

enthält eine oder mehrere [xsl:when](#)- und gegebenenfalls abschließend eine [xsl:otherwise](#)-Instruktion.

[xsl:variable](#)

definiert eine Variable, die mit einem Namen als Referenz versehen wird und der Werte, Strings oder ganzer Code zugeordnet werden kann.

Übungen

- „Hallo Wuppertal“
 - Alle Personen im Register ausgeben lassen
 - Alle Personen im Register alphabetisch
 - Die „unklaren“ Personen?
 - Alle Personen, push-Ansatz (apply-templates)
 - Mehr Ausgabe
 - Schöner Ausgabe
 - Gruppiert nach Alphabet
 - Gruppiert nach Geburtsjahren, nach Berufen?
 - *Was hätten **Sie** gerne?*
-
- Disclaimer: Das hier sind keine Textdaten!

Übung, vielleicht ...

13.2.2020

Liebes Tagebuch,

heute habe ich ein XSLT-Script geschrieben, das XML-Daten zu einem HTML-File transformiert, in dem Javascript eine SVG-Grafik erzeugt!

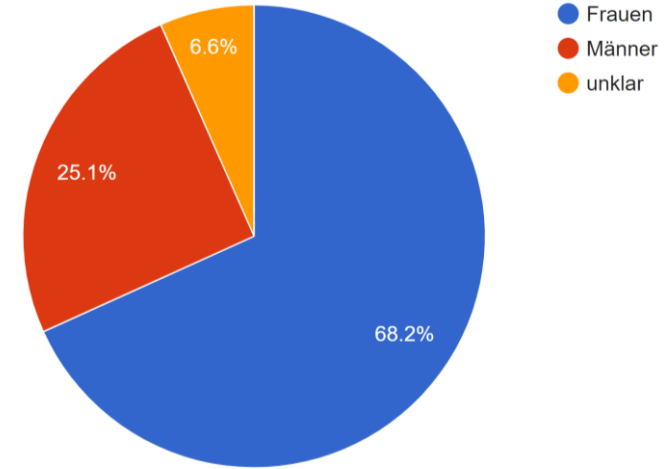
Vielleicht bin ich jetzt eine Software-Entwicklerin?
Jedenfalls kann ich jetzt mit Daten kritischer Editionen machen, was immer ich will.

Übungen

... nur wenn Sie *eXtrem* tapfer sind ...

- Was brauchen wir dafür?
 - Grundlogik: XSLT schreibt ein HTML-Dokument, in dem ein Javascript eine Grafik zeichnet
 - Grundsetting: [Google Chart Pie](#)
- Vorgehen
 - Logik der Datei (Kopie als pieChartStart.html) verstehen
 - Manuell anpassen (pieChartStart-Schritt2.html)
 - Dynamisch von XSLT generieren lassen

Geschlechter der Personen im Okopenko-Register



Strategien

Pull

- Hole gezielt Daten und tue etwas damit
→ typisch: for-each, for-each-group

Push

- Verarbeite Daten, wenn sie Dir über den Weg laufen
→ typisch: template, apply-templates, call-template

Wenn etwas schief läuft

- Die Fehlermeldungen sind meistens sehr aussagekräftig!
- Wenn zuviel rauskommt: das unsichtbare template für text()
- Wenn nichts rauskommt ... Typische Ursachen ...
 - Der Kontext ist ein anderer, als man denkt
 - Namespaces!!!
- Debugging
 - Adler und Robben
 - XSLT Debugger in oXygen
 - Kontrollausgaben