

X-TECHNOLOGIEN VERTIEFT

X-Technologien vertieft
Sommersemester 2019

Martina Scholger
Zentrum für Informationsmodellierung –
Austrian Centre for Digital Humanities

Organisatorisches

- Erreichbarkeit
 - martina.scholger@uni-graz.at
 - 0316 380 2291
 - Sprechstunde nach Vereinbarung
- Zur Lehrveranstaltung
 - Die **Unterlagen** zur LV befinden sich auf der Lernplattform Moodle
 - **Benotung:** Mitarbeit, Hausübungen, Abschlussprojekt
 - **Anwesenheit:** prüfungsimmanent
 - **Software:** XML-Editor oXygen

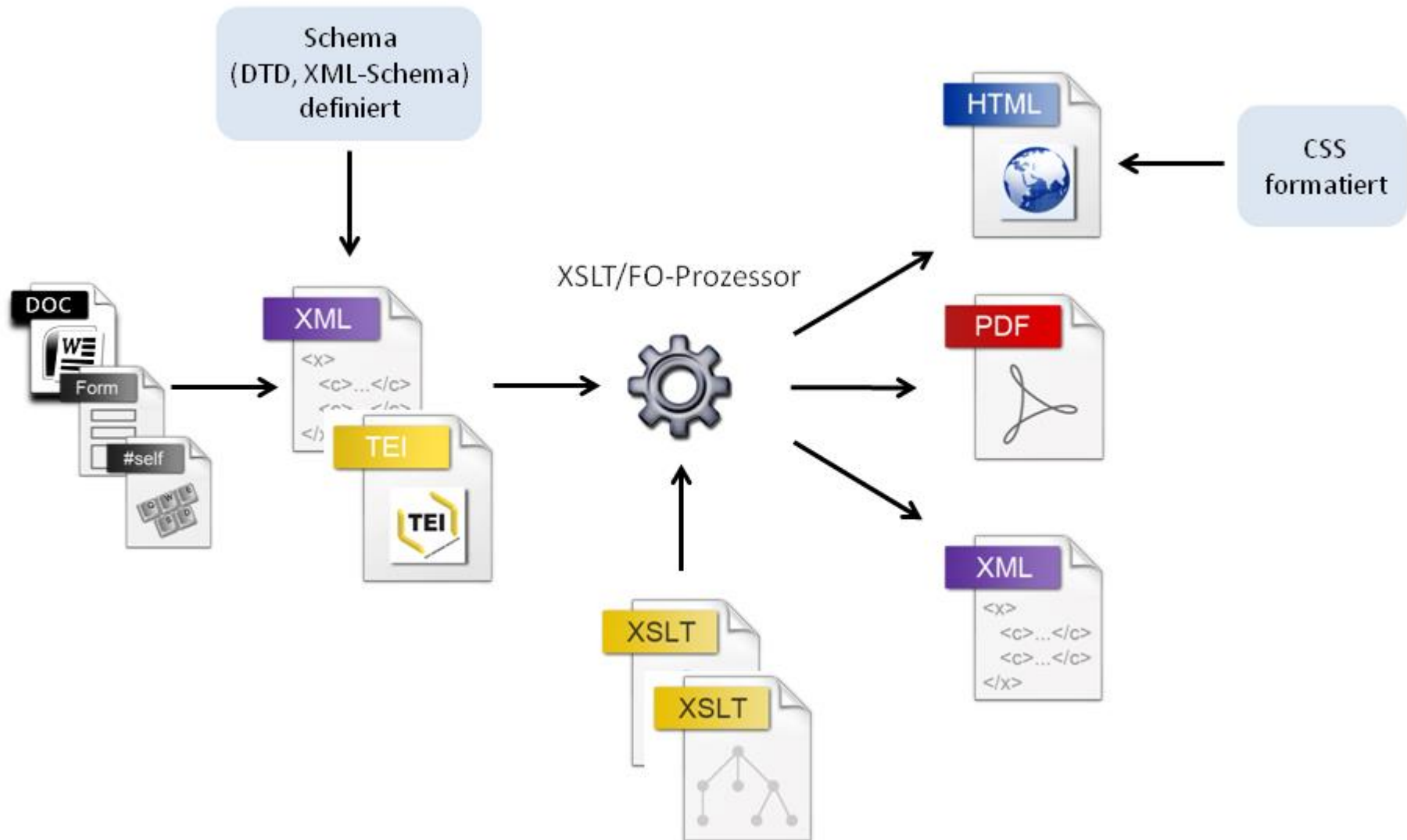
Semesterplan

- XPath Wiederholung
 - XML als Baum
 - XPath-Ausdrücke
 - Achsen
 - Lokalisierungsschritte
 - Knotentypen
 - Prädikate
 - Funktionen
 - Vergleiche
- XSLT
 - Templates
 - Schleifen
 - Sortierung
 - Gruppierung
- XML to HTML / XML to Print / XML to Excel / XML to Text / XML to XML
- XQuery
 - Syntax
 - FLOWR
 - Funktionen
- XML-Datenbank (eXist-db oder BaseX)
 - Installation
 - Kleinprojekt anlegen
- Regular Expressions
- Geisteswissenschaftliches Asset Management System GAMS
 - Verwendung des Clients
 - Verwendung von XSL-Templates

Was ist XSL?

- Das Akronym steht für „Extensible Stylesheet Language“ und bezeichnet eine Familie von Sprachen zur Abfrage und Transformation von XML-Dokumenten. Sie besteht aus drei Teilsprachen:
 - **XML Path Language (XPath)**
 - Abfragesprache, um Teilbäume eines XML-Eingabedokumentes zu adressieren und zu manipulieren.
 - **XSL Transformations (XSLT)**
 - Transformationssprache, um XML-Eingabedokumente in unterschiedliche Ausgabeformate (XML, HTML, PDF, Text, etc.) zu überführen.
 - **XSL Formatting Objects (XSL:FO)**
 - Seitenbeschreibungssprache, die der Druckaufbereitung von XML-Eingabedokumenten dient.

XSLT Publikationsworkflow



XML Dokumente





XML Repräsentation

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<book>
```

```
  <introduction>Dieses Buch führt Sie durch  
  die Grundlagen von XSLT ... </introduction>
```

```
  <chapter>
```

```
    <heading>Einführung</heading>
```

```
    <section>Das Design von XSLT ...
```

```
  </section>
```

```
    <section>XML Grundlagen ... </section>
```

```
  </chapter>
```

```
  <chapter>
```

```
    <heading>XPath</heading>
```

```
    <section>Das XPath-Datenmodell ...
```

```
  </section>
```

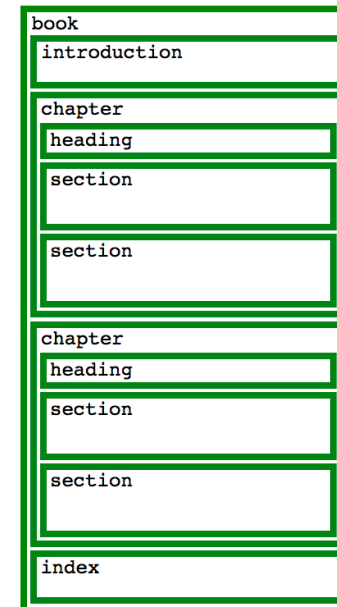
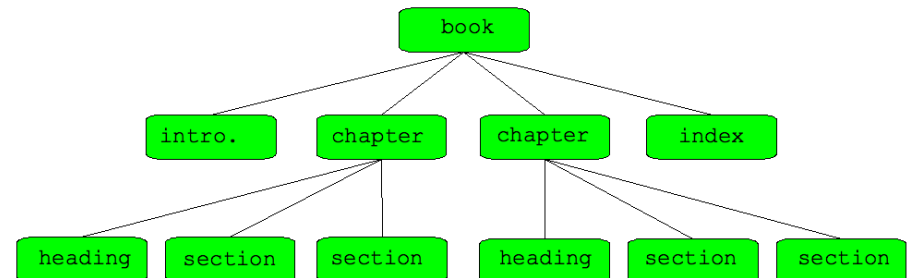
```
    <section>Lokalisierungspfade ...
```

```
  </section>
```

```
  </chapter>
```

```
  <index> ... Symbole ... </index>
```

```
</book>
```



XPath bzw. XPath-Ausdruck

- *... is a language for addressing parts of an XML document (XPath Specification – W3C)*
- ... dient zur Navigation in XML-Dokumenten und Erzeugung von „Rückgaben“.
- ... wird verwendet um Teile zu extrahieren, neu zu organisieren und anzuordnen, Ergänzungen zu machen, zu zählen, zu nummerieren, etc.
- ... wird von anderen X-Technologien verwendet bzw. bildet die Voraussetzung dafür: XSLT (eXtensible Stylesheet Language), XQuery (XML Query Language), XLink und XPointer.

Gliederungsansicht oXygen

XPath 2.0 XPath ausführen auf 'Ausgewählte Projektressourcen'

Gliederung

Elementnamen-Filter

- TEI "sha-ham"
 - teiHeader
 - fileDesc
 - titleStmt Hamlet, Prince of Denmark
 - title Hamlet, Prince of Denmark
 - author William Shakespeare
 - respStmt Craig A. Berry, Martin Mueller, and Clifford Wulfman
 - respStmt Jeffrey Cousens and Bill Parod
 - respStmt Lawrence Berland, Hilary Bina, Katherine Gould, Kreg Segal, Nicholas
 - publicationStmt
 - sourceDesc "false" The Wordhoard Shakespeare
 - revisionDesc
 - text
 - front
 - body
 - div "sha-ham1" Act 1
 - div "sha-ham2" Act 2
 - head Act 2
 - div "sha-ham201" Act 2, Scene 1
 - div "sha-ham202" Act 2, Scene 2
 - div "sha-ham3" Act 3
 - div "sha-ham4" Act 4
 - div "sha-ham5" Act 5

mws.294.xml x hamlet.xml x

TEI	teiHeader	fileDesc	titleStmt	respStmt
1	<?xml version="1.0" encoding="UTF-8"?>			
2	<TEI xmlns="http://www.tei-c.org/ns/1.0" xml:id="sha-ham">			
3	<teiHeader>			
4	<fileDesc>			
5	<titleStmt>			
6	<title>Hamlet, Prince of Denmark</title>			
7	<author>William Shakespeare</author>			
8	<respStmt>			
9	<name>Craig A. Berry, Martin Mueller, and Clifford Wulfman</name>			
10	<resp>editors</resp>			
11	</respStmt>			
12	<respStmt>			
13	<name>Jeffrey Cousens and Bill Parod</name>			
14	<resp>technical editors</resp>			
15	</respStmt>			
16	<respStmt>			
17	<name>Lawrence Berland, Hilary Bina, Katherine Gould, Kreg Segal, Nicholas			
18	<resp>editorial assistants</resp>			
19	</respStmt>			
20	</titleStmt>			
21	<publicationStmt>			
22	<p>			
23	<title>The WordHoard Shakespeare</title>			
24	is a joint project of the Perseus Project at Tufts University, The Northwest			
25	<title>The Globe Shakespeare</title>, the one-volume version of the			
26	<title>Cambridge Shakespeare</title>, edited by W. G. Clark, J. Glover, and			
27	<title>Internet Shakespeare</title>			
28	editions of the quartos and folios have been consulted to create a modern			
29	</p>			

XSLT

- Extensible Stylesheet Language
- W3C Standard seit 1999
- Sprache zur Transformation von XML Dokumenten
- XSL ist auch XML
- Eingabebaum (XML) – Transformationsbaum (XSLT) –
Ausgabebaum (HTML, XML, Text, etc.)

Wozu XSLT?

- Verwalten von Dokumenten
 - Aufräumen von Daten
 - Anreicherung mit neuen Daten
 - Zusammenführen von mehreren Dokumenten
 - Aufspalten in Einzeldokumente
- Aufbereiten für den Datenaustausch
- Aufbereitung zur Darstellung und Publikation
 - Auswahl treffen
 - Organisieren
 - Etc.

XML Input – Stylesheet – HTML Output

The Input Document

```
<?xml version="1.0" encoding="UTF-8"?>  
<paragraph>Hello, world!</paragraph>
```

The Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>  
<xsl:stylesheet  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  version="1.0">  
  
  <xsl:template match="paragraph">  
    <html>  
      <head>  
        <title>A Short Test Document</title>  
      </head>  
      <body>  
        <div>  
          <p>  
            <xsl:apply-templates/>  
          </p>  
        </div>  
      </body>  
    </html>  
  </xsl:template>  
</xsl:stylesheet>
```

