XSL-FO – Extensible Stylesheet Language Formatting Objects

Mit XSL-FO können aus XML-Daten Druckvorlagen erstellt werden. Die Formatierungsanweisungen sind in ein XSLT-Stylesheet integriert. Der Transformationsprozess ist zweistufig: Zunächst wird ein FO-Baum erstellt, aus dem dann wiederum die Zieldatei generiert wird.

Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/
1999/XSL/Transform" version="2.0">
 <xsl:template match="/">
❶ <fo:root xmlns:fo="http://www.w3.org/1999/XSL/
Format">
 ❷ <fo:layout-master-set>
  ❸ <fo:simple-page-master master-name="str"
      page-height="29.7cm" page-width="21.0cm">
   ❹ <fo:region-body margin-top="1.5cm"
      margin-bottom="1.5cm" margin-left="2cm"
      margin-right="2cm" region-name="main"/>
   ❺ <fo:region-before extent="3cm"
                       region-name="oben"/>
   ❻ <fo:region-after extent="1.5cm"
                      region-name="unten"/>
    </fo:simple-page-master>
  ❼ <fo:page-sequence-master master-name="ged">
      <fo:repeatable-page-master-reference
         master-reference="str"/>
    </fo:page-sequence-master>
   </fo:layout-master-set>
 ❽ <fo:page-sequence master-reference="ged">
  ❾ <fo:static-content flow-name="unten">
      <fo:block space-before="5pt"
                text-align="right">
         <fo:page-number/>
      </fo:block>
    </fo:static-content>
  ❿ <fo:flow flow-name="main" font-family="Arial">
    ⓫ <fo:block font-size="16pt">
      ⓬ <xsl:apply-templates/>
      </fo:block>
    </fo:flow>
   </fo:page-sequence>
  </fo:root>
 </xsl:template>
</xsl:stylesheet>
```

❶ Beginn des Formatierungsbaums

❷ Definition von Seitenvorlagen: unterschiedliche Seitengestaltung für verschiedene Seiten wie Titelseite, Textseite rechts, Textseite links, etc.

❸ Hier wird das Format für eine Seitenart (hier: `str`) definiert

❹ `fo:region-body` definiert den Hauptbereich der Seite

❺ `fo:region-before` definiert den Kopfzeilenbereich, z. B. für Kolumnentitel

❻ `fo:region-after` definiert den Fußzeilenbereich, z. B. für Fußnoten

❼ Hier wird die Reihenfolge der unterschiedlich definierten Seitenformate festgelegt (falls es unterschiedliche gibt)

❽ Hier wird die Ausgabe der Seiteninhalte beschrieben; Verknüpfung mit der Vorlage `ged`

❾ Ausgabe statischen Inhalts, hier ein Bereich mit Seitenzahl

❿ Ausgabe laufender Inhalte

⓫ Definition eines Block-Bereiches (rechteckiger Anzeigebereich), für dessen Inhalte eine Schriftgröße festgelegt wird

⓬ An dieser Stelle werden weitere XSLT-Templates für die Ausgabe von Inhalten berücksichtigt

Regeln

- Ein XSL-FO-Dokument besteht aus einem Bereich, in dem Seitenvorlagen definiert werden („layout master set“) und einem oder mehreren Bereichen, in denen die auszugebenden Seiteninhalte beschrieben werden („page sequences“)
- Im Master werden auch Seitenbereiche („regions“) definiert, von denen mindestens der Hauptbereich vorhanden sein muss:

`region-body`	Hauptbereich
`region-before`	Kopfzeilenbereich
`region-after`	Fußzeilenbereich
`region-start`	linker Seitenbereich
`region-end`	rechter Seitenbereich

- Eine „page sequence“ wird mit einer bestimmten Vorlage verknüpft (über das Attribut `master-reference`) und kann folgende Elemente enthalten:

`static-content`	zur Ausgabe statischer Inhalte
`flow`	zur Ausgabe laufender Inhalte

- Innerhalb von `fo:static-content` und `fo:flow` können Blockelemente (Absätze, Tabellen, Listen, ...) definiert werden; diese wiederum können Inline-Elemente und XSLT-Anweisungen enthalten

2015 – Autorin, Satz: Ulrike Henny; Gestaltung: Markus Schnöpf

Institut für Dokumentologie und Editorik e.V.
c/o Cologne Center for eHumanities (CCeH)
Universität zu Köln
Universitätsstraße 22
50923 Köln

i|d|e

XML

Kurzreferenz für Fortgeschrittene I

(mit XMLSchema, XSLT und XSL-FO)

Institut für Dokumentologie und Editorik

www.i-d-e.de
info@i-d-e.de

XML Schema

XML Schema ist ein Beispiel für eine Schemasprache, mit der die Struktur und Teile des Inhalts eines XML-Dokumentdokumenttypen definiert werden. Durch die Anwendung eines Schemadokuments auf ein XML-Dokument wird überprüft, ob dieses den Vorgaben entspricht.

Beispiel – Schema

