181135 VU Semistrukturierte Daten 1

XSL - Extensible Stylesheet Language

(Teil 2)

27.10.2005

Reinhard Pichler

Inhalt

- Stylesheets
- XSLT
  - Aufbau eines XSLT Stylesheets
  - Abarbeitung eines XSLT Stylesheets
  - Erzeugung des Result tree
  - **Kontrollstrukturen: for-each, if, choose**
  - **Sortieren, Nummerieren**
  - **Variablen, Parameter**
  - **zusätzliche Funktionen (gegenüber XPath)**
- **XSLFO**

xsl:for-each

- xsl:for-each:
  - Schleife über eine Knotenmenge, die mittels Expression des select-Attributs bestimmt wird.
  - current node und current node list innerhalb der Schleife entsprechen dieser selektierten Knotenmenge
  - Content von xsl:for-each: Anweisungen der Schleife
- Beispiel:

```
<xsl:template match="lehre">
  <xsl:value-of select="veranstaltung[1]/titel"/>
  <xsl:for-each select="veranstaltung[position() &gt; 1]">
   <xsl:text>, </xsl:text>
   <xsl:value-of select="titel" />
  </xsl:for-each>
</xsl:template>
```

-> Liste aller Lehrveranstaltungsnamen

for-each vs. apply-templates

- Üblicherweise austauschbar, z.B.:

```
<xsl:template match="lehre">
  <xsl:value-of select="veranstaltung[1]/titel"/>
  <xsl:apply-templates select="veranstaltung[position() &gt; 1]"/>
</xsl:template>

<xsl:template match="veranstaltung[position()&gt;1]">
    <xsl:text>, </xsl:text>
    <xsl:value-of select="titel" />
</xsl:template>
```

- Vorteile von for-each:
  - Einfache Formulierung von Joins, z.B.: mit document() Funktion (zur Verknüpfung mehrerer XML-Dokumente) bzw. mit Variablen
  - Auch in einfachen Stylesheets (mit LRE als top-level Element) möglich (mit absolutem Pfad in der select-Expression)

xsl:if

- xsl:if
  - Definition von bedingten Anweisungen
  - test-Attribut enthält die Bedingung als boolean Expression
  - kein else-Zweig möglich

- Beispiel 1:

```
<xsl:template match="veranstaltung">
  <xsl:value-of select="titel"/>
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

- Beispiel 2:

```
<xsl:template match="item">
   <tr>
      <xsl:if test="position() mod 2 = 0">
        <xsl:attribute name="bgcolor">yellow</xsl:attribute>
      </xsl:if>
      <xsl:apply-templates/>
   </tr>
</xsl:template>
```

xsl:choose

- xsl:choose
  - Zur Unterscheidung beliebig vieler Fälle im Stil von

    if ... then ... else if ... then .... else if ... then ... else ....
  - Keine Attribute; Content enthält ein oder mehrere xsl:when-Elemente und optional ein xsl:otherwise-Element
  - Ausgeführt wird der Content des ersten xsl:when-Elements, für das das test Attribut true ergibt. Falls alle false liefern, wird der Content des xsl:otherwise Elements ausgeführt.
- xsl:when: test-Attribut enthält die Bedingung als boolean Expression
- xsl:otherwise: keine Attribute
- Allgemeine Struktur:

<xsl:choose>

<xsl:when test="boolescher Ausdruck">...das geschieht....**</xsl:when>**

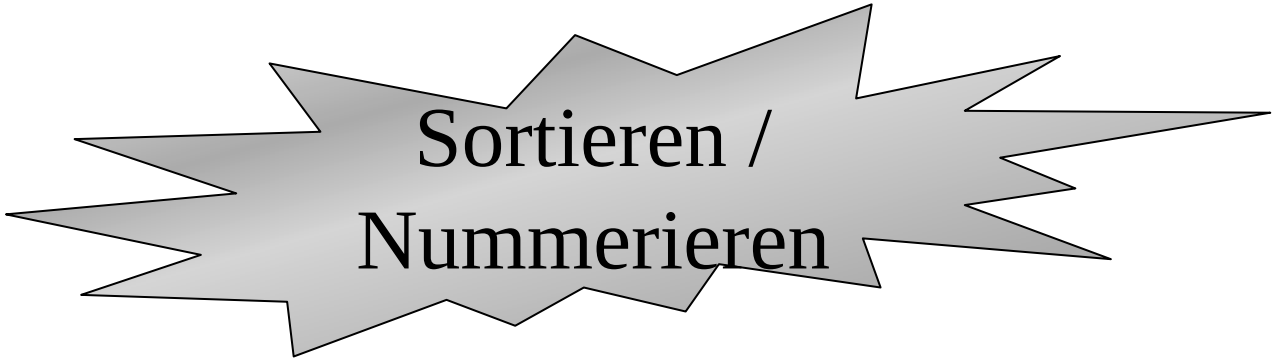
<xsl:when test="boolescher Ausdruck">...ansonsten das....**</xsl:when>**

<xsl:when test="boolescher Ausdruck">...und sonst das....**</xsl:when>**

<xsl:otherwise>...wenn gar nichts geht, dann das... **<xsl:otherwise>**

</xsl:choose>

xsl:sort



- xsl:sort als Kind von for-each und apply-templates erlaubt
  => Veränderung der Ordnung der selektierten Knoten
- Mehrere xsl:sort Elemente können aufeinander folgen
  => primäre, sekundäre, ... Sortierung
- Attribute von xsl:sort
  - select: string-expression als Sortier-Schlüssel
  - data-type: text, number
  - order: ascending, descending
  - case-order: lower-first, upper-first (d.h.: ob Groß- oder Kleinbuchstaben Vorrang haben)

Beispiele

Beispiel 1: Sortiere das Ergebnis von xsl:for-each

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0"
  encoding="ISO-8859-1"/>

<xsl:template match="/">
  <resultat>
    <xsl:for-each select="//mitarbeiter">
      <xsl:sort select="name" order="ascending"/>
      <tupel>
        <xsl:copy-of select="."/>
      </tupel>
    </xsl:for-each>
  </resultat>
</xsl:template>

</xsl:stylesheet>
```

Beispiel 2: Sortiere das Ergebnis von xsl:apply-templates

XML-Dokument:

```
<employees><employee><name><given>James</given>
                              <family>Clark</family></name>...
          </employee>
</employees>
```

Templates im XSLT-Stylesheet:

```
<xsl:template match="employees">
   <ul><xsl:apply-templates select="employee">
         <xsl:sort select="name/family"/>
         <xsl:sort select="name/given"/>
       </xsl:apply-templates> </ul>
</xsl:template>

<xsl:template match="employee">
   <li> <xsl:value-of select="name/given"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="name/family"/> </li>
</xsl:template>
```

xsl:number

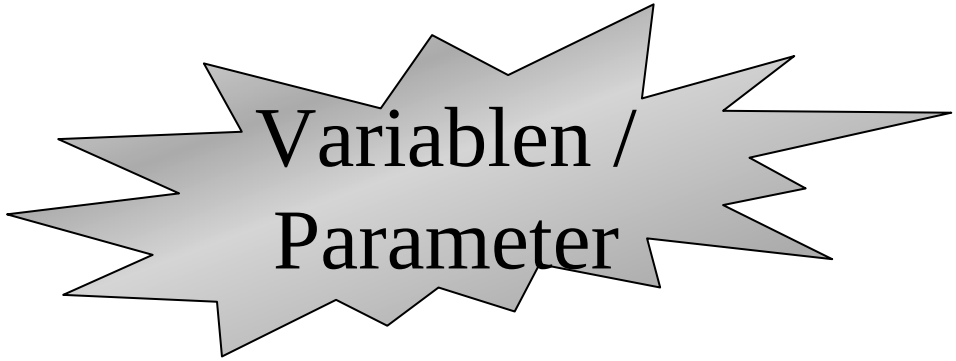
- Verwendung:
  - ähnlich wie ordered list (ol) in HTML
  - fügt formatierte ganze Zahlen ins Dokument ein
  - nummeriert automatisch Elemente eines oder mehrerer Element-Typen durch, sofern nicht mit value-Attribut eine best. Zahl angegeben wird.
- 9 optionale Attribute:
  - `value`: XPath Ausdruck, der in Zahl umgewandelt und auf die nächst größere ganze Zahl gerundet wird.
  - `level`: gibt an, ob und wie die Tiefe eines Elements im Dokumentenbaum berücksichtigt werden soll. 3 mögliche Werte:
    - level= "single" (=default): bzgl. jedem parent beginnt Zählung von vorne
    - level="any": alle Knoten desselben Typs im Baum (nicht nur Geschwister)
    - level="multiple": wie any, aber Nummerierung berücksichtigt Tiefe (z.B. 1.2, 1.3.1)

- `format`: Fünf Formate für die Nummerierung sind möglich:
  - format = "1": numerisch (1,2,3...)
  - format = "a": alphabetisch, Kleinbuchstaben (a,b,c...)
  - format = "A": alphabetisch, Großbuchstaben (A,B,C...)
  - format = "i": römische Zahlen, Kleinbuchstaben (i,ii,iii...)
  - format = "I": römische Zahlen, Großbuchstaben (I,II,III...)

  Bemerkung: Bei level="multiple" kann das format-Attribut auch mehre Format- Specifier enthalten, z.B.: format= "1.1.a"

- weitere Attribute:
  - from: Startknoten für Zählung
  - lang: Sprache (für alphabetische Nummerierung relevant)
  - letter-value: steuert alphabetische Darstellung von Zahlen
  - grouping-separator, grouping size, z.B. 1234567 -> 1.234.567
  - count: XPath Ausdruck, der angibt, welche Typen von Knoten gezählt werden (auch wenn ev. nicht alle davon ausgegeben werden)

Variablen



- Definition:
  - Mittels xsl:variable Element
  - Global (als top-level Element) oder lokal innerhalb eines Template
  - Lokale Definition kann globale Definition überlagern
- Variablen-Bindung:
  - Mittels select-Attribut oder als Content von xsl:variable, d.h.:
    - `<xsl:variable name = "var" select="expr"/>`
    - `<xsl:variable name = "var" />.....</xsl:variable>`
  - Wert einer Variable kann nicht verändert werden, d.h.: einmalige Zuweisung
- Verwendung von Variablen:
  - mit `$var`
  - z.B.: in Attributen <**veranstaltung jahr="{$thisyear}**">

Beispiel: Variable verbindet unterschiedliche Kontexte

```
<lehre>
   <mitarbeiter>
      <name>...</name>
      <veranstaltung nr="xxx"/> <veranstaltung nr="zzz"/>
   </mitarbeiter>
   <veranstaltung nr="xxx"><titel>...</titel>  ...  </veranstaltung>
   <veranstaltung nr="yyy"><titel>...</titel>  ...  </veranstaltung>
 </lehre>

<resultat xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xsl:version="1.0" >
     <xsl:for-each select="/lehre/veranstaltung">
        <xsl:variable name="vst" select="."/>
        <xsl:for-each select="//mitarbeiter/veranstaltung[@nr=
           $vst/@nr]/../name">
           <tupel>
             <xsl:copy-of select="."/>
             <xsl:copy-of select="$vst/titel"/>
           </tupel>
        </xsl:for-each>
     </xsl:for-each>
   </resultat>
```

-> erzeugt Paare (Mitarbeiter,Veranstaltung)

RTF: Result Tree Fragment

- RTF: fünfter möglicher Datentyp bei Variablen und Parametern
- Kann entstehen bei Variablen/Parameterbindung mit komplexem Content (wenn dieser Subelemente produziert), z.B.:

```
<xsl:variable name="numvar">
  <num><xsl:value-of select="1"/></num>
  <num><xsl:value-of select="2"/></num>
  <num><xsl:value-of select="3"/></num>
</xsl:variable>
```

- RTF ist sehr ähnlich wie ein node-set:

  `<xsl:copy-of select="$numvar"/>` liefert
  `<num>1</num><num>2</num><num>3</num>`

- Node-set Funktionen sind aber im allgemeinen nicht erlaubt, z.B.:

  `<xsl:value-of select="count($numvar)"/>` liefert einen Fehler

- Bei Umwandlung in einen String muss man sich einen zusätzlichen parent dieser Knoten dazudenken: **`<xsl:value-of select="$numvar"/>`**
  liefert die Konkatenation aller String-Werte, d.h.: `"123"`

Parameter

- Definition/Parameter-Bindung:
  - Mittels xsl:param Element
  - Bindung exakt wie bei Variablen.
  - Allerdings: Zugewiesener Wert bei Parameter ist nur der Default für den Fall, dass kein anderer Parameterwert übergeben wird.
- Parameter-Übergabe:
  - Globale Parameter: von XSLT-Prozessor versorgt
  - Lokale Parameter in benannten Templates: mittels xsl:with-param:

```
<xsl:template name="ichbinvielfaerbig">
  <xsl:param name="farbe">green</xsl:param>
  <font color="{$farbe}"><xsl:value-of select="."/></font>
</xsl:template>

...<xsl:call-template name="ichbinvielfaerbig">
      <xsl:with-param name="farbe">red</xsl:with-param>
   </xsl:call-template>
```

Beispiel: Rekursives Template

Beim Erzeugen einer Hotelseite soll der Integer-Wert des Attributs „sterne“ als Folge von Stern-Bildern ausgegeben werden (z.B.: drei Sterne hintereinander).

```
<xsl:template match="bewertung" name="sternwandler">
   <xsl:param name="sternenzahl" select="@sterne"/>
   <img src="star.gif"/>
   <xsl:if test="$sternenzahl>1">
        <xsl:call-template name="sternwandler">
               <xsl:with-param name="sternenzahl"
                      select="($sternenzahl)-1"/>
        </xsl:call-template>
   </xsl:if>
</xsl:template>
```

Defaultwert
ist der Attributwert

ist Sternenzahl
<= 1 dann keine
Rekursion mehr

neuer Parameterwert
für nächsten
rekursiven Aufruf

document()

- erlaubt den Zugriff auf Knoten(mengen) mehrer Dokumente

  node-set **document**(*object,node-set?*)

- Parameter beim Aufruf von document():
  - `document(string)`

    string = URI des zusätzlichen Input-Dokuments,
    Spezialfall: string="" referenziert das XSLT-Stylesheet

    Ergebnis = root node des Dokuments
  - `document(string,node-set)`

    string = URI des zusätzlichen Input-Dokuments,

    Ergebnis = Knotenmenge, die mit der node-set expression selektiert wird
  - `document(node-set,node-set)`

    Ergebnis = Vereinigung aller Knotenmengen, die man bei document()-Aufrufen mit string-Werten der ersten node-set expression (und gleich bleibender zweiter node-set expression) erhält.
- Ordnung von Knoten unterschiedlicher Dokumente:
  - prozessorabhängig

Beispiel

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
   "http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml"/>
<xsl:variable name="emps" select="document('merge2.xml')"/>
<xsl:template match="/">
   <employees>
   <xsl:for-each select="employees/employee">
      <xsl:copy-of select="."/>
   </xsl:for-each>

   <xsl:for-each select="$emps/employees/employee">
      <xsl:copy-of select="."/>
   </xsl:for-each>
   </employees>
</xsl:template>
</xsl:stylesheet>
```

Variable die mit dem root node des anderen Dokuments initialisiert wird

Man braucht immer ein default Dokument (kann auch dummy Dokument sein).

Beispiel Xal13

Keys

- Top- Level Element xsl:key
  - Definition von keys (x,y,z)
  - x = node, y = key name, z = key value
- Attribute von xsl:key (alle mandatory):
  - name: Name des Schlüssels (d.h.: y)
  - match: expression, die node-set liefert (d.h.: Menge aller nodes x)
  - use: liefert relativ zu jedem node x ein oder mehrere key values z

    d.h.: expression von use liefert string: dieser string ist der key value z

    expression von use liefert node-set: string-value jedes Knoten ist ein z.
- Funktion *node-set* **key** (*string,object*)
  - im Prinzip wie id() (wobei das erste Argument von key() der key name ist),

    z.B.: angenommen es gibt nur ein ID-Attribut im Dokument:

    `<!ATTLIST div id ID #IMPLIED>`

    => Für **`<xsl:key name="idkey" match="div" use="@id"/>`**

    sind **`key("idkey",@ref)`** und **`id(@ref)`** äquivalent

    (vorausgesetzt das ref-Attribut enthält keinen Whitespace)

Keys vs. IDs

Keys bieten einige Verallgemeinerungen gegenüber IDs:

- Keys können auch noch „nachträglich“ im Stylesheet definiert werden.
- Key-value z eines Elements y muss nicht unbedingt ein Attribut sein (kann bel. XPath-Expression für context-node y sein).
- Key-value kann ein bel. String sein (nicht nur Name-Token)
- Es kann im Dokument mehrere Keys geben (mit unterschiedlichen Namen x und unterschiedlichen „Value spaces“ für die Werte z).
- Ein Knoten y kann mehrere Keys x besitzen.
- Key values müssen nicht eindeutig sein (d.h.: meherere Knoten y können für einen key x denselben key value z haben).

Weitere Funktionen

- *node-set* **current**()
  - Auswahl des current node
  - auf äußerster Ebene wie "." , z.B.: **<xsl:value-of select="."/>** ist dasselbe wie **<xsl:value-of select ="current()"/>**
  - wichtig für Joins, z.B. **veranstaltung[@nr=current()/@nr]**
- *string* **generate-id**(*node-set?*)
  - erstellt eine eindeutige id für den ersten Knoten in der Knotenmenge
  - Wenn das Argument fehlt, wird die id für den current node berechnet
  - Die id ist prozessorabhängig (muss aber für einen best. Knoten bei wiederholten Aufrufen von generate-id immer dieselbe id sein!).
  - Beispiel: Hyperlinks in einem HTML-Dokument:

    Def. Anker: **<a href="{generate-id(expr)}"/>**

    Def. Link: ** ..... **

- XSL-Formatting Objects:
  - W3C-Recommendation seit 2001
- XSLFO bietet CSS Fähigkeiten (und mehr) für XSL an
  - Beruht auf ähnlichem Formatierungsmodell wie CSS
  - 56 XSL-FO Elemente entsprechen rechteckigen Bereichen (analog zu CSS-Boxes) bzw. Containern für solche Bereiche
  - XSL-FO Attribute entsprechen CSS-Eigenschaften
  - erweiterte Möglichkeiten gegenüber HTML/XML + CSS, z.B.: XSLFO ist seitenorientiert, unterstützt Fußnoten, Randnotizen, …
- XSLFO-Prozessor:
  - Ergebnisdokument wird auf PDF gerendert (bzw. von Browser dargestellt, derzeit aber noch keine Unterstützungen)
  - Im Moment noch keine Software, die alle Features unterstützt

XSLFO Areas

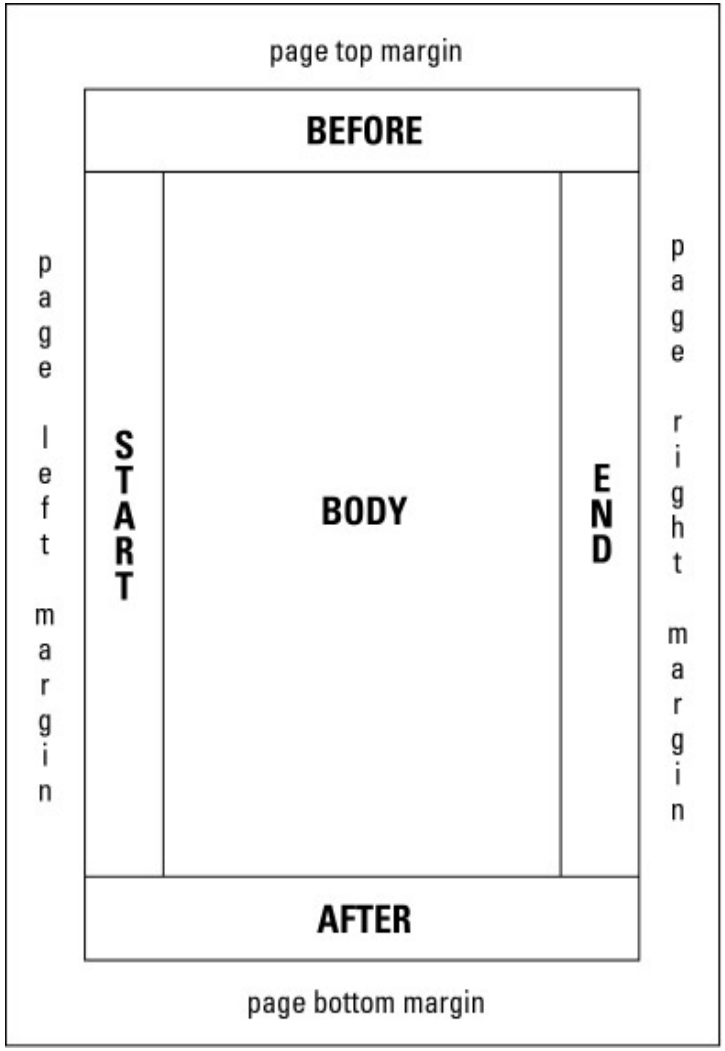
4 Arten von rechteckigen Bereichen bzw. Containern:

- Regions: Container auf oberster Ebene
  - eine Seite enthält region-body, region-before, region-start, ...
  - Page Margins außerhalb von Regions
- Block Areas
  - können andere Block-, Line- und Inline Areas enthalten
  - sequentielle Positionierung auf Seite (nicht mittels Koordinaten!)
- Line Areas
  - Textzeilen innerhalb eines Blocks
  - keine eigenen Formatierungsobjekte (werden vom Formatierer erzeugt)
- Inline Areas
  - enthält Teile einer Zeile, z.B. Text, andere Inline Areas
  - automatisch in Line Areas gerendert

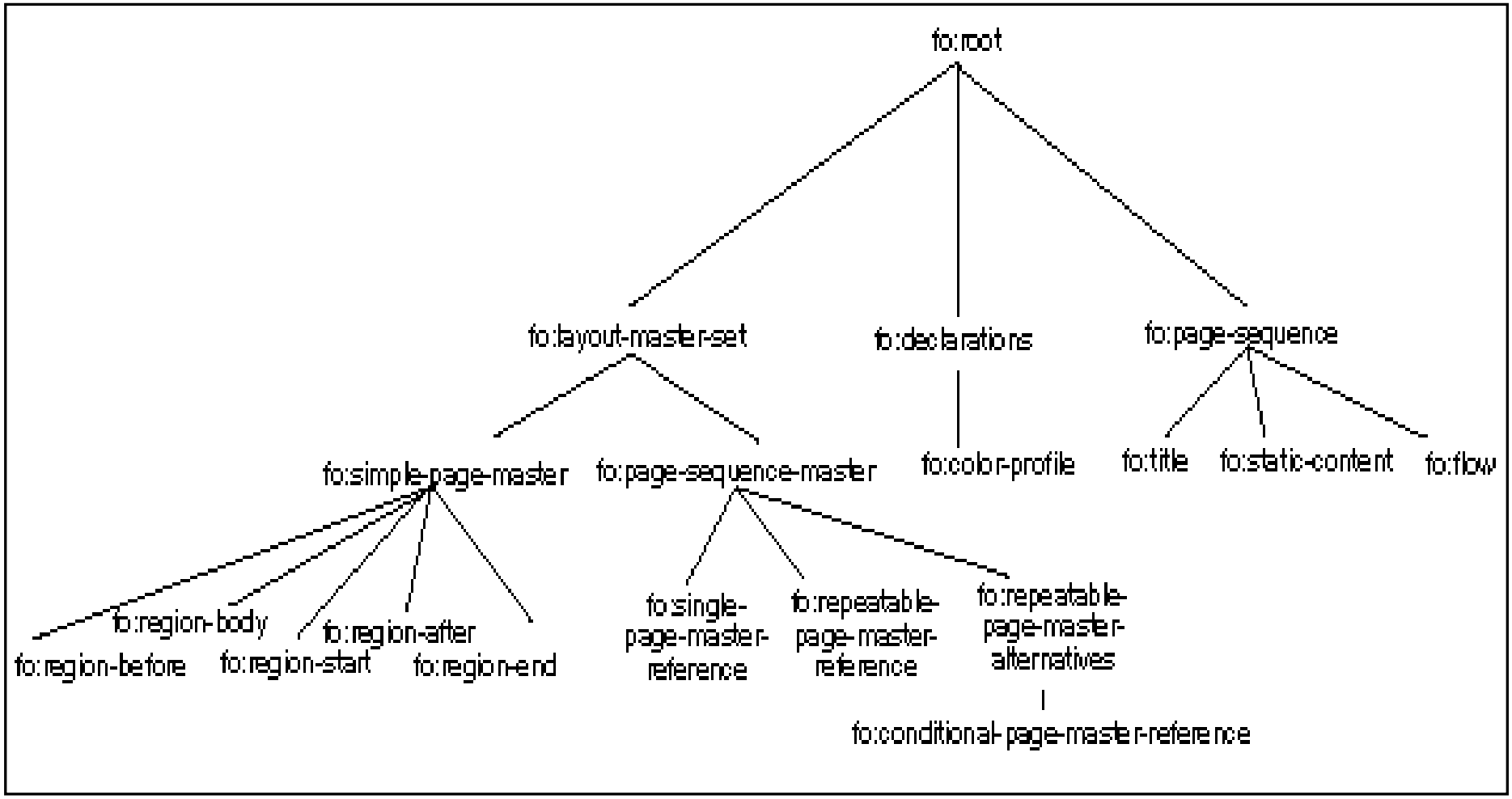
Bereichscontainer

Oberste Container

- **`region-body`**
- **`region-before`**
- **`region-after`**
- **`region-start`**
- **`region-end`**



Aufbau eines XSLFO-Dokuments



- Namespace: http://www.w3.org/1999/XSL/Format
- Präfix: üblicherweise fo

layout-master-set

- Enthält Vorlagen (= *page-master)*
- Grundsätzliches Layout:
  - header, footer, margins, body
- derzeit nur *simple-page-master* für rechteckige Seiten
- Jede Seite laut page-sequence ist einem *page-master* zugeordnet

page-sequence

- Definiert den Inhalt für eine Menge von Seiten
- Ist einem bestimmten *page-master* zugeordnet (mittels Attribut master-reference).
- Optionales *title* Element
- Beliebig viele *static-content* Elemente die auf jeder Seite stehen (header, seitenzahl)
- Ein *flow* Element (pro Region-Container), das den Inhalt der Seite definiert

XSLFO-Prozessoren

- Apache FOP
  - unvollständige Implementierung; derzeit: Version 0.20.4
  - benötigt Parser, Transformer (zB Xerces, Xalan) und Klassen aus Batik-Projekt (alle benötigten Packages beim Download inkludiert)
  - **`set CLASSPATH = fop.jar;xalan-2.3.1.jar;xercesImpl-2.0.1.jar;xml-apis.jar;avalon-framework-cvs-20020315.jar;logkit-1.0.jar;jimi-1.0.jar`**
  - **`java org.apache.fop.apps.Fop fo-Datei pdf-Datei`**
- Weitere XSLFO-Prozessoren (kommerzielle Produkte)
  - XSL Formatter: AntennaHouse
  - XEP: RenderX
  - E3: Arbortext
  - XSL-FO Renderer: Advent
  - Adobe Document Server: Adobe Systems

Beispiel

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format" version="1.0">
<xsl:output method="xml"/>
<xsl:template match="/">
  <fo:root>
   <fo:layout-master-set>
     <fo:simple-page-master master-name="Lehre"
        page-height="29.7cm"
        page-width="21cm">
      <fo:region-body margin-top="2cm"/>
      <fo:region-before extent="1cm"/>
      <fo:region-after extent="1cm"/>
     </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-reference="Lehre">
   <fo:static-content flow-name="xsl-region-before">
        <fo:block>Lehrveranstaltungen</fo:block>
   </fo:static-content>
   <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates select="//titel"/>
   </fo:flow></fo:page-sequence></fo:root>
</xsl:template>
<xsl:template match="titel">
   <fo:block font-size="28pt">
        <fo:inline font-style="italic">
                <xsl:value-of select="."/>
        </fo:inline></fo:block>
</xsl:template></xsl:stylesheet>
```

Root-Element fo:root

Definition von einem simple master mit Namen "lehre" und Setzen von Region-Eigenschaften

Seitensequenzen

statischer Inhalt, in dem Fall nur Kopfzeile (könnte auch Seitennr. etc. enthalten)

Seitentext; hier der dynamisch mit XSLT generierte Text; arbeitet weiter auf titel Knoten

weiteres template, welches die Titel extrahiert und formatiert

Beispiel FOP2

Weitere Beispiele

- Zeichen-Formatierungen:

  - Zahlreiche Attribute (von fo:block, fo:inline, ...)
  - z.B.: color, font-family, font-weight, font-style, font-size, ...

- Listen, Tabellen:

  - Elemente fo:list-block, fo:list-item, fo:list-item-label, ...
  - Elemente fo:table, fo:table, fo:table-column, fo:table-body, fo:table-row, fo:table-cell, ...

- Weitere Formatierungsobjekte:

  - Graphiken: fo:external-graphic
  - Seitennummer: fo:page-number
  - Verweis auf Seite: fo:page-number-citation

Übungsbeispiel A

Betrachten Sie die XML-Datei Teilnehmer.xml laut Übungsbeispiel A des 5. Termins plus die XML-Datei Gruppen.xml.

Schreiben Sie ein Stylesheet Xml2Htm.xsl, das aus diesen beiden XML-Dateien eine HTML-Datei mit folgender Tabelle erzeugt.

1. Jede Zeile enthält Nachname, Vorname, MatrNr, Kennzahl, Email und Gruppe eines Studenten.
2. Die Studenten sollen alphabetisch (nach dem Nachname – bei Gleichheit nach dem Vornamen) sortiert werden.
3. Zusätzlich soll (als erste Spalte) eine fortlaufende Nummer ausgegeben werden.

Verwenden Sie für die Zeilen der Tabelle abwechselnd zwei verschiedene Formatierungen (z.B.: Hintergrundfarbe, Schriftfarbe, etc.), für die Sie entsprechende attribute-sets definieren.

Übungsbeispiel B

Erstellen Sie für das XML Dokument laut Übungsbeispiel B des ersten Termins (d.h.: relationale Uni-Datenbank) ein XSLT-Stylesheet sql.xsl, das ein HTML-Dokument mit einer Tabelle erzeugt, die das Ergebnis der folgenden SQL-Anfrage enthält:

select h.MatrNr, v.Titel, v.VorlNr
from hören h, Vorlesungen v, Professoren p
where p.Name = ´Sokrates´ **and** v.gelesenVon = p.PersNr **and**
v.VorlNr = h.VorlNr

(d.h. MatrNr der Studenten, die Vorlesungen von Sokrates besuchen, plus VorlNr und Titel dieser Vorlesungen)