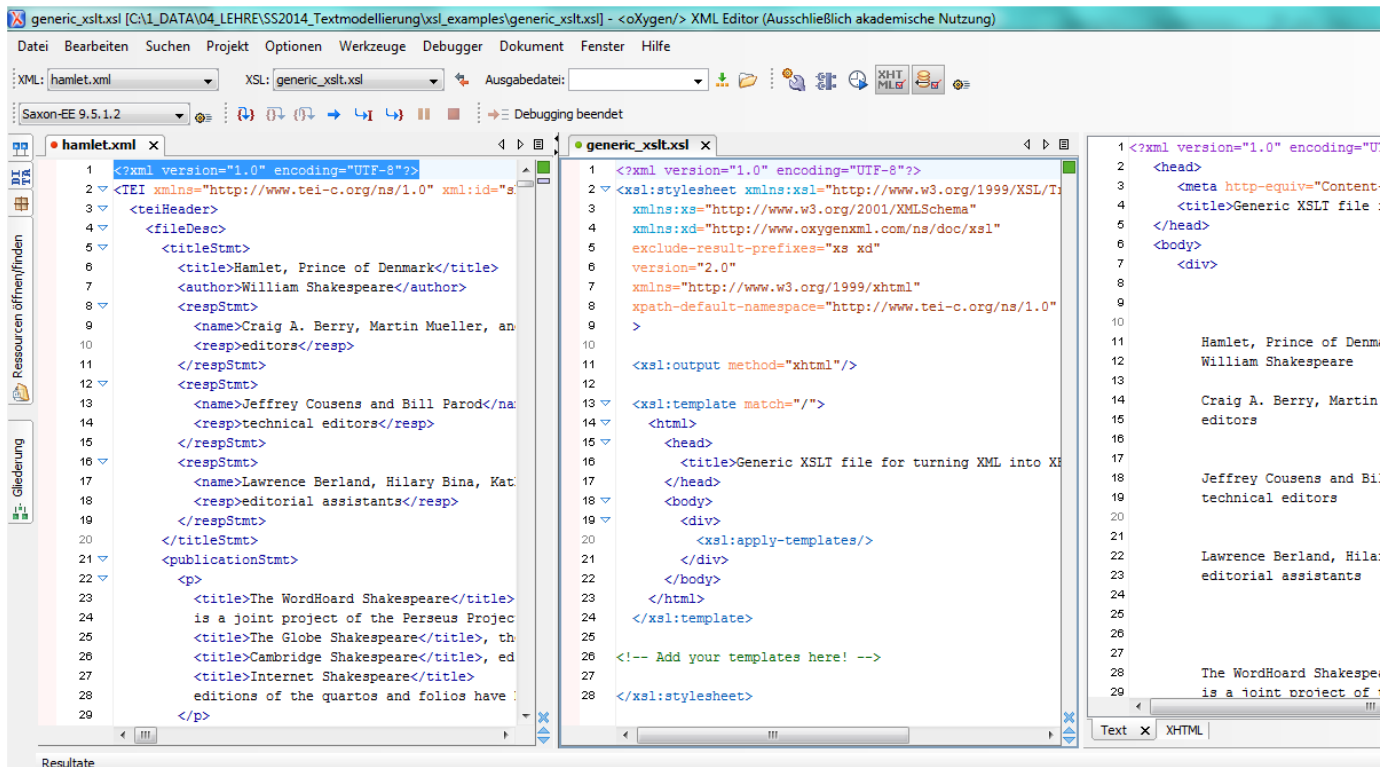
The Output Document

```
<html>  
  <head>  
    <title>A Short Test Document</title>  
  </head>  
  <body>  
    <div>  
      <p>  
        ????  
      </p>  
    </div>  
  </body>  
</html>
```



Transformation XML to Text/XML/HTML

- Öffnen Sie hamlet.xml und trans_01.xsl
- Verwenden Sie den XSLT-Debugger in Oxygen



XPath bzw. XPath-Ausdruck

- *... is a language for addressing parts of an XML document* (XPath Specification – W3C)
- ... dient zur Navigation in XML-Dokumenten und Erzeugung von „Rückgaben“.
- ... wird verwendet um Teile zu extrahieren, neu zu organisieren und anzuordnen, Ergänzungen zu machen, zu zählen, zu nummerieren, etc.
- ... wird von anderen X-Technologien verwendet bzw. bildet die Voraussetzung dafür: XSLT (eXtensible Stylesheet Language), XQuery (XML Query Language), XLink und XPointer.

Knotentypen

Ein Knoten ist eine bestimmte Position im XML Baum

- Dokumentknoten
- Wurzelknoten `<TEI> . . . </TEI>`
- Elementknoten

```
<element> . . . Textinhalt . . . </element>
<element>
    <element> . . . Textinhalt . . . </element>
</element>
```

- Attributknoten `<element attribute="value">`
- Textknoten `... Textinhalt ...`

Terminologie

- Elemente und Textknoten werden über
 - Vorfahren
 - Nachkommen
 - Eltern
 - Geschwisterbeschrieben

Bewegung im Baum

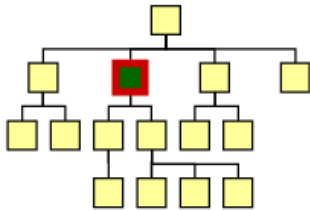
- Lokalisierungsschritte
- Knotentests
- Lokalisierungspfade
 - [Schritt]/[Schritt]/[Schritt]/[Schritt]/[Schritt]
`/TEI/teiHeader/fileDesc/titleStmt/title`
- Kontext
 - Absolute Pfade
`/TEI/teiHeader/fileDesc/titleStmt/title`
 - Relative Pfade (ausgehend von titleStmt)
`respStmt/name`

Kontextknoten

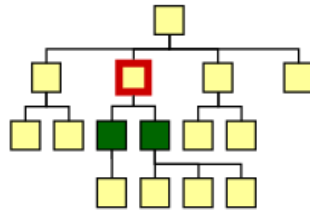
- Der Kontextknoten (context node) bezeichnet jene Stelle im Dokument, an der wir uns gerade befinden
- Vom Kontextknoten aus kann man sich an jede beliebige Stelle im Dokument bewegen
- Wir tun das über XPath-Achsen

XPath-Achsen

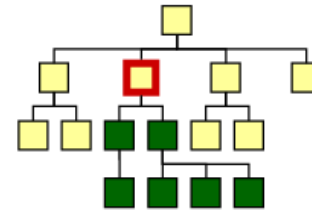
self::



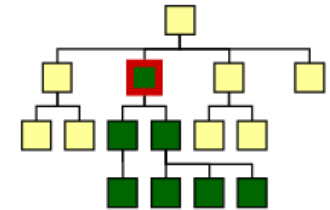
child::



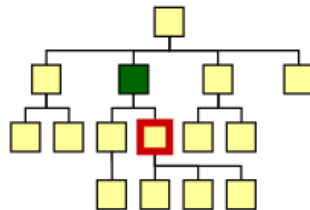
descendant::



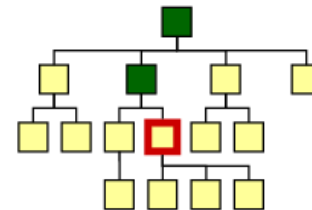
descendant-or-self::



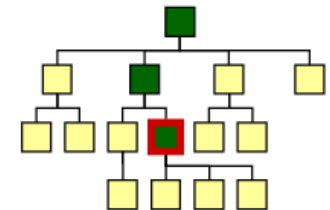
parent::



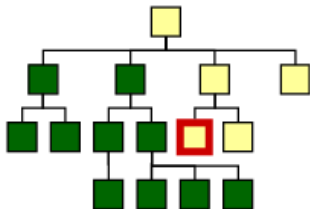
ancestor::



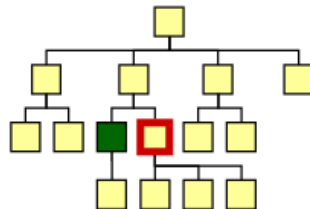
ancestor-or-self::



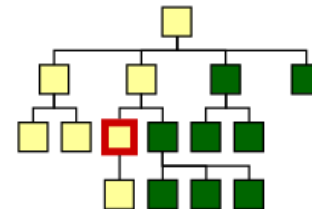
preceding::



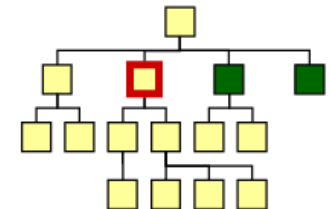
preceding-sibling::



following::



following-sibling::



□ Kontextknoten
■ Knoten der Achse

XPath-Achsen ::

Achse	Kurzform	Langform	Erklärung
Kontextknoten	.	self::node()	der aktuelle Knoten
Kindknoten	head	child::head	die direkten Kindelemente des aktuellen Knotens
	*	child::*	Alle Kindelemente des aktuellen Knotens
Nachkommen	//div	descendant::div	Alle Nachkommen des aktuellen Knotens
Elternknoten	..	parent::node()	Das Elternelement des aktuellen Knotens
	[none]	parent::author	Das Elternelement (sofern es sich um das Element <author> handelt) wird selektiert
Vorfahren	[none]	ancestor::*	Alle Vorfahren des aktuellen Knotens
Nachfolgende	[none]	following::p	Alle nachfolgenden <p>-Elemente
	[none]	following-sibling::*	Alle nachfolgenden Geschwisterelemente
Vorhergehende	[none]	preceding::p	Alle vorhergehenden <p>-Elemente
	[none]	preceding-sibling::*	Alle vorhergehenden Geschwisterelemente
Attribute	@target	attribute::target	Das @target Attribut des Kontextknotens
	@*	attribute::*	Alle Attribute des Kontextknotens

Lokalisierungsschritte

- Syntax: Achse::Knotentest[Prädikat]

`//castList/castItem/role[@xml:id='Claudius']/following-sibling::roleDesc`

Rückgabebetypen von XPath-Ausdrücken

- Elemente
- Knotenmengen
- Zahlen
- Strings
- Wahrheitswerte/Boolean
- Sequenzen

Baukasten

- Verkettete Lokalisierungsschritte /..../..../
- Bedingungen, Prädikate [...]
- Klammern (...(...))
- Operatoren
 - and or | = != < > + - * div
- Funktionen
 - Funktionsname(Argument,Argument)

Übung: Hamlet

- Öffnen Sie die Datei **hamlet.xml** in Oxygen
- Stellen Sie sicher, dass der XPath Suchschlitz auf „XPath 2.0“ eingestellt ist
- Suchen Sie nach dem Autor des Dokuments

XPath Funktionen

- Mit Funktionen kann die Übersetzung der XML-Ausgangsdaten in den Ergebnisbaum (XML, HTML, FO, etc.) kontrolliert, eingeschränkt und verändert werden.
- Eine Funktion wird über ihren **Namen** aufgerufen, gefolgt von **runden Klammern ()**, die **Parameter** enthalten können.

`count(//sp)`

`//TEI/teiHeader//titleStmt/title/substring-after(., 'The Tragedie of Hamlet,')`

- Parameter sind jene Einheiten (Elemente, Attribute, Textknoten, etc.), auf die die Funktion angewendet werden soll.
- Funktionen können einen gesamten Xpath-Ausdruck enthalten oder können innerhalb eines Ausdrucks zur weiteren Verfeinerung verwendet werden.

Funktionen (Testfile: hamlet.xml)

- `count()`
 - Liefert die Anzahl der Knoten der übergebenen Knotenmenge
 - Beispiel: `count(//speaker)`
 - Zählt alle Sprecher
 - Aufgabe: zählen Sie alle Zeilen
 - Lösung: `count(//l)`
 - Aufgabe: zählen Sie alle Zeilen des dritten Aktes
 - Lösung: `count(//div[@type='act'][@n='3']//l)`
- `position()`
 - Ermittelt die Position eines Knotens im Dokument
 - Aufgabe: Geben Sie den Sprecher des jeweils 5. Sprechaktes in Akt 1 aus
 - Lösung (langform): `//div[@type='act'][@n='1']//sp[position()=5]/speaker`
 - Lösung (kurzform): `//div[@type='act'][@n='1']//sp[5]/speaker`

Funktionen (Testfile: hamlet.xml)

- `name()`
 - Liefert eine Liste aller Elemente die innerhalb der übergebenen Knotenmenge auftreten
 - Beispiel: `//sp/*/name()`
 - Gibt eine Liste der innerhalb der Sprechakte verwendeten Elemente aus
- `last()`
 - Liefert den letzten Knoten der übergebenen Knotenmenge
 - Aufgabe: geben Sie den jeweils letzten Sprecher einer Szene aus
 - Beispiel: `//sp[position()=last()]/speaker`
- `not()`
 - Kehrt den Wahrheitswert des Xpath-Ausdrucks um
 - Aufgabe: geben Sie alle Sprecher aus, bis auf den jeweils letzten
 - Beispiel: `//sp[not(position()=last())]/speaker`

Funktionen (Testfile: hamlet.xml)

- `distinct-values()`
 - Gibt nur die unterschiedlichen Ergebniswerte aus
 - Aufgabe: geben Sie die unterschiedlichen SprecherInnen aus
 - Lösung: `distinct-values(//speaker)`
- `string-length()`
 - ermittelt die Länge eines Textstrings
 - Aufgabe: geben Sie die Textlängen der einzelnen Zeilen aus
 - Lösung: `//l/string-length(.)`
- `concat(input-string1, input-string2, ...)`
 - Fügt Argumente zusammen
 - Aufgabe: Setzen Sie vor den Rollennamen den Textstring „Rolle:“
 - Lösung: `//listPerson/person/concat('Rolle: ', persName[@type='standard'])`

Funktionen (Testfile: hamlet.xml)

- `substring-before(input-string, substring)`
 - Liefert jene Teilzeichenkette eines Textknotens (`input-string`), die vor dem angegebenen Zeichen (`substring`) liegt
 - Aufgabe: geben Sie aus dem ersten `<title>`-Element im `<titleStmt>` alle Zeichen vor dem Beistrich aus, also „The Tragedie of Hamlet“
 - Lösung: `substring-before(//titleStmt/title[@type='statement'], ',')`
- `substring-after(input-string, substring)`
 - Liefert jene Teilzeichenkette eines Textknotens, die hinter dem angegebenen Zeichen liegt
 - Aufgabe: geben Sie aus dem `<title>`-Element im `<titleStmt>` alle Zeichen nach dem Beistrich aus
 - Lösung: `substring-after(//titleStmt/title[@type='statement'], ',')`
- `substring(input-string, start, length)`
 - Extrahiert aus einem String ab einer bestimmten Position (`start`) eine bestimmte Menge an Zeichen (`length`)
 - Aufgabe: Geben Sie aus dem ersten `<title>`-Element im `<titleStmt>` den Textstring ab dem 5. Buchstaben mit einer Zeichenlänge von 8 Buchstaben aus
 - Lösung: `substring(//titleStmt/title[@type='statement'], 5, 8)`