```
<?xml version="1.0" encoding="UTF-8"?>
❶<xs:schema xmlns:xs="http://www.w3.org/2001/
 XMLSchema" xmlns:ged="http://ged.de/nr"
   targetNamespace="http://ged.de/nr"
 ❷ elementFormDefault="qualified">
❸ <xs:element name="gedicht">
   <xs:complexType>
  ❹ <xs:sequence>
    ❺ <xs:element name="titel" type="ged:titel"/>
    ❻ <xs:element name="autor" type="xs:string"
                               default="anonym"/>
    ❼ <xs:element name="strophe"
                  maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
❽ <xs:complexType name="titel">
    <xs:simpleContent>
     <xs:extension base="xs:string">
    ❾ <xs:attribute name="lang" use="required">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="de"/>
         <xs:enumeration value="en"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
  </xs:schema>
```

❶ Wurzelelement des Schemas mit Namensraumdeklarationen (XML-Schema: `xs`, Gedicht: `ged`) und der Angabe des Zielnamensraums (`targetNamespace`)

❷ Es wird festgelegt, dass die Elemente in einem Dokument, das diesem Schema genügen soll, explizit dem Zielnamensraum zugeordnet sein müssen

❸ Deklaration eines „komplexen“ Elements `gedicht`, das weitere Elemente enthält

❹ Die Kindelemente von `gedicht` müssen in der angegebenen Reihenfolge vorkommen

❺ Ein Element mit dem Namen `titel`, das einen in diesem Schema selbst definierten Datentyp hat

❻ Ein „einfaches“ Element mit dem Namen `autor`, das nur Zeichendaten (Typ: `xs:string`) enthält. Zusätzlich ist ein Standardwert angegeben, der verwendet wird, falls das Element im Dokument leer ist

❼ Ein „einfaches“ Element `strophe`, das beliebig oft vorkommen kann

❽ Definition eines „komplexen“ Datentypen mit dem Namen `titel`. Definiert wird ein Element mit einfachem (`simpleContent`) Textinhalt (`base="xs:string"`). Komplex ist es, weil es ein Attribut `lang` hat

❾ Definition eines obligatorischen Attributs mit dem Namen `lang`, für das genaue Werte festgelegt sind

Beispiel – Zuweisung

```
<?xml version="1.0" encoding="UTF-8"?>
❶<gedicht ❷xmlns="http://ged.de/nr"
❸xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"
❹xsi:schemaLocation="http://ged.de/nr gedicht.xsd">
```

❶ Wurzelelement der XML-Datei, dem das Schema zugewiesen werden soll

❷ Angabe des Default-Namensraums der XML-Datei

❸ Deklaration des XSI-Namensraums (XML Schema-Instanz)

❹ Zuweisung des Schemas: Angabe des Namensraums, für den das Schema gilt (`http://ged.de/nr`) und des Ortes, an dem es liegt (hier der Dateiname: `gedicht.xsd`)

Regeln für XML Schema

- Einfache Elemente („simple elements“) enthalten nur Text, komplexe Elemente („complex elements“) können wiederum Elemente enthalten und/oder Attribute haben. Attribute sind immer einfach („simple types“)
- Das Vorkommen der Elemente kann durch Indikatoren für Reihenfolge (Elemente `xs:all`, `xs:choice`, `xs:sequence`) und Häufigkeit (Attribute `minOccurs`, `maxOccurs`) genauer festgelegt werden
- Elemente kommen standardmäßig genau einmal vor, Attribute sind standardmäßig optional
- Es gibt Standarddatentypen (`xs:string`, `xs:integer`, `xs:boolean`, `xs:date`, etc.) und selbst definierte Datentypen
- Für Werte/Inhalte können Beschränkungen („restrictions“, „data facets“), Erweiterungen („extensions“) und Muster („data patters“ = reguläre Ausdrücke) definiert werden

XSLT für Fortgeschrittene

Gruppieren

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

– gruppiert Elemente einer Grundmenge anhand eines Schlüssels und führt für jede Gruppe Anweisungen aus

Beispiel

```
❶ ...<xsl:for-each-group
  select="collection("gedichte")//gedicht"
  group-by="substring(titel,1,1)">
❷ <xsl:sort select="current-grouping-key()"/>
❸ <xsl:value-of select="current-grouping-key()"/>
❹ <br />
❺ <xsl:for-each select="current-group()">
 ❻ <xsl:sort select="titel"/>
  ❼ <h2>
    ❽ <xsl:value-of select="titel"/>
    </h2>
  ❾ <xsl:apply-templates select="strophe"/>
  </xsl:for-each>
 </xsl:for-each-group>...
```

❶ Gruppiere alle Gedichte (Elemente `gedicht`) nach dem Anfangsbuchstaben ihres Titels (1. Zeichen des Elements `titel`) und tue für jede Gruppe von Gedichten etwas

❷ Sortiere jede Gruppe nach dem Gruppierungsschlüssel (hier: dem Anfangsbuchstaben des Titels)

❸ Gib den aktuellen Gruppierungsschlüssel aus

❹ Es wird etwas in das Ergebnisdokument geschrieben (hier: ein Zeilenumbruch)

❺ Für jedes Element in der aktuellen Gruppe (jedes Gedicht in der Gruppe) soll etwas getan werden

❻ Die Elemente in der aktuellen Gruppe werden sortiert

❼ Eine HTML-Überschrift wird in das Zieldokument geschrieben

❽ Der Titel des aktuellen Gedichts in der aktuellen Gruppe wird ausgegeben

❾ Es werden weitere Templates berücksichtigt, die auf Elemente `strophe` matchen

Kopieren

`<xsl:copy>`

– erzeugt eine Kopie des Kontextknotens (ohne Kindknoten und Attribute)

`<xsl:copy-of select="XPath-Ausdruck">`

– erzeugt eine Kopie des kompletten Teilbaumes (mit Kindknoten und Attributen)