Anhang B - Nutzung von XSQL

Bei der Erstellung von XSQL-Seiten, werden immer zwei Dateien benötigt. Als erstes die XSQL-Datei und dann das dazugehörige Stylesheet (welches wiederum zur Formatierung der HTML-Tags ein CSS verwenden kann), die XSL-Datei.

In der XSQL-Datei stehen immer die SQL-Anweisung drin. Dieses beinhaltet Updates, Inserts, Select und Deletes. Im Header wird die XML-Version und der ISO-Standard definiert.
Der Header ist immer anzugeben. Das Attribut encoding="ISO-8859-1" ist optional, ist aber zu empfehlen, wenn <!--Kommentare--> benutzt werden, die Umlaute, wie ö,ä,ü usw. enthalten. Ohne dieses Zusatzattribut, kommt es zur Fehlermeldung des Parsers, da Umlaute nicht Standard sind. XML-Dateien mit den Encoding-Attribut können nur vom IE5 mit dem MSXML Parser 3.0 gelesen werden. Dieser Parser ist nicht Standard, so dass ein Update durchgeführt werden muß. Dieses Update ist bei www.microsoft.de erhältlich. Der Parser wird unverzichtbar, wenn die XML-Datei Elemente mit Umlauten enthält.

Beispiel für XML-Datei mit Umlaut:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Dozent>
      <Name>Kudraß</Name>
      <Vorname>Thomas<Vorname>
</Dozent>
```

Beispiel für XSQL-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="dohtml.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
  SELECT VORNAME,NAME, TELEFON,EMAIL from dozent WHERE VORNAME &lt;> 'N.'
  <!--"Kleiner als" funktioniert nicht, man benutze '&lt; -->
 </xsql:query>
```

In dem Tag <?xml-stylesheet type="text/xsl" **href="dohtml.xsl"**?> muß das XSL-File mit angegeben werden, welches dafür verantwortlich ist, in welcher Art die Ausgabe erfolgt. Diese kann als HTML, PDF oder als Grafik (SVG, Plug-In gibt's kostenlos bei Adobe , funktioniert im IE und Netscape) erfolgen.
In dem Tag <xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql"> werden die Datenbankverbindung und das Servlet definiert (Definition siehe: Installation Tomcat, XDK).
Als nächstes kommt die reine SQL-Anweisung. Diese erfolgt wie gehabt, mit einer einzigen Ausnahme. Bei einem Vergleich mit '<>' muß das '<' durch die HTML-Substituition < ersetzt werden (sieht dann so aus '<>'). Der Grund dafür liegt vermutlich in der Architektur des Parsers

und ist eine reine Vorsichtsmaßnahme, da es bei der Schreibweise '<>' zu einer Verwechslung mit einem Tag kommen könnte.

Aufbau von Querys in XSQL-Dateien

Standardmäßig sehen XSQL-Querys so aus:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="doz.xsl"?>
<xsql:query connection="xml" xmlns:xsql="urn:oracle-xsql">
  SELECT vorname, name,email, telefon from dozent
 </xsql:query>
```

Bei dieser Art von Query werden kein Rowset-Name (Zeilensatz) und kein Row-Name (Zeile) vergeben, so dass sie im XSL-Stylesheet mit <xsl:for-each select="ROWSET/ROW">.
Ohne Angabe der Zeile <?xml-stylesheet type="text/xsl" href="doz.xsl"?> wird das generierte XML-File ausgegeben.