Funktionen (Testfile: hamlet.xml)

- `replace(input-string, regex-pattern, replacement-string)`
 - Ersetzt Zeichen und Zeichenketten in einem Textstring
 - Aufgabe: Ersetzen Sie alle „er“ in Bernardo durch „a“
 - Lösung: `replace('Bernardo', 'er', 'a')`
- `translate(input-string, characters-to-match, replacement-characters)`
 - Ersetzt einzelne Zeichen in einem Textstring (nur 1:1 möglich)
 - Aufgabe: Ersetzen Sie den Buchstaben “e” in Bernardo durch “a”
 - Lösung: `translate('Bernardo', 'e', 'a')`
- `upper-case(input-string)`
 - Übersetzt die Zeichenkette in Großbuchstaben
 - Aufgabe: Übersetzen Sie den Titel in Großbuchstaben
 - Lösung: `//titleStmt/title/upper-case(.)`
- `lower-case(input-string)`
 - Übersetzt die Zeichenkette in Kleinbuchstaben
 - Aufgabe: Übersetzen Sie den Titel in Kleinbuchstaben
 - Lösung: `//titleStmt/title/lower-case(.)`

Funktionen (Testfile: hamlet.xml)

- `normalize-space()`
 - schneidet überflüssigen Leerraum weg, reduziert Weißraum auf ein Leerzeichen
 - Vergleichen Sie die Ergebnisse aus:
`//div[@type='act'][@n='1']//l[@n='1']/string-length()` und
`//div[@type='act'][@n='1']//l[@n='1']/string-length(normalize-space(.))`
 - Der Leerraum muss zuerst entfernt werden, sonst wird er mitgezählt
- `starts-with(input-string, substring)`
 - Beginnt der Textknoten (input-string) mit der übergebenen Zeichenkette (substring) wird „wahr“ zurückgeliefert
 - Beispiel: `//l[starts-with(normalize-space(), 'G')]`
 - Alle gesprochenen Zeilen, die mit „G“ beginnen. Zuvor müssen die Zeilen normalisiert werden.
- `current-date()`
 - Gibt das aktuelle Datum aus

Funktionen (Testfile: hamlet.xml)

- `matches(input-string, regex)`
 - Bestimmt, ob ein String einem bestimmten Muster entspricht
 - Aufgabe: Geben Sie alle Zeilen aus, die das Wort „night“ enthalten
 - Lösung: `//[matches(., 'night')]`
 - Lösung: `//[matches(., 'night')] | //ab[matches(., 'night')]`
- `tokenize(input-string, regex)`
 - Teilt eine Zeichenkette in Teilzeichenketten auf, der reguläre Ausdruck bestimmt den Teiler
 - Aufgabe: Geben Sie alle einzelnen Tokens der Zeilen aus.
 - Lösung: `//tokenize(normalize-space(.), ' ')`

Funktionen (Testfile: hamlet.xml)

- `reverse(input-string)`
 - Kehrt die Ergebnisliste um: (1 to 10) funktioniert, (10 to 1) aber nicht, daher `reverse(1 to 10)`
 - Aufgabe: Geben Sie alle speaker in umgekehrter Reihenfolge aus.
 - Lösung: `reverse(//speaker)`
- `name(input-string)`
 - Gibt den GI (generic Identifier) des Elements zurück
 - `//*/name()` sucht nach allen Elementen im Dokument
 - Aufgabe: Suchen Sie nach allen unterschiedlichen Elementen im Textabschnitt
 - Lösung: `distinct-values(//text//*/name())`

Numbers

Input-Argument: 291, 36, 473, 27

- `count((arg))`
 - Anzahl der Argumente
- `avg((arg))`
 - Durchschnittswert
- `max((arg))`
 - Maximalwert der Sequenz
- `min((arg))`
 - Minimalwert der Sequenz
- `sum((arg))`
 - Summe

`count(//sp/speaker/count(.))` oder aber viel kürzer `count(//speaker)`

Wertvergleich

Vergleich von einem Item mit exakt einem anderen Item

- **eq** equal to
- **ne** not equal to
- **gt** greater than
- **ge** greater than or equal to (not less than)
- **lt** less than
- **le** less than or equal to (not greater than)

Geben Sie alle Elemente aus, bei denen das @who-Attribut den Wert FH enthält

```
/*[@who eq 'FH']
```

Allgemeine Vergleichsoperatoren

Vergleich von mehreren Items auf beiden Seiten möglich

- `=` equal to
- `!=` not equal to
- `>` greater than (oder `>`;))
- `>=` greater than or equal to (nicht kleiner als; oder `>=`;))
- `<` less than (oder `<`;))
- `<=` less than or equal to (nicht größer als; oder `<=`;))

Geben Sie alle Elemente aus, bei denen das `@who`-Attribut den Wert `FH` oder `SS` enthält

```
/*[@who = ('FH', 'SS')]
```

```
/*[@who eq 'FH' or @who eq 'SS']
```

Funktionen

- Einfacher Einstieg
 - https://www.w3schools.com/xml/xpath_intro.asp
- XPath und XSLT Funktionen (w3schools):
 - https://www.w3schools.com/xml/xsl_functions.asp
- XPath Spezifikationen des (W3C):
 - https://www.w3.org/standards/techs/xpath#w3c_all
- Versionen:
 - XPath 1.0, XPath 2.0, XPath 3.0

XSLT

- Extensible Stylesheet Language
- W3C Standard seit 1999
- Sprache zur Transformation von XML Dokumenten
- XSL ist auch XML
- Eingabebaum (XML) – Transformationsbaum (XSLT) –
Ausgabebaum (HTML, XML, Text, etc.)

Hauptkomponenten von XSLT

- `<xsl:stylesheet>`
- `<xsl:output>`
- `<xsl:template>`
- `<xsl:apply-templates>`

<xsl:stylesheet>

<xsl:stylesheet

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
xmlns="http://www.w3.org/1999/xhtml"  
xpath-default-namespace="http://www.tei-c.org/ns/1.0"  
version="2.0">
```

- **xsl:stylesheet**: umschließendes Element, enthält das gesamte Stylesheet
- **xmlns:xsl**: verpflichtend anzugeben
- **xmlns**: Namensraum des Zieldokuments (xhtml)
- **xpath-default-namespace**:
 - optional anzugeben, wenn TEI-Dokumente transformiert werden
 - XPath nimmt an, dass sich Pfadausdrücke auf TEI-Dokument beziehen
 - Alternativ dazu wird der Namensraum des TEI-Dokuments angegeben
 - **xmlns:tei**="http://www.tei-c.org/ns/1.0"
 - XPath muss dann immer mit **tei:** beginnen
- **version**: verpflichtend anzugeben

<xsl:output>

```
<xsl:output method="xml" indent="yes" encoding="utf-8"/>
```

- **method:** Weitere Werte für “method” sind: xml, text, html
- **indent:** wird auf “yes” gesetzt, um lange Zeilen umzuberechnen
- **encoding:** Zeichenkodierung immer auf utf-8 setzen

<xsl:template>

- Ein XSLT-Dokument besteht aus einer Reihe von Template-Regeln

```
<xsl:template match="div">
```

- Enthält Anweisungen, die bei einem “match” ausgeführt werden
- Das @match-Attribut enthält einen XPath-Ausdruck. D.h. ein Knoten (Element, Attribut, etc.) aus dem Eingabedokument wird selektiert und das Template darauf angewendet.

```
<xsl:template match="div">  
  <p><xsl:apply-templates/></p>  
</xsl:template>
```

- Prozessor trifft auf <div> im Eingabedokument (XML)
- Ein <p> im Ausgabedokument (HTML) wird erzeugt.
- Verarbeite Inhalt von <div> (XML)
- Gib das Resultat innerhalb von <p> (HTML) aus

<xsl:apply-templates>

- Leeres Element
- Anweisung, dass die Verarbeitung weiterer Templates fortgeführt werden soll
- Die zu verarbeitende Knotenmenge wird über einen Xpath-Ausdruck bestimmt
- Wie der Knoten verarbeitet wird, wird in der Templaterregel angegeben

```
<xsl:apply-templates select="[XPath]" />
```

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

```
<xsl:template match="div">  
  <xsl:apply-templates select="lg" />  
</xsl:template>
```

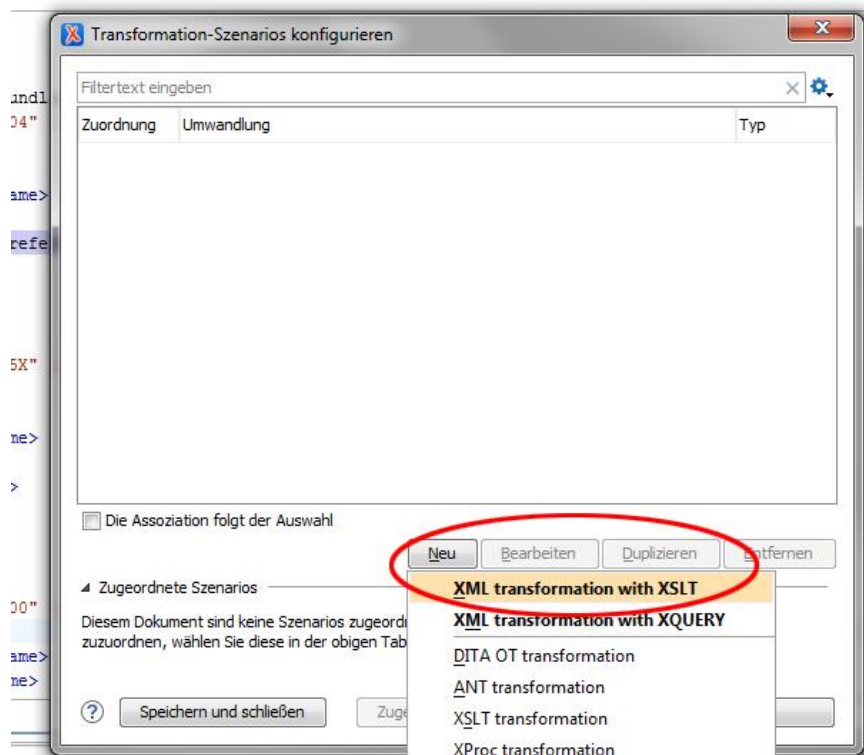
Zum Einstieg ein einfaches Beispiel

- Öffnen Sie die Datei **verse.xml** in Oxygen
- Erstellen Sie zu dieser Datei ein Stylesheet mit dem Namen **verse.xsl**
- Wechseln Sie zum XSLT Debugger
- Ausgabe der XML-Elemente in HTML:
 - XML `<div>` in HTML `<section>`
 - XML `<head>` in HTML `<h1>`
 - XML `<lg>` in HTML `<p>`
 - XML `<l>` in HTML durch `
` voneinander getrennt
 - XML `<byline>` in HTML `<p>`
- Die `<byline>` soll rechtsbündig ausgegeben werden, ergänzen Sie das Element um eine Klasse und fügen Sie ein CSS Dokument hinzu.

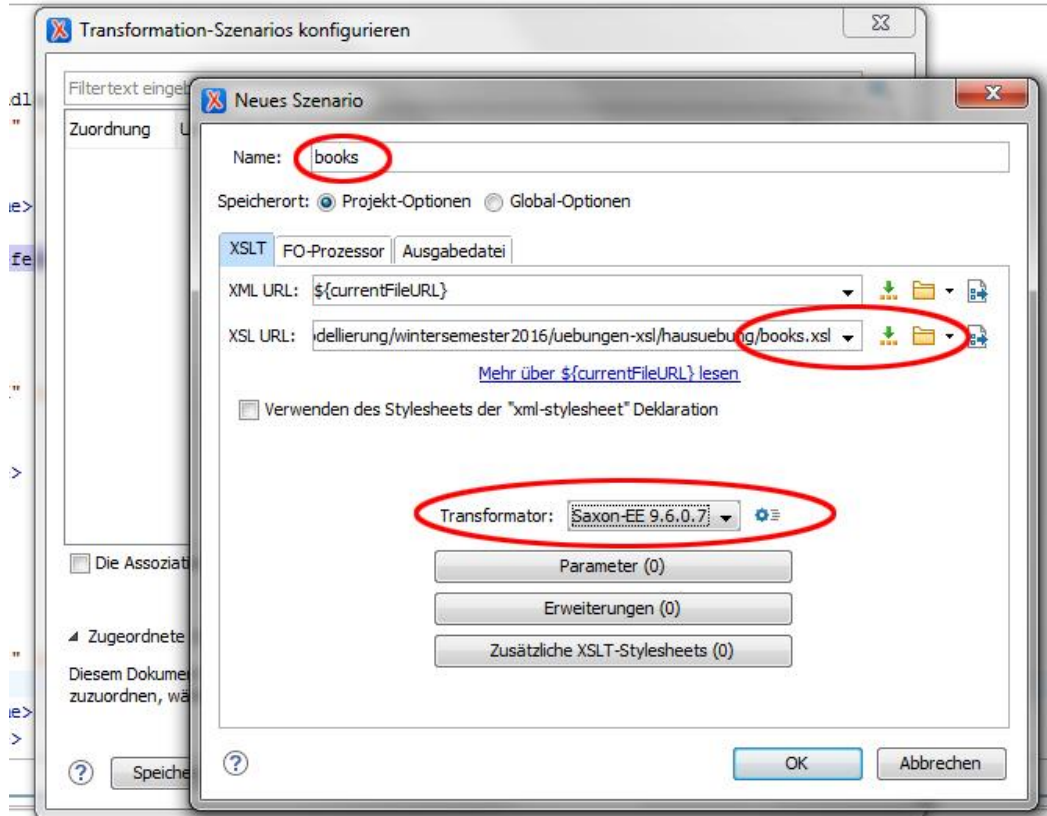
Transformationsszenario einrichten



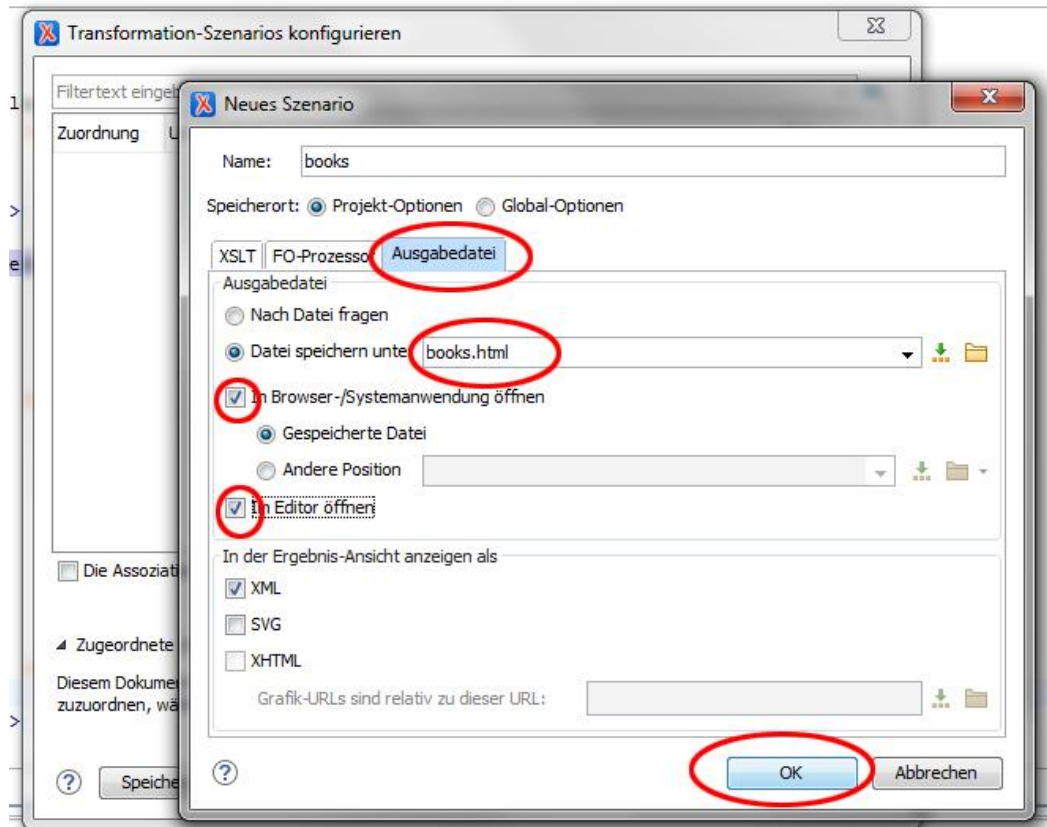
Transformation-Szenarios konfigurieren (Strg+Umschalt+C)



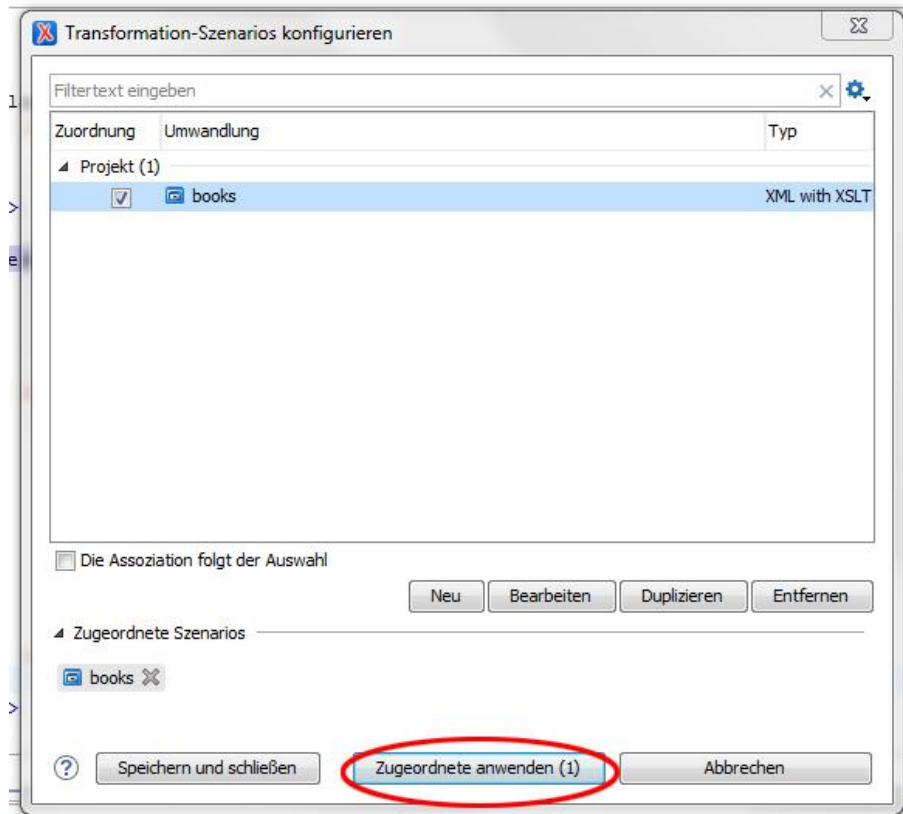
Transformationsszenario einrichten



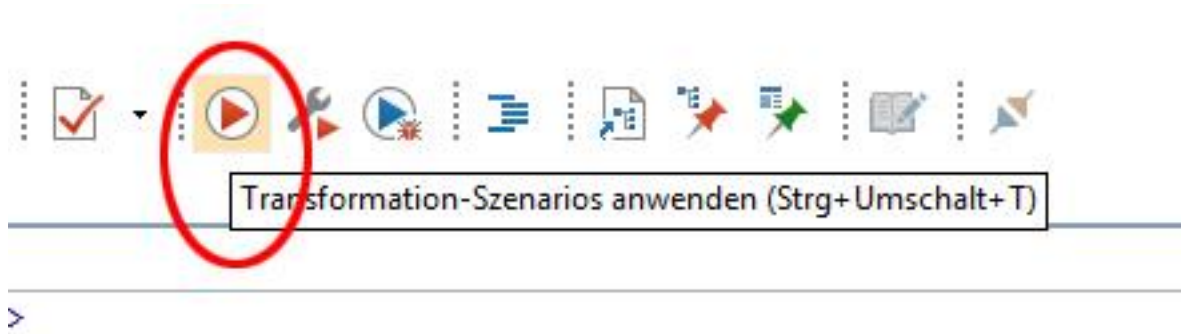
Transformationsszenario einrichten



Transformationsszenario einrichten



Transformationsszenario einrichten



TEI to HTML

- ```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/1999/xhtml"
 xpath-default-namespace="http://www.tei-c.org/ns/1.0"
 version="2.0">
 <xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
 <html>
 <head>
 <title>Titel ausgeben</title>
 </head>
 <body>
 <xsl:apply-templates/>
 </body>
 </html>
 </xsl:template>

 <!-- weitere Templaterregeln -->

</xsl:stylesheet>
```

# XSLT Processing Model

- Wo starten wir?
  - Beim Dokumentknoten (der „Vorgänger“ des Wurzelements)
- Was passiert, wenn kein passendes Regelwerk gefunden wird?
  - Die Defaultregeln (Built-in) kommen zum Tragen
- Was passiert, wenn genau eine Regel zutrifft?
  - Sie wird angewendet
- Was passiert, wenn mehr als eine Regel zutrifft?
  - Templates werden überschrieben
  - Built-in priority
  - User-specified priority

# Built-in Regeln

- Element: verarbeite die Kindknoten (Elemente, Textknoten), wende Templaterregeln an (build-in oder spezifische Regeln)
- Attribute: kein Output
- Text: der Text wird ausgegeben

# Built-in priority

```
<xsl:template match="div">
 <act>
 <xsl:apply-templates/>
 </act>
</xsl:template>
```

```
<xsl:template match="div/div">
 <scene>
 <xsl:apply-templates/>
 </scene>
</xsl:template>
```

# User-specified priority

```
<xsl:template match="div" priority="10">
 <act>
 <xsl:apply-templates/>
 </act>
</xsl:template>
```



# Zwei Extremfälle

Eine leere Regel für das Wurzelement:

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

 <xsl:template match="/">

 </xsl:template>

</xsl:stylesheet>
```



# Zwei Extremfälle

Eine leeres Regelwerk:

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

</xsl:stylesheet>
```

## <xsl:value-of />

- Schreibt den Wert (Text) des selektierten Knotens in das Ergebnisdokument.
- Der Text der Kindelemente wird zwar ebenfalls berücksichtigt, kann aber nicht mehr weiterverarbeitet werden

```
<h1><xsl:value-of select="head" /></h1>
```

Ergebnis:

```
<h1>London</h1>
```

# Elemente und Attribute

`<xsl:element name="[elementname]">`

- Konstruiert ein Element im Ergebnisbaum
- Elemente können auch direkt in das Ergebnis geschrieben werden:  
`<elementname>Text</Elementname>`

`<xsl:attribute name="[attributname]">`

- Konstruiert ein Attribut im Ergebnisbaum
- Attribute können auch direkt in das Ergebnis geschrieben werden:  
`<elementname attributname="attributwert">Text</Elementname>`
- Dynamische Wertausgabe im Attribut:  
`<elementname attributname="{XPATH}">Text</Elementname>`

`<xsl:text>`

- Schreibt Zeichendaten in das Zieldokument, z.B. Leerzeichen, Beistriche, sonstigen Text: `<xsl:text>`, `</xsl:text>`

# Attribute

- XML Input:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
 <person gender="male">
 <firstname>William</firstname>
 <lastname>Blake</lastname>
 </person>
 <person gender="female">
 <firstname>Agatha</firstname>
 <lastname>Christie</lastname>
 </person>
</persons>
```

- XSL Transformation (langform):

```
<xsl:template match="person">

 <xsl:attribute name="class">
 <xsl:value-of select="@gender" />
 </xsl:attribute>
 <xsl:value-of select="lastname" />

</xsl:template>
```

- XSL Transformation (kurzform):

```
<xsl:template match="person">
 <li class="{@gender}">
 <xsl:value-of select="lastname" />

</xsl:template>
```

# Attribute

- HTML Ergebnis

```
<li class="male">Blake
```

```
<li class="female">Christie
```

- CSS

```
.male {
 background-color: yellow;
 font-weight: bold;
}
```

```
.female {
 background-color: green;
 color: white;
 font-style: italic;
}
```

# Schleifen mit <xsl:for-each>

```
<xsl:for-each select="XPath">
```

```
...
```

```
</xsl:for-each>
```

Beispiel: Erzeuge eine Liste der Nachnamen

```

```

```
 <xsl:for-each select="person">
```

```

```

```
 <xsl:value-of select="lastname" />
```

```

```

```
 </xsl:for-each> ...
```

```

```

# Im Vergleich

- <xsl:for-each>

```
<xsl:template match="persons">

 <xsl:for-each select="person">
 <xsl:sort select="lastname" order="ascending"/>

 <xsl:value-of select="firstname"/>

 </xsl:for-each>

</xsl:template>
```

- <xsl:apply-templates />

```
<xsl:template match="persons">

 <xsl:apply-templates select="person" />

</xsl:template>
```

```
<xsl:template match="person">

 <xsl:value-of select="firstname"/>

</xsl:template>
```

# Bedingungen <xsl:if>

- Die Anweisung wird ausgeführt, wenn die Bedingung im @test-Attribut erfüllt ist
- Abfrage von Einzelfällen

```
<xsl:if test="not(@gender)">
 <xsl:text>Sehr geehrte Damen und Herren</xsl:text>
</xsl:if>
```



# Bedingungen <xsl:choose>

- Unterscheidung mehrerer Fälle

```
<xsl:choose>
 <xsl:when test="@gender='male'">
 <xsl:text>Sehr geehrter Herr </xsl:text>
 <xsl:value-of select="lastname" />
 </xsl:when>
 <xsl:when test="@gender='female'">
 <xsl:text>Sehr geehrte Frau </xsl:text>
 <xsl:value-of select="lastname" />
 </xsl:when>
 <xsl:otherwise><xsl:text>Sehr geehrte Damen und
Herren</xsl:text></xsl:otherwise>
</xsl:choose>
```

# Sortierreihenfolge

- Sortiert die Knotenmenge, die über das @select-Attribut ausgewählt wurde, nach bestimmten Kriterien
- Erstes Kindelement von <xsl:for-each> oder <xsl:apply-templates>
- Wird als leeres Element notiert
- Attribute:
  - data-type (text | number)
  - order (ascending | descending)
  - case-order (upper-first | lower-first)
  - lang (de | en | ...)

```
<xsl:apply-templates select="persons">
 <xsl:sort select="lastname" order="descending" />
</xsl:template>
```

# Strukturen wiederverwenden

- ```
<xsl:template match="persons">
  <html>
    <body>
      <table>
        <xsl:for-each select="person">
          <xsl:sort select="lastname" order="ascending"/>
          <xsl:call-template name="push-name"/>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template name="push-name">
  <tr>
    <td><xsl:value-of select="firstname" /></td>
    <td><xsl:value-of select="lastname" /></td>
  </tr>
</xsl:template>
```

Strukturen kopieren: <xsl:copy-of>

- Teile oder gesamtes XML-Dokument in das Ergebnis kopieren
- Z.B. um die Struktur eines XML-Dokuments zu verändern
- <xsl:copy-of> kopiert die über @select ausgewählten Elemente vollständig
- Inhalte, Attribute, und Kindelemente werden ebenfalls kopiert

XML-Input:

```
<p xml:id="para_1">  
  Das ist mein erster Absatz  
</p>
```

XSL-Stylesheet:

```
<xsl:template match="p[@xml:id='para_1']">  
  <xsl:copy-of select="." />  
</xsl:template>
```

XML-Ergebnis:

```
<p xml:id="para_1">  
  Das ist mein erster Absatz  
</p>
```

Strukturen kopieren: <xsl:copy>

- Nur der aktuelle Knoten wird in das Ergebnis kopiert
- Attribute, Inhalte und Knoten werden nicht kopiert

XML-Input:

```
<p xml:id="para_2">  
  Das ist mein zweiter Absatz  
</p>
```

XSL-Stylesheet:

```
<xsl:template match="p[@xml:id='para_2']">  
  <xsl:copy select="." />  
</xsl:template>
```

XML-Ergebnis:

```
<p>  
  
</p>
```

Identity Transform

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

```
<xsl:template match="vorname">
  <xsl:element name="forename">
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>
```

Variablen

- Variablen dienen als Kurzreferenz

- Variable erstellen:

```
<xsl:variable name="Variablenname" select="XPath-Ausdruck"/>
```

- Variable verwenden:

```
$Variablenname
```

Leerraum

- `<xsl:strip-space elements="list-of-elements">`
 - Definiert Elemente, bei denen Leerraum entfernt werden soll
 - `<xsl:strip-space elements="firstname lastname">`
 - FranzKafka
- `<xsl:preserve-space elements="list-of-elements">`
 - Definiert Elemente, bei denen Leerraum erhalten werden soll
 - `<xsl:preserve-space elements="firstname lastname">`
 - Franz Kafka
- Top-Level-Elemente – Kindelement von `<xsl:stylesheet>`
- Im Attribut `@elements` wird eine Liste der Elemente angegeben
- `elements="*"` gilt für alle Elemente

Mehrere Zieldokumente

- `<xsl:result-document href="string">`
 - Erstellen mehrerer Zieldokumente
 - In @href werden Pfad und Dateiname des Zieldokuments angegeben

```
<xsl:template match="Postkarte">
```

```
  <xsl:result-document href="front.xml">
```

Hier wird ein eigenes Dokument für die Vorderseite der Postkarte erstellt.

```
  </xsl:result-document>
```

```
  <xsl:result-document href="back.xml">
```

Hier wird ein eigenes Dokument für die Rückseite der Postkarte erstellt.

```
  </xsl:result-document>
```

```
</xsl:template>
```

Referenzen

- Kurzes W3C Tutorial:
https://www.w3schools.com/xml/xsl_intro.asp
- XSLT bei SelfHTML: <http://de.selfhtml.org/xml/darstellung>
- Kay, Michael: XSLT 2.0 programmer's reference. Wiley 2004.
- Tennison, Jeni: Beginning XSLT. Apress Media 2004.

Grundstruktur HTML5

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Titel des Dokuments</title>
```

```
    <meta charset="UTF-8" />
```

```
  </head>
```

```
  <body>
```

```
    Inhalt des Dokuments ...
```

```
  </body>
```

```
</html>
```

HTML Übersicht

- https://www.w3schools.com/html/html5_intro.asp

Hauptstrukturelemente

- `<html>` Wurzelement
- `<head>` Kopfdaten mit Metadateninformationen zum Dokument (zumindest `<title>`)
- `<body>` Abschnitt für den Inhalt (wird im Browserfenster angezeigt)

Kopfdaten

- `<title>` Dokumenttitel (wird in der Titelleiste des Browsers angezeigt)
- `<meta charset="UTF-8" />`
enthält die Zeichenkodierung
- `<link rel="stylesheet" href="project.css" />`
Verknüpft das Dokument mit einem CSS-Stylesheet

HTML Übersicht

Body-Elemente:

- Strukturelemente

- `<header>` Enthält Kopfzeileninformationen (Projekttitle, Logo, ...)
- `<section>` Sektion
- `<div>` Abschnitt (kann statt `<section>` verwendet werden, oder zur Definition eines Unterabschnittes von `<section>`)
- `<footer>` Enthält Fußzeileninformationen

- Blockelemente

- `<h1>`, `<h2>` ... `<h6>` Überschriften (Sechs Stufen)
- `<p>` Absatz
- `<blockquote>` Blockzitat
- `` Ungeordnete Liste
- `` Geordnete Liste
- `` Listenpunkt (innerhalb von `` oder ``)

HTML Übersicht

Weiter Body-Elemente und Attribute:

- Inzeilige Elemente

- `` Eine beliebige Textstelle (wird für Hervorhebungen, Styling, etc. verwendet)
- `<q>` Inzeiliges Kurzzitat
- `` Betonter Text (wird kursiv angezeigt)
- `` Stark betonter Text (wird fett angezeigt)
- `<code>` Computercode
- `` Hyperlink
- `` Abbildung einfügen

- HTML Attribute

- `id` identifiziert ein Element eindeutig, der Wert einer ID darf nur 1x im Dokument vorkommen
- `class` Um Elemente zu identifizieren (Wert darf mehrfach vorkommen) und CSS Eigenschaften darauf anzuwenden)
- `title` ToOLTIPtext, wird angezeigt, wenn man mit der Maus über das Element fährt

XML-DATENBANKEN

X-Technologien vertieft
Sommersemester 2019

Martina Scholger
Zentrum für Informationsmodellierung –
Austrian Centre for Digital Humanities

Was wissen wir

- HTML, CSS, JavaScript
 - Informative digitale Ressourcen in ansprechendem Design zur Verfügung stellen; Interaktion hineinbringen
- Wissenschaft und Forschung → digital scholarly editions
- XML & TEI
 - Zur Modellierung und Annotation der Quellen
- XPath & XSLT
 - Zur Abfrage der Dokumente und zur Generierung unterschiedlicher Ausgabeformate
- oXygen
 - Programm um XML-Dokumente zu schreiben und zu edieren

Was fehlt?

- Eine einfache Möglichkeit um Daten sammlungs-
übergreifend zu analysieren.
- Eine Suchmaschine und Datenbank um Inhalte
anzufagen
- Ein Webserver zur Publikation von XML/TEI Dokumenten

- XML Datenbank - **eXist**



- XML Framework – **GAMS** (Geisteswissenschaftliches
Asset Management System)



XML-Datenbanken

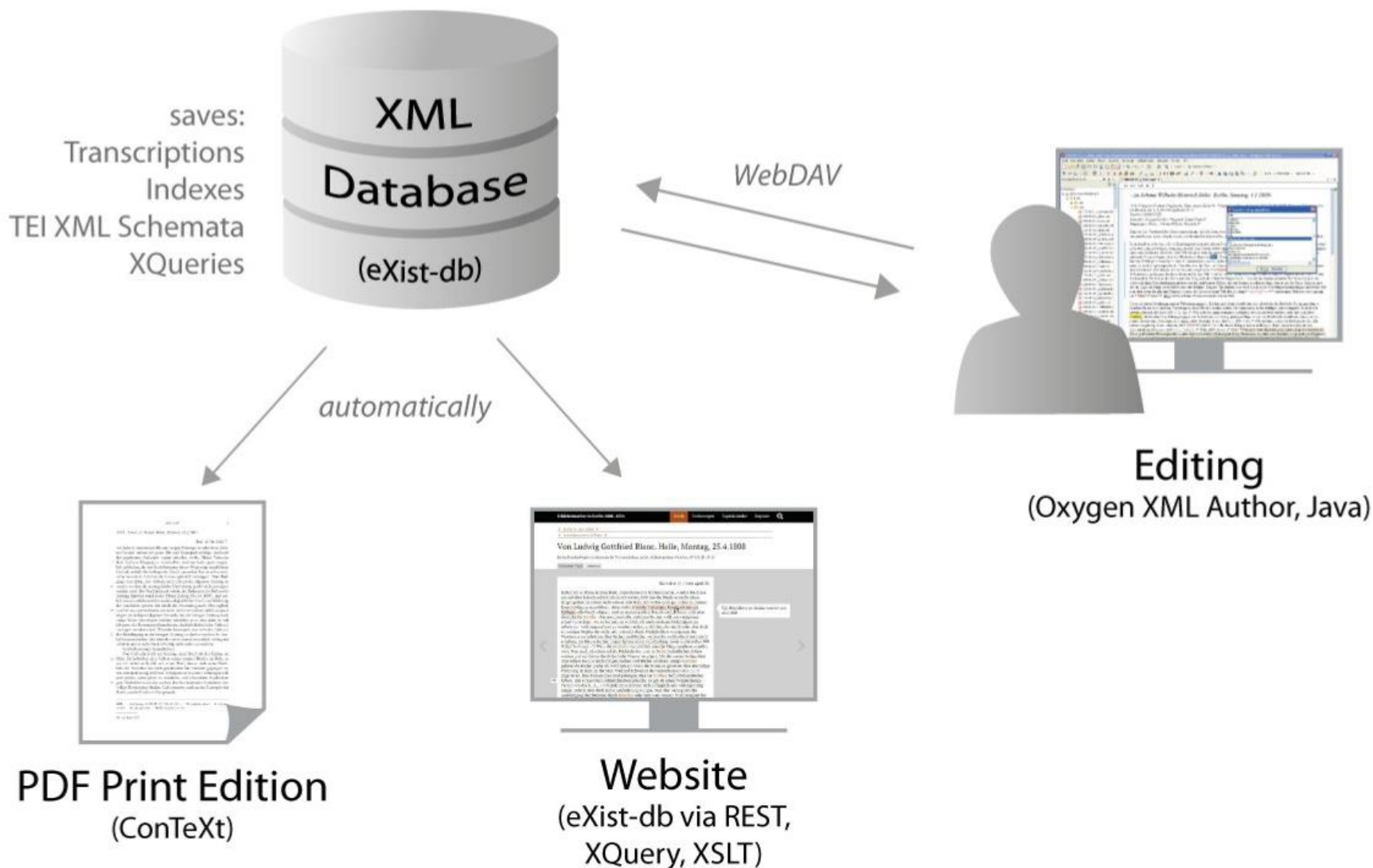
- Speicherung von XML-Daten
- Persistente Indizes
- Analyse von einem oder mehreren XML-Dokumenten, oder Fragementen daraus
- Einfache und effiziente Verarbeitung von XML Dokumenten mittels X-Technologien
- Verarbeitung von semi-strukturierten Informationen (vs. relationales Modell - Tabellen) = NoSQL Datenbank
- Native XML Datenbanken: https://de.wikipedia.org/wiki/XML-Datenbank#Native_XML-Datenbanken
- eXist
 - <http://exist-db.org/>
- BaseX
 - <http://basex.org/>

Ziel



- Einführung in eXist-db
- Basiskenntnisse in XQuery
 - W3C XML Query Language
- Entwicklung einer kleinen Webapplikation

- Native XML-Datenbank
- Entwicklung seit 2000 (Wolfgang Meier)
- Dokument im Zentrum – NoSQL document database
- Open Source Software (Java)
 - Verwenden
 - Beitragen
- Verwendung der Abfragesprache XQuery (W3C)
- Webserver (store, retrieve, update XML documents)
- Verschiedene Interfaces für den Datenzugriff: REST, Webdav etc.
- Ausführen von XQuery über Webanfrage



eXist vs. SQL and NoSQL Datenbanken

- eXist ist (nicht ausschließlich) dokumentorientiert
 - XML: mixed-content
 - Umgang mit Namespaces
 - Schemaless
 - Strukturierte Suche (unterschiedliche Indizes)
 - Formulare
 - Applikationsentwicklung in eXist
-
- SQL, MySQL, Oracle ist tabellenorientiert
 - JSON: datenorientiertes Dokumentenformat

Installation von eXist


- Dokumentation → Quick Start
 - <http://www.exist-db.org/exist/apps/doc/>
- Voraussetzung: zumindest JRE (Java runtime environment) oder JDK (Java Development Kit), mindestens Version 8
- Download aktuelle eXist-Version <http://www.exist-db.org>
- <http://localhost:8080>

Dashboard


Dashboard

localhost:8080/exist/apps/dashboard/index.html


Not logged in




Backup




Collections




eXide - XQuery IDE




Java Admin Client




Markdown Parser in XQuery




Monitoring and Profiling for eXist (Monex)




Package Manager




Shutdown



ThunDemoApp



Usermanager



Features

- Integrated development environment (IDE) → eXide
- Integrierte Prozessoren für XPath, XQuery, XSLT etc.
- Indizes (e.g. Lucene-basierter Volltextindex)
- Benutzer- und Rechtemanagement
- Applikationsmanagement (Package-Manager)
- Versionierung
- etc.

XQuery

- XML Query Language
 - <https://www.w3.org/TR/xquery/>
- Abfragesprache für XML-Daten
- W3C Empfehlung seit 2007
- XQuery baut auf XPath auf (selbes Datenmodell)
- Keine XML-Syntax
- Selektion und Extraktion von XML- Fragmenten, Konstruktion von neuen Elementen
- Turing complete
- Dateiendung **.xquery**, **.xq**, .xql .xqm, etc.

XQuery basics

- Literale (Zeichenketten wie “hello world”; numerische Werte wie 1)
- Variablen (\$foo), an die Werte gebunden werden
- Funktionen, built-in Funktionen wie substring-before('hello','l') oder eigene Funktionen
- Kommentare (: das ist ein Kommentar! :)
- Vergleiche: =, <, >, eq
- Bedingungen: if then else
- Deklarationen: declare namespace tei="http://www.tei-c.org/ns/1.0"
- FLWOR Expressions: das Herzstück von XQuery

XQuery - Datenmodell

- Knoten Element, Attribut, Textknoten, ...
- Atomarer Werte String, Zahl, ...
- Item Knoten oder atomarer Wert
- Sequenz Liste von Items
(geordnet, nicht hierarchisch)

XQuery - Sequenz

- ()
- (1, 2, 3)
- („Graz“, „Wien“, „Salzburg“, („Klagenfurt“, „Linz“))
- (<ort>
 <name>Graz</name>
</ort>,
<ort>
 <name>Wien</name>
</ort>)

XQuery – Sequenz

- Wie entstehen Sequenzen?
- Direkt konstruiert
 - (`<ort>...</ort>`, `<ort>...</ort>`)
- Als Ergebnis von Pfadausdrücken:
 - `doc("orte.xml")//placeName (<ort>...</ort>, <ort>...</ort>)`
- Als Rückgabe von Funktionen:
 - `tokenize(normalize-space(//text), "\s") ("Wir", "sind", "in", "Graz")`

XQuery – Ausdrücke

- =, !=, <, >, <=, >=
 - Vergleichsausdrücke
 - Vergleich von einzelnen Werten oder Sequenzen
 - 1 > 2 false
 - ("Sabine") = ("Sabine", "Anna") true

FLOWR Expressions

- FLWOR ist das Akronym für "For, Let, Where, Order by, Return"
- **for** – iteriert über jedes item einer Sequenz
- **let** – benennt eine Sequenz, bindet sie an eine Variable
- **where** – Filtert die Sequenz (optional)
- **order by** – sortiert die Sequenz (optional)
- **return** – gibt das Ergebnis zurück

Einfache FLOWR Expression

- In oXygen ausführen
- `for $item in (7, 3, 22, 13)`
 `order by $item`
 `return $item`
- Resultat: 3, 7, 13, 22

FLOWR Expression II

```
let $people := ('Katrin', 'Barbara', 'Bernhard')  
for $person in $people  
let $greeting := concat('Hallo ', $person, '!')  
return $greeting
```

eXist

- eXist starten
- Dashboard öffnen: <http://localhost:8080>
- Als admin einloggen
- Über den Collection Browser die Collection „tm“ und „dta“ anlegen
- XML/TEI Dateien in der Collection „dta“ ablegen
- Datei im Webbrowser über REST Interface anzeigen:
 - http://localhost:8080/exist/rest/tm/dta/abschatz_gedichte_1704.TEI-P5.xml

Anbindung an Oxygen

- oXygen > Optionen > Einstellungen
- Datenquellen
- Eine exist-db XML-Verbindung erstellen
- eXist-Benutzername und Kennwort eingeben
- oXygen neu starten (sicherstellen, dass eXist läuft)
- Im Datenquellen Explorer (Fenster > Datenquellen Explorer) wird die Verbindung angezeigt
- Offizielle Dokumentation (etwas veraltet):
<https://exist-db.org/exist/apps/doc/oxygen.xml>

Aufruf über REST Interface

- Über die REST-Schnittstelle (Aufruf im Browser) können...
 - Ressourcen angefordert werden (hier die XML-Datei):
 - http://localhost:8080/exist/rest/db/tm/dta/abschatz_gedichte_1704.TEIP5.xml
 - http://localhost:8080/exist/rest/db/tm/dta/abschatz_gedichte_1704.TEI-P5.xml?_query=declare namespace tei="http://www.tei-c.org/ns/1.0"; //tei:persName

Oxygen - > eXist

- Dateien in Oxygen bearbeiten und in eXist speichern
- Z.B. Einfache XSLT-Datei anlegen (Titel ausgeben)

```
<?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"
  xmlns:tei="http://www.tei-c.org/ns/1.0"
  exclude-result-prefixes="xs"
  version="2.0">

  <xsl:template match="/">
    <h1>
      <xsl:value-of select="//tei:title[@type='main' and not(@level)]"/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

- Datei speichern unter URL...
- Beim 1. Speichern Verbindung angeben
 - Folder-Icon anklicken
 - Server-URL: <http://localhost:8080/exist/webdav/db/>
 - Benutzer, Passwort angeben
 - Automatisch verbinden, Speichern anhängen
 - Verbinden und Datei in Collection speichern

XSLT ausführen

- XSLT über REST
 - http://localhost:8080/exist/rest/db/dta/abschatz_gedichte_1704.TEI-P5.xml?_xsl=dta.xsl
- XQuery unter Verwendung des extension module „transform“ (eXist-spezifische Erweiterung)

```
xquery version "3.1";  
declare default namespace "http://www.tei-c.org/ns/1.0";  
declare option exist:serialize "method=html media-type=text/html";
```

```
transform:transform(  
  doc("/db/tm/dta/abschatz_gedichte_1704.TEI-P5.xml"),  
  doc("/db/tm/dta/dta.xsl"),  
  ()  
)
```

FLOWR Expression

- XQuery-Test in eXide (eXist-interpreter Editor)
- Eval

```
xquery version "3.1";  
declare default namespace "http://www.tei-c.org/ns/1.0";  
for $title in collection('/db/tm/dta')//tei:title  
order by $title  
return $title
```


HTML Ausgabe

```
xquery version "3.1";  
declare default namespace "http://www.tei-c.org/ns/1.0";  
declare option exist:serialize "method=html media-type=text/html";
```

(: das ist ein Kommentar in eXist :)

```
let $hello := 'Hello World!'  
for $title in collection('/db/dta')//tei:title  
order by $title
```

```
return  
<html>  
  <head>  
    <title>{$hello}</title>  
  </head>  
  <body>  
    <ul>  
      <li>{$title}</li>  
    </ul>  
  </body>  
</html>
```

Directory Listing

```
xquery version "3.1";
```

```
for $resource in collection("/db/tm/dta")
```

```
return
```

```
    base-uri($resource)
```

Directory Listing (XML Struktur)

xquery version "3.1";

<texts>

{

for \$resource in collection("/db/tm/dta")

return

<text name="{base-uri(\$resource)}" />

}

</texts>

Directory Listing (XML Struktur)

```
xquery version "3.1";  
declare default namespace "http://www.tei-c.org/ns/1.0";
```

```
<texts>  
  {  
    for $resource in collection("/db/tm/dta")  
  
    return  
      <text name="{replace(base-uri($resource), '.+/(.+)$', '$1')}"  
  />  
    {  
      $resource//tei:title  
    }  
  }  
</texts>
```

Directory Listing (XML Struktur)

```
xquery version "3.1";  
declare default namespace "http://www.tei-c.org/ns/1.0";  
declare option exist:serialize "method=html media-type=text/html";
```

```
<texts>  
  {  
    let $data-collection := "/db/tm/dta"  
    for $resource in collection("$data-collection")  
    let $uri := base-uri($resource)  
  
    return  
      <text uri="{ $uri }" name="{replace({ $uri }, '.+/(.+)$', '$1')}" />  
      {  
        $resource//tei:title  
      }  
  }  
</texts>
```

Suche (Verwendung eines Volltextindex)

- Schritt 1: Suchformular erstellen
- Index erlaubt schnelles Auffinden über einen Index-Key
- Ermöglicht schnelles Suchen
- Zunächst muss das Bootstrap-Suchformular in der Datei listing.xq erweitert werden. Die blau markierten Teile sind zwingend für die Suchfunktion in eXist notwendig. Ein paar davon sind bereits in Bootstrap vorhanden, andere müssen ergänzt werden:

```
<form method="POST" action="search-index.xq" class="form-inline my-2 my-lg-0">  
  <input class="form-control mr-sm-2" type="text"  
placeholder="Search" aria-label="Search" name="searchphrase"  
size="30" />  
  <button class="btn btn-outline-success my-2 my-sm-0"  
type="submit">Search</button>  
</form>
```

Suche (Verwendung eines Volltextindex)

- `method="POST"`
 - Methode, um Daten/Anfrage zum Server zu senden
- `action="search-index.xq"`
 - Datei, die die Anfrage verarbeitet
- `type="text"`
- `name="searchphrase"`
 - Suchanfrage, die übergeben wird
- `size="30"`
 - Größe des Eingabefeldes, frei zu wählen
- `type="submit"`
 - Button oder Eingabefeld, das das Abschicken steuert

Suche (Verwendung eines Volltextindex)

- Schritt 2: Suchindex erstellen
- search-index.xq erstellen bzw. in eXist speichern
- Siehe die Erläuterungen in der Datei search-index.xq

Suche (Verwendung eines Volltextindex)

- Schritt 3: Index konfigurieren
- Der Index wird in einer „shadow“ Datenbankstruktur verwaltet
- Java Admin Client vom dashboard aus starten und einloggen
- In der Collection /db/system/config/db/ sollte bereits ein Konfigurationsfile collection.xconf liegen, das so aussieht:

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <triggers>
    <trigger
class="org.exist.extensions.exquery.restxq.impl.RestXqTrigger"/
>
  </triggers>
</collection>
```

Suche (Verwendung eines Volltextindex)

- Indexstruktur ergänzen. Im text-Element wird angegeben, auf welches Element (= qname=qualified name) die Suche gehen soll:

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <triggers>
    <trigger
class="org.exist.extensions.exquery.restxq.impl.RestXqTrigger"/>
  </triggers>
  <index xmlns:tei="http://www.tei-c.org/ns/1.0">
    <.lucene>
      <text qname="tei:l"/>
    </.lucene>
  </index>
</collection>
```

Suche (Verwendung eines Volltextindex)

- Danach muss noch die dta Collection zum Index hinzugefügt werden.
- Im Java Admin Client bleiben. Mit dem nach oben zeigenden Pfeil kann man zur übergeordneten Collection navigieren.
- An der obersten Ebene die Collection dta markieren und auf File > reindex collection gehen

Suche (Verwendung eines Volltextindex)

- Schritt 4: Suche testen, z.B.
- Herz
- Herz Diamanten
 - Alle Zeilen mit Herz oder Diamanten
- Herz AND Diamanten
 - Beide Worte müssen enthalten sein
- herz AND diaMANten
 - Suche ist case-insensitive

Literatur & Ressourcen

- ZIM-Bibliothek
 - Siegel, Erik; Retter, Adam (2014): eXist. A nosql document database and application platform.
 - Lehner, Wolfgang; Schöning, Harald (2004): XQuery. Grundlagen und fortgeschrittene Methoden.
 - Walmsley, Priscilla (2007): XQuery.
- Online-Ressourcen
 - eXist (<http://exist-db.org>)
 - McCreary, Dan: XQuery Wikibook (<https://en.wikibooks.org/wiki/XQuery>)

X-TECHNOLOGIEN VERTIEFT

X-Technologien vertieft
Sommersemester 2019

Martina Scholger
Zentrum für Informationsmodellierung –
Austrian Centre for Digital Humanities

Übung

- Geben Sie alle Orte in den Briefftexten aus:
 - `//body//p//settlement`
- Geben Sie alle Absendeorte der Briefe aus:
 - `//correspAction[@type='sent']/placeName`
- Geben Sie alle unterschiedlichen Absendeorte aus:
 - `distinct-values(//correspAction[@type='sent']/placeName)`
- Welche Ortsangaben starten mit ‚W‘?
 - `//settlement[starts-with(.,'W')]`
- In welchen Briefen (Nummer) hat Bearbeiter ‚SS‘ Änderungen vorgenommen?
 - `//change[@who='SS']/ancestor::TEI//publicationStmt/idno/@n`

Übung

- Wieviele Änderungen sind im Schnitt pro Brief festgehalten?
- `count(//revisionDesc/change) div count(//TEI)`
- Alle verschiedenen Personen?
- `distinct-values(//persName/@key)`
- Suchen Sie nach allen Orten, die den String 'burg' enthalten
- `//settlement[contains(., 'burg')]`
- Geben Sie alle Briefe aus dem Jahr 1811 aus.
- `//TEI[teiHeader//correspAction[contains(date,'1811')]]`
- Geben Sie alle Briefe aus, die nach 1811 gesendet wurden.

Numbers

Input-Argument: 291, 36, 473, 27

- `count((arg))`
 - Anzahl der Argumente
- `avg((arg))`
 - Durchschnittswert
- `max((arg))`
 - Maximalwert der Sequenz
- `min((arg))`
 - Minimalwert der Sequenz
- `sum((arg))`
 - Summe

`count(//sp/speaker/count(.))` oder aber viel kürzer `count(//speaker)`

Wertvergleich

Vergleich von einem Item mit exakt einem anderen Item

- **eq** equal to
- **ne** not equal to
- **gt** greater than
- **ge** greater than or equal to (not less than)
- **lt** less than
- **le** less than or equal to (not greater than)

Geben Sie alle Elemente aus, bei denen das @who-Attribut den Wert FH enthält

```
/*[@who eq 'FH']
```

Allgemeine Vergleichsoperatoren

Vergleich von mehreren Items auf beiden Seiten möglich

- `=` equal to
- `!=` not equal to
- `>` greater than (oder `>`;))
- `>=` greater than or equal to (nicht kleiner als; oder `>=`;))
- `<` less than (oder `<`;))
- `<=` less than or equal to (nicht größer als; oder `<=`;))

Geben Sie alle Elemente aus, bei denen das `@who`-Attribut den Wert `FH` oder `SS` enthält

```
/*[@who = ('FH', 'SS')]
```

```
/*[@who eq 'FH' or @who eq 'SS']
```

Funktionen

- Einfacher Einstieg
 - https://www.w3schools.com/xml/xpath_intro.asp
- XPath und XSLT Funktionen (w3schools):
 - https://www.w3schools.com/xml/xsl_functions.asp
- XPath Spezifikationen des (W3C):
 - https://www.w3.org/standards/techs/xpath#w3c_all
- Versionen:
 - XPath 1.0, XPath 2.0, XPath 3.0

XSLT

- Extensible Stylesheet Language
- W3C Standard seit 1999
- Sprache zur Transformation von XML Dokumenten
- XSL ist auch XML
- Eingabebaum (XML) – Transformationsbaum (XSLT) –
Ausgabebaum (HTML, XML, Text, etc.)

Hauptkomponenten von XSLT

- `<xsl:stylesheet>`
- `<xsl:output>`
- `<xsl:template>`
- `<xsl:apply-templates>`

<xsl:stylesheet>

<xsl:stylesheet

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
xmlns="http://www.w3.org/1999/xhtml"  
xpath-default-namespace="http://www.tei-c.org/ns/1.0"  
version="2.0">
```

- **xsl:stylesheet**: umschließendes Element, enthält das gesamte Stylesheet
- **xmlns:xsl**: verpflichtend anzugeben
- **xmlns**: Namensraum des Zieldokuments (xhtml)
- **xpath-default-namespace**:
 - optional anzugeben, wenn TEI-Dokumente transformiert werden
 - XPath nimmt an, dass sich Pfadausdrücke auf TEI-Dokument beziehen
 - Alternativ dazu wird der Namensraum des TEI-Dokuments angegeben
 - **xmlns:tei**="http://www.tei-c.org/ns/1.0"
 - XPath muss dann immer mit **tei:** beginnen
- **version**: verpflichtend anzugeben

<xsl:output>

```
<xsl:output method="xml" indent="yes" encoding="utf-8"/>
```

- **method:** Weitere Werte für “method” sind: xml, xhtml, text, html
- **indent:** wird auf “yes” gesetzt, um lange Zeilen umzuberechnen
- **encoding:** Zeichenkodierung immer auf utf-8 setzen

<xsl:template>

- Ein XSLT-Dokument besteht aus einer Reihe von Template-Regeln

```
<xsl:template match="div">
```

- Enthält Anweisungen, die bei einem “match” ausgeführt werden
- Das @match-Attribut enthält einen XPath-Ausdruck. D.h. ein Knoten (Element, Attribut, etc.) aus dem Eingabedokument wird selektiert und das Template darauf angewendet.

```
<xsl:template match="div">  
  <p><xsl:apply-templates/></p>  
</xsl:template>
```

- Prozessor trifft auf <div> im Eingabedokument (XML)
- Ein <p> im Ausgabedokument (HTML) wird erzeugt.
- Verarbeite Inhalt von <div> (XML)
- Gib das Resultat innerhalb von <p> (HTML) aus

<xsl:apply-templates>

- Leeres Element
- Anweisung, dass die Verarbeitung weiterer Templates fortgeführt werden soll
- Die zu verarbeitende Knotenmenge wird über einen Xpath-Ausdruck bestimmt
- Wie der Knoten verarbeitet wird, wird in der Templaterregel angegeben

```
<xsl:apply-templates select="[XPath]" />
```

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

```
<xsl:template match="div">  
  <xsl:apply-templates select="lg" />  
</xsl:template>
```

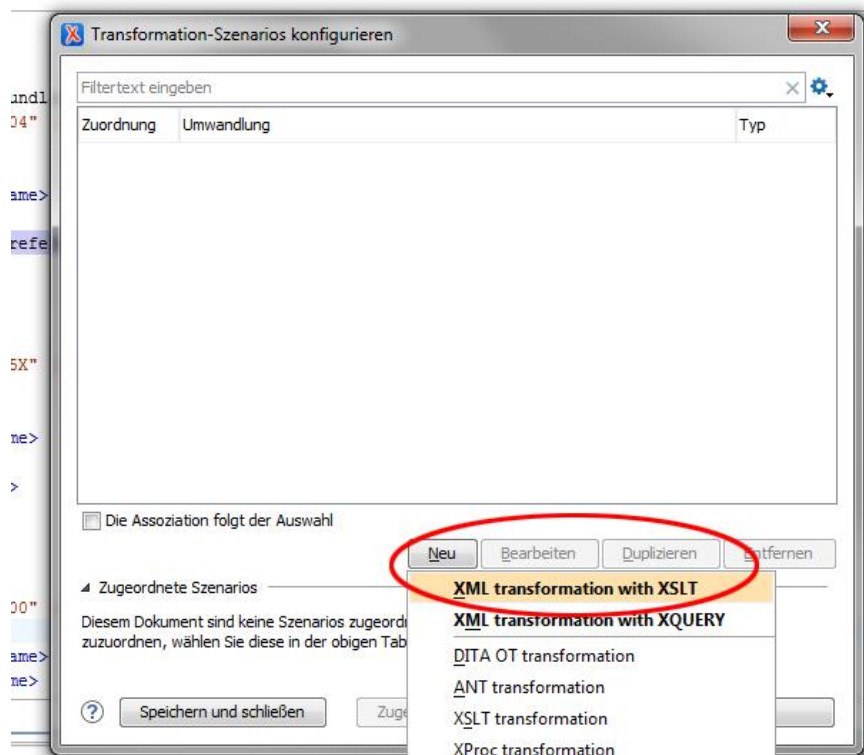
Zum Einstieg ein einfaches Beispiel

- Öffnen Sie die Datei **verse.xml** in Oxygen
- Erstellen Sie zu dieser Datei ein Stylesheet mit dem Namen **verse.xsl**
- Wechseln Sie zum XSLT Debugger
- Ausgabe der XML-Elemente in HTML:
 - XML `<div>` in HTML `<section>`
 - XML `<head>` in HTML `<h1>`
 - XML `<lg>` in HTML `<p>`
 - XML `<l>` in HTML durch `
` voneinander getrennt
 - XML `<byline>` in HTML `<p>`
- Die `<byline>` soll rechtsbündig ausgegeben werden, ergänzen Sie das Element um eine Klasse und fügen Sie ein CSS Dokument hinzu.

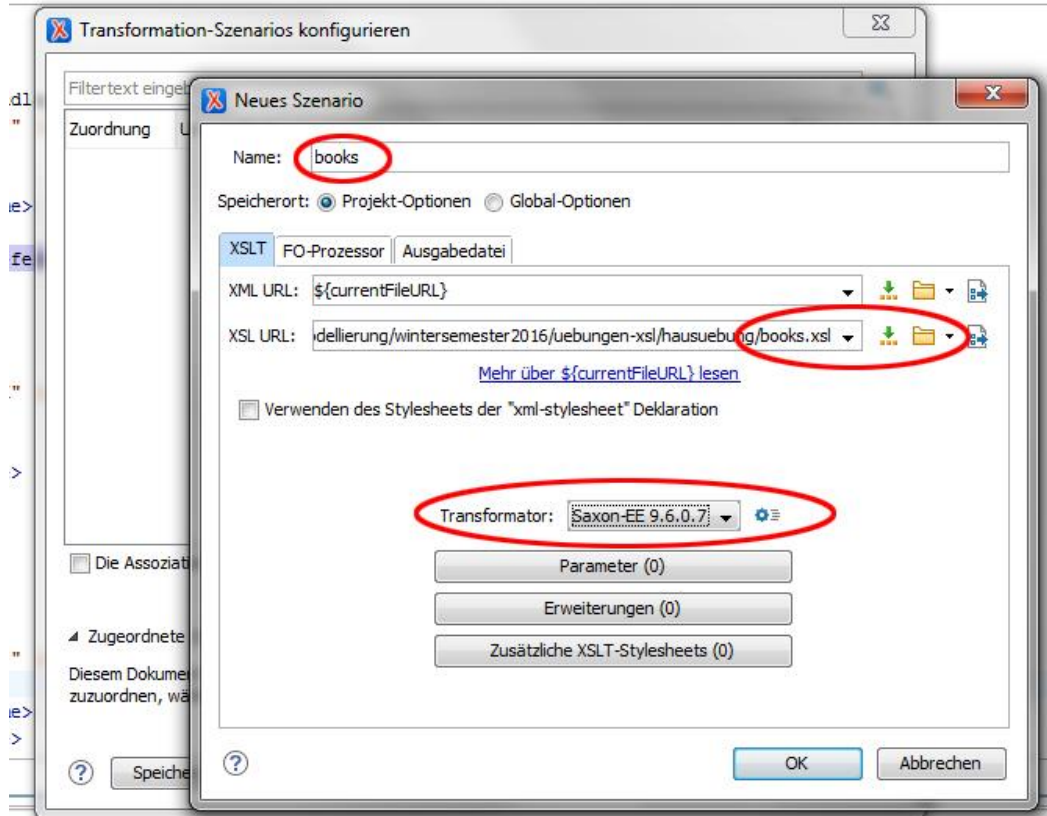
Transformationsszenario einrichten



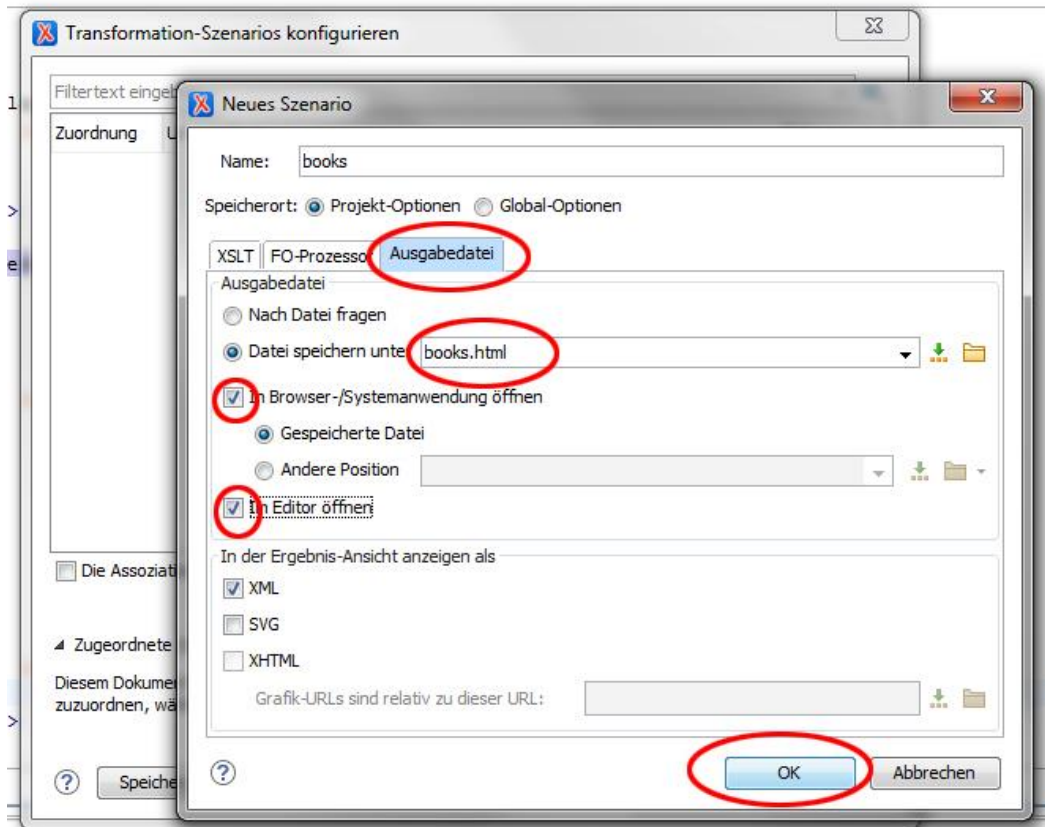
Transformation-Szenarios konfigurieren (Strg+Umschalt+C)



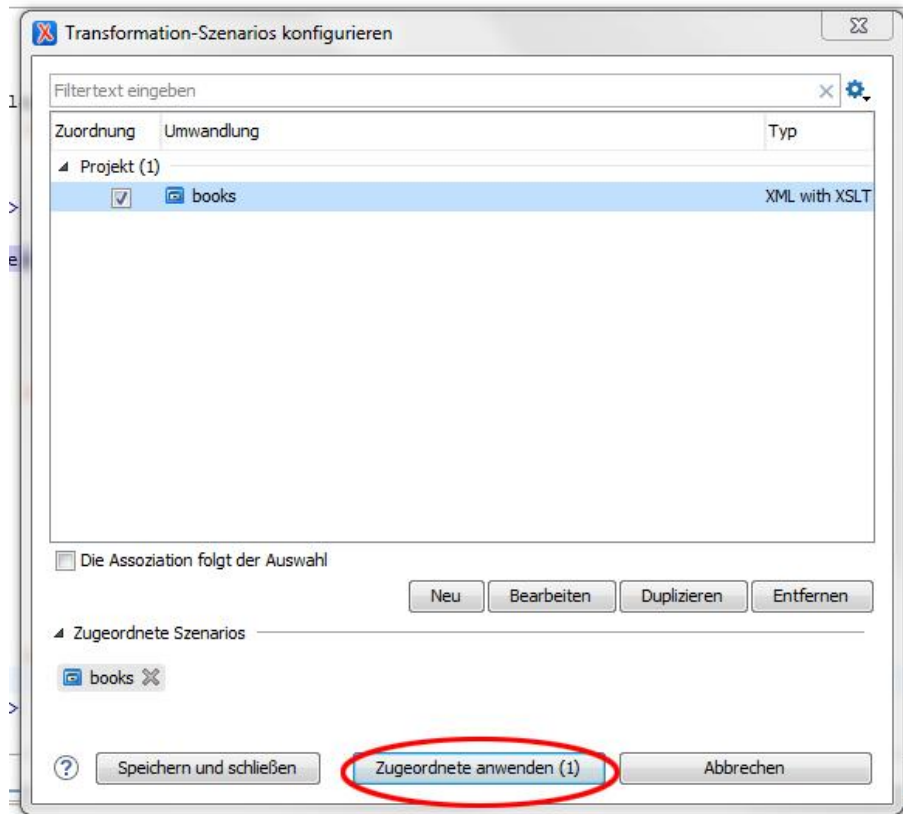
Transformationsszenario einrichten



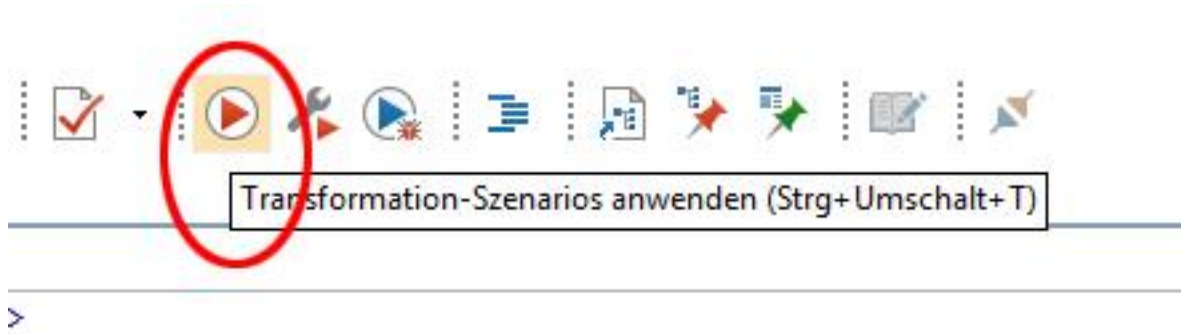
Transformationsszenario einrichten



Transformationsszenario einrichten



Transformationsszenario einrichten



TEI to HTML

- ```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/1999/xhtml"
 xpath-default-namespace="http://www.tei-c.org/ns/1.0"
 version="2.0">
 <xsl:output method="xml" indent="yes"/>
 <xsl:template match="/">
 <html>
 <head>
 <title>Titel ausgeben</title>
 </head>
 <body>
 <xsl:apply-templates/>
 </body>
 </html>
 </xsl:template>

 <!-- weitere Templaterregeln -->

</xsl:stylesheet>
```

# XSLT Processing Model

- Wo starten wir?
  - Beim Dokumentknoten (der „Vorgänger“ des Wurzelements)
- Was passiert, wenn kein passendes Regelwerk gefunden wird?
  - Die Defaultregeln (Built-in) kommen zum Tragen
- Was passiert, wenn genau eine Regel zutrifft?
  - Sie wird angewendet
- Was passiert, wenn mehr als eine Regel zutrifft?
  - Templates werden überschrieben
  - Built-in priority
  - User-specified priority

# Built-in Regeln

- Element: verarbeite die Kindknoten (Elemente, Textknoten), wende Templaterregeln an (build-in oder spezifische Regeln)
- Attribute: kein Output
- Text: der Text wird ausgegeben

# Built-in priority

```
<xsl:template match="div">
 <act>
 <xsl:apply-templates/>
 </act>
</xsl:template>
```

```
<xsl:template match="div/div">
 <scene>
 <xsl:apply-templates/>
 </scene>
</xsl:template>
```

# User-specified priority

```
<xsl:template match="div" priority="10">
 <act>
 <xsl:apply-templates/>
 </act>
</xsl:template>
```



# Zwei Extremfälle

Eine leere Regel für das Wurzelement:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="3.0">

 <xsl:template match="/">

 </xsl:template>

</xsl:stylesheet>
```

# Zwei Extremfälle

Eine leeres Regelwerk:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="3.0">

</xsl:stylesheet>
```



## <xsl:value-of />

- Schreibt den Wert (Text) des selektierten Knotens in das Ergebnisdokument.
- Der Text der Kindelemente wird zwar ebenfalls berücksichtigt, kann aber nicht mehr weiterverarbeitet werden

```
<h1><xsl:value-of select="head" /></h1>
```

Ergebnis:

```
<h1>London</h1>
```

# Elemente und Attribute

`<xsl:element name="[elementname]">`

- Konstruiert ein Element im Ergebnisbaum
- Elemente können auch direkt in das Ergebnis geschrieben werden:  
`<elementname>Text</Elementname>`

`<xsl:attribute name="[attributname]">`

- Konstruiert ein Attribut im Ergebnisbaum
- Attribute können auch direkt in das Ergebnis geschrieben werden:  
`<elementname attributname="attributwert">Text</Elementname>`
- Dynamische Wertausgabe im Attribut:  
`<elementname attributname="{XPATH}">Text</Elementname>`

`<xsl:text>`

- Schreibt Zeichendaten in das Zieldokument, z.B. Leerzeichen, Beistriche, sonstigen Text: `<xsl:text>`, `</xsl:text>`

# Attribute

- XML Input:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
 <person gender="male">
 <firstname>William</firstname>
 <lastname>Blake</lastname>
 </person>
 <person gender="female">
 <firstname>Agatha</firstname>
 <lastname>Christie</lastname>
 </person>
</persons>
```

- XSL Transformation (langform):

```
<xsl:template match="person">

 <xsl:attribute name="class">
 <xsl:value-of select="@gender" />
 </xsl:attribute>
 <xsl:value-of select="lastname" />

</xsl:template>
```

- XSL Transformation (kurzform):

```
<xsl:template match="person">
 <li class="{@gender}">
 <xsl:value-of select="lastname" />

</xsl:template>
```

# Attribute

- HTML Ergebnis

```
<li class="male">Blake
```

```
<li class="female">Christie
```

- CSS

```
.male {
 background-color: yellow;
 font-weight: bold;
}
```

```
.female {
 background-color: green;
 color: white;
 font-style: italic;
}
```

# Schleifen mit <xsl:for-each>

```
<xsl:for-each select="XPath">
```

```
...
```

```
</xsl:for-each>
```

Beispiel: Erzeuge eine Liste der Nachnamen

```

```

```
 <xsl:for-each select="person">
```

```

```

```
 <xsl:value-of select="lastname" />
```

```

```

```
 </xsl:for-each> ...
```

```

```

# Im Vergleich push & pull

```
<xsl:apply-templates />
```

```
<xsl:template match="persons">

 <xsl:apply-templates select="person" />

</xsl:template>
```

```
<xsl:template match="person">

 <xsl:value-of select="firstname" />

</xsl:template>
```

```
<xsl:for-each>
```

```
<xsl:template match="persons">

 <xsl:for-each select="person">
 <xsl:sort select="lastname" order="ascending" />

 <xsl:value-of select="firstname" />

 </xsl:for-each>

</xsl:template>
```

# Bedingungen <xsl:if>

- Die Anweisung wird ausgeführt, wenn die Bedingung im @test-Attribut erfüllt ist
- Abfrage von Einzelfällen

```
<xsl:if test="not(@gender)">
 <xsl:text>Sehr geehrte Damen und Herren</xsl:text>
</xsl:if>
```

# Bedingungen <xsl:choose>

- Unterscheidung mehrerer Fälle

```
<xsl:choose>
 <xsl:when test="@gender='male'">
 <xsl:text>Sehr geehrter Herr </xsl:text>
 <xsl:value-of select="lastname" />
 </xsl:when>
 <xsl:when test="@gender='female'">
 <xsl:text>Sehr geehrte Frau </xsl:text>
 <xsl:value-of select="lastname" />
 </xsl:when>
 <xsl:otherwise><xsl:text>Sehr geehrte Damen und
Herren</xsl:text></xsl:otherwise>
</xsl:choose>
```



# Sortierreihenfolge

- Sortiert die Knotenmenge, die über das @select-Attribut ausgewählt wurde, nach bestimmten Kriterien
- Erstes Kindelement von <xsl:for-each> oder <xsl:apply-templates>
- Wird als leeres Element notiert
- Attribute:
  - data-type (text | number)
  - order (ascending | descending)
  - case-order (upper-first | lower-first)
  - lang (de | en | ...)

```
<xsl:apply-templates select="persons">
 <xsl:sort select="lastname" order="descending" />
</xsl:template>
```

# Strukturen wiederverwenden

- ```
<xsl:template match="persons">
  <html>
    <body>
      <table>
        <xsl:for-each select="person">
          <xsl:sort select="lastname" order="ascending"/>
          <xsl:call-template name="push-name"/>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template name="push-name">
  <tr>
    <td><xsl:value-of select="firstname" /></td>
    <td><xsl:value-of select="lastname" /></td>
  </tr>
</xsl:template>
```

Strukturen kopieren: <xsl:copy-of>

- Teile oder gesamtes XML-Dokument in das Ergebnis kopieren
- Z.B. um die Struktur eines XML-Dokuments zu verändern
- <xsl:copy-of> kopiert die über @select ausgewählten Elemente vollständig
- Inhalte, Attribute, und Kindelemente werden ebenfalls kopiert

XML-Input:

```
<p xml:id="para_1">  
  Das ist mein erster Absatz  
</p>
```

XSL-Stylesheet:

```
<xsl:template match="p[@xml:id='para_1']">  
  <xsl:copy-of select="." />  
</xsl:template>
```

XML-Ergebnis:

```
<p xml:id="para_1">  
  Das ist mein erster Absatz  
</p>
```

Strukturen kopieren: <xsl:copy>

- Nur der aktuelle Knoten wird in das Ergebnis kopiert
- Attribute, Inhalte und Knoten werden nicht kopiert

XML-Input:

```
<p xml:id="para_2">  
  Das ist mein zweiter Absatz  
</p>
```

XSL-Stylesheet:

```
<xsl:template match="p[@xml:id='para_2']">  
  <xsl:copy select="." />  
</xsl:template>
```

XML-Ergebnis:

```
<p>  
  
</p>
```

Identity Transform

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

```
<xsl:template match="vorname">
  <xsl:element name="forename">
    <xsl:apply-templates />
  </xsl:element>
</xsl:template>
```

Identity Transform in Version 3

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

```
<xsl:template match="vorname">  
  <xsl:element name="forename">  
    <xsl:apply-templates />  
  </xsl:element>  
</xsl:template>
```

Variablen

- Variablen dienen als Kurzreferenz

- Variable erstellen:

```
<xsl:variable name="Variablenname" select="XPath-Ausdruck"/>
```

- Variable verwenden:

```
$Variablenname
```

Leerraum

- `<xsl:strip-space elements="list-of-elements">`
 - Definiert Elemente, bei denen Leerraum entfernt werden soll
 - `<xsl:strip-space elements="firstname lastname">`
 - FranzKafka
- `<xsl:preserve-space elements="list-of-elements">`
 - Definiert Elemente, bei denen Leerraum erhalten werden soll
 - `<xsl:preserve-space elements="firstname lastname">`
 - Franz Kafka
- Top-Level-Elemente – Kindelement von `<xsl:stylesheet>`
- Im Attribut `@elements` wird eine Liste der Elemente angegeben
- `elements="*"` gilt für alle Elemente

collection()

- `<xsl:variable name="briefe"
select="collection('./?select=l_*.xml')"/>`

Mehrere Zieldokumente

- `<xsl:result-document href="string">`
 - Erstellen mehrerer Zieldokumente
 - In @href werden Pfad und Dateiname des Zieldokuments angegeben

```
<xsl:template match="Postkarte">
```

```
  <xsl:result-document href="front.xml">
```

Hier wird ein eigenes Dokument für die Vorderseite der Postkarte erstellt.

```
  </xsl:result-document>
```

```
  <xsl:result-document href="back.xml">
```

Hier wird ein eigenes Dokument für die Rückseite der Postkarte erstellt.

```
  </xsl:result-document>
```

```
</xsl:template>
```

Referenzen

- Kurzes W3C Tutorial:
https://www.w3schools.com/xml/xsl_intro.asp
- XSLT bei SelfHTML: <http://de.selfhtml.org/xml/darstellung>
- Kay, Michael: XSLT 2.0 programmer's reference. Wiley 2004.
- Tennison, Jeni: Beginning XSLT. Apress Media 2004.

Grundstruktur HTML5

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Titel des Dokuments</title>
```

```
    <meta charset="UTF-8" />
```

```
  </head>
```

```
  <body>
```

```
    Inhalt des Dokuments ...
```

```
  </body>
```

```
</html>
```

HTML Übersicht

- https://www.w3schools.com/html/html5_intro.asp

Hauptstrukturelemente

- `<html>` Wurzelement
- `<head>` Kopfdaten mit Metadateninformationen zum Dokument (zumindest `<title>`)
- `<body>` Abschnitt für den Inhalt (wird im Browserfenster angezeigt)

Kopfdaten

- `<title>` Dokumenttitel (wird in der Titelleiste des Browsers angezeigt)
- `<meta charset="UTF-8" />`
enthält die Zeichenkodierung
- `<link rel="stylesheet" href="project.css" />`
Verknüpft das Dokument mit einem CSS-Stylesheet

HTML Übersicht

Body-Elemente:

- Strukturelemente

- `<header>` Enthält Kopfzeileninformationen (Projekttitle, Logo, ...)
- `<section>` Sektion
- `<div>` Abschnitt (kann statt `<section>` verwendet werden, oder zur Definition eines Unterabschnittes von `<section>`)
- `<footer>` Enthält Fußzeileninformationen

- Blockelemente

- `<h1>`, `<h2>` ... `<h6>` Überschriften (Sechs Stufen)
- `<p>` Absatz
- `<blockquote>` Blockzitat
- `` Ungeordnete Liste
- `` Geordnete Liste
- `` Listenpunkt (innerhalb von `` oder ``)

HTML Übersicht

Weiter Body-Elemente und Attribute:

- Inzeilige Elemente

- `` Eine beliebige Textstelle (wird für Hervorhebungen, Styling, etc. verwendet)
- `<q>` Inzeiliges Kurzzitat
- `` Betonter Text (wird kursiv angezeigt)
- `` Stark betonter Text (wird fett angezeigt)
- `<code>` Computercode
- `` Hyperlink
- `` Abbildung einfügen

- HTML Attribute

- `id` identifiziert ein Element eindeutig, der Wert einer ID darf nur 1x im Dokument vorkommen
- `class` Um Elemente zu identifizieren (Wert darf mehrfach vorkommen) und CSS Eigenschaften darauf anzuwenden)
- `title` ToOLTIPtext, wird angezeigt, wenn man mit der Maus über das Element fährt