```
<?xml version="1.0" encoding= "iso-8859-1"?>
- <ROWSET>
  - <ROW num="1">
      <VORNAME>N.</VORNAME>
      <NAME>N.</NAME>
    </ROW>
  - <ROW num="2">
      <VORNAME>Klaus</VORNAME>
      <NAME>Bastian</NAME>
      <EMAIL>bastian@imn.htwk-leipzig.de</EMAIL>
      <TELEFON>0341/3076432</TELEFON>
    </ROW>
  - <ROW num="3">
      <VORNAME>Hans-Jürgen</VORNAME>
      <NAME>Dobner</NAME>
      <EMAIL>dobner@imn.htwk-leipzig.de</EMAIL>
      <TELEFON>0341/3076486</TELEFON>
    </ROW>
</ROWSET>
```

Der Rowset-Name ist <ROWSET> und der Row-Name <ROW> . Die Zeile enthält den Datensatz. Bei Angabe der Bezeichnung werden diese auch verwendet.

Beispiel:

XSQL-Datei:

```
<?xml version="1.0"?>
<xsql:query connection="xml" rowset-element="IMN" row-element="DOZENT"
xmlns:xsql="urn:oracle-xsql">
  SELECT vorname, name,email, telefon from dozent
 </xsql:query>
```

WICHTIG: Querys dürfen keine Semikolons enthalten. Bei COPY & PASTE aus SQL+ aufpassen, da dort Querys immer mit Semikolons abgeschlossen werden!!

Ausgabe der XML-Datei:

```
<?xml version="1.0" encoding= "iso-8859-1"?>
- <IMN>
  - <DOZENT num="1">
      <VORNAME>N.</VORNAME>
      <NAME>N.</NAME>
    </DOZENT>
  - <DOZENT num="2">
      <VORNAME>Klaus</VORNAME>
      <NAME>Bastian</NAME>
      <EMAIL>bastian@imn.htwk-leipzig.de</EMAIL>
      <TELEFON>0341/3076432</TELEFON>
    </DOZENT>
  - <DOZENT num="3">
      <VORNAME>Hans-Jürgen</VORNAME>
      <NAME>Dobner</NAME>
      <EMAIL>dobner@imn.htwk-leipzig.de</EMAIL>
      <TELEFON>0341/3076486</TELEFON>
    </DOZENT>
 </IMN>
```

Im XSL-Stylesheet wird die Query mit <XSL:for-each select="IMN/DOZENT"> angesprochen.

WICHTIG: BEI DER ABFRAGE VON QUERYS IN XSL-STYLESHEETS MÜSSEN ALLE SPALTENNAME GROSS GESCHRIEBEN WERDEN

Beispiel für Ausgabe in Tabelle ohne Verwendung von Rowset-element und row-element:

```
<table border="1">
      <tr>
      <td align="center">Vorname</td> <!--Tabellenkopf -->
      <td align="center">Name</td>
      <td align="center">Telefon</td>
      <td align="center">E-Mail</td>
     </tr>
<xsl:for-each select="ROWSET/ROW">
     <tr>

      <td> <xsl:value-of select="VORNAME"/> </td>
      <td> <xsl:value-of select="NAME"/> </td>
      <td> <xsl:value-of select="TELEFON"/> </td>
      <td> <xsl:value-of select="EMAIL"/> </td>
    </tr>
</xsl:for-each>
</table>
```

Alternativ-Querys

Falls für eine Query nur ein leeres Resul-Set existieren sollte, ist es möglich Alternativ-Querys anzugeben, die dann ausgeführt werden. Diese werden mit in das <xsql:query> eingefügt.

<xsql:query>select …
<xsql:no-rows-query>select 'hello' from dual</xsql:no-rows-query>
</xsql:query>

Zusatz:

Soll die Query auf eine Maximal-Anzahl von Zeilen beschränkt werden, muß im Tag <xsql:query ..> das Attribut max-rows="{integer}" hinzugefügt werden.

Querys mit Insert, Update und Delete

XSQL-Files mit Insert, Update und Delete werden üblicher von Formularen über die Submit-Funktion aufgerufen. Beim Submit werden die nötigen Parameter mit übertragen.

Insert*:*

```
<?xml version = "1.0"?>
<xsql:insert-request connection="xml" xmlns:xsql="urn:oracle-xsql"
  table="dozent" transform="insert.xsl"/>
```

table ist die Bezeichnung der Tabelle in der die Daten abgespeichert werden sollen.
In transform wird das XSL-Stylesheet angegeben, das die Daten für das Insert aufbereitet.

Insert.xsl:

```
<?xml version = "1.0" encoding="ISO-8859-1"?>
<ROWSET xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
   <xsl:for-each select="request/parameters">
   <ROW>
 <TITEL><xsl:value-of select="titel"/> </TITEL>
<VORNAME><xsl:value-of select="vorname"/></VORNAME>
<NAME><xsl:value-of select="name"/></NAME>
<FBEREICH><xsl:value-of select="fbereich"/></FBEREICH>
<TELEFON><xsl:value-of select="telefon"/></TELEFON>
<EMAIL><xsl:value-of select="email"/></EMAIL>
<SPERRZEITEN><xsl:value-of select="sperr"/></SPERRZEITEN>
<SPEZIFIKATION><xsl:value-of select="spez"/></SPEZIFIKATION>
</ROW>
   </xsl:for-each>
</ROWSET>
```

Für die Dozenten wird als Schlüssel eine ID verwendet. Da diese aber per Trigger vergeben wird, muß diese auch bei einem Insert per XSQL nicht angegeben werden. Standardwert bei Nicht-Angabe ist NULL.

In der Insert.xsl werden Daten aus einem HTML-Formular verarbeitet. Aus diesen Daten wird intern ein XML-File generiert welches den folgenden Aufbau hat.

```
<?xml version="1.0" encoding= "iso-8859-1"?>
- <request>
  - <parameters>
      <name>Bastian</name>
      <sperr>Sperrzeit IMN + Mittwochs ab 16.15</sperr>
      <vorname>Klaus</vorname>
      <fbereich>IMN</fbereich>
      <email>bastian@imn.htwk-leipzig.de</email>
      <spez>Professor</spez>
      <titel>Prof. Dr.</titel>
      <telefon>0341/3076432</telefon>
    </parameters>
    <session />
    <cookies />
  </request>
```

Die Elemente bekommen die Namen der Formularelemente, die im HTML-Formular verwendet wurden.

<parameters> sind einfach Variablen, die von dem HTML-Formular übermittelt wurden.

In <session> werden Parameter gespeichert, die über mehrere Seiten hinweg benutzt werden können. Diese können mit <xsql:set-session-param name="*NAME*" value="*WERT*"/>

Mit <cookies> kann auf Werte die Cookies stehen zugegriffen.

Erstellen von Cookies mit: <xsql:set-cookie name="*NAME*" value="*WERT*"/>

Der Zugriff auf <parameters>,<session>,<cookies> erfolgt in Transform-Stylesheets für Insert-, Update- und Deleterequests durch:

<xsl:for-each select="**request/parameters**"> für <parameters>
<xsl:for-each select="**request/session**"> für <session>
<xsl:for-each select="**request/cookies**"> für <cookies>

WICHTIG: Auch wenn das Root-Tag im XSQL-Skript ein anderes ist, in Transform-Stylesheets stets die obere Schreibweise verwenden!!

Multiple-Parameters

Sollte ein Parameter mehrfach übergeben werden z.B. für das Löschen mehrerer Zeilen werden mehrere ID's übergeben. Diese ID's sind aber alle unter einem Variablennamen abgespeichert. Sollte daher eine Variable mehrfach verwendet werden werden ROW's erzeugt. Das Request-File sieht wie folgt aus:

Das Element <del> ist der Name der Checkbox und die Zahlen sind die ID's die als Werte der Checkbox übertragen werden.

Mehrere Parameter mit gleichem Name

```
<?xml version="1.0" ?>
- <request>
  - <parameters>
    - <row>
        <del>30</del>
      </row>
    - <row>
        <del>32</del>
      </row>
    - <row>
        <del>34</del>
      </row>
    - <row>
        <del>77</del>
      </row>
    </parameters>
    <session />
    <cookies />
  </request>
```

Ein Parameter

```
<?xml version="1.0" ?>
- <request>
  - <parameters>
      <del>77</del>
    </parameters>
    <session />
    <cookies />
  </request>
```

Da das Request-File mit einem Parameter anders aufgebaut ist, wie eins mit einem Parameter der mehrfach verwendet wird. Dazu muß in dem XSL-Stylesheet eine Abfrage eingebaut werden, die untersucht, ob es sich um ein File mit einfacher oder mehrfacher Nutzung eines Parameters handelt.

Diese kann z.B. so aussehen:

```
<?xml version = "1.0" encoding="ISO-8859-1"?>

<ROWSET xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
<xsl:variable name="first" select="request/parameters/del"/>
<xsl:variable name="second" select="number(request/parameters/row[1]/@num)"/>
 <xsl:if test="$first != ''">
       <ROW>
           <DOZID><xsl:value-of select="request/parameters/del"/></DOZID>
       </ROW>
</xsl:if>


<xsl:if test="$second != ''">

      <xsl:for-each select="request/parameters/row">
         <ROW>
            <DOZID><xsl:value-of select="del"/></DOZID>
         </ROW>
      </xsl:for-each>
</xsl:if>
</ROWSET>
```

Delete:

```
<?xml version = "1.0"?>
<xsql:delete-request connection="xml" xmlns:xsql="urn:oracle-xsql" table="doz ent" transform="delete.xsl"
key-columns="DOZID"/>
```

Delete-Querys werden mit dem Tag <xsql:delete-request> erzeugt. Die Parameter connection und transform gelten analog zum insert-request.

Mit key-columns werden die Spalten definiert, die zum Identifizieren der Spalten helfen sollen. Hier wäre die Angabe des Schlüssels angebracht, es sei denn es sollen Spalten mit einem bestimmten Wert gelöscht werden. Mehrere Spalten werden durch Kommas getrennt.

Das dazugehörige XSL-File ist wie das Insert-request-XSL-File aufgebaut. Nur reicht es hier ein ROWSET zu erzeugen, das als Elemente die Spalten enthält, die als key-columns definiert wurden.

XSL-Stylesheet:

```
<?xml version = "1.0" encoding="ISO-8859-1"?>

<ROWSET xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">
<xsl:variable name="first"><xsl:value-of select="request/parameters/del"/>
</xsl:variable>
<xsl:variable name="second" select="number(request/parameters/row[1]/@num)"/>
 <xsl:if test="$first != ''">
       <ROW>
           <DOZID><xsl:value-of select="request/parameters/del"/></DOZID>
       </ROW>
</xsl:if>

<xsl:if test="$second != ''">

      <xsl:for-each select="request/parameters/row">
         <ROW>
            <DOZID><xsl:value-of select="del"/></DOZID>
         </ROW>
      </xsl:for-each>
</xsl:if>
</ROWSET>
```

Der Parameter 'del' ist ein Parameter aus einem HTML-Formular.

Update:

```
<xsql:update-request connection="xml" xmlns:xsql="urn:oracle-xsql"
table="dozent" transform="doupdate.xsl" key-columns="DOZID"/>
```

Der Aufbau der Query erfolgt analog zum **delete-request.**

Das XSL-File muß die zu geänderten Werte enthalten sowie die Werte, die zur Identifikation nötig sind. Zur Identifikation sind Schlüssel von Vorteil, wenn es sich um einzelne Updates handelt.

Verwendung von Parametern in Attributen

Um Parameter in Attributen wie z.B. Verwendung einer in der Datenbank gespeicherten

E-Mail-Adresse für einen mailto-Link:

```
HTML: <a href="mailto:Datenbank-E-Mail-Adresse">E-Mail-Adresse</a>
```

Damit dieser Tag aus XSL erzeugt, sind folgende Schritte notwendig:

Sollte es sich um **einen** übergebenen Parameter handeln, kann auf diesen direkt mit {@Variable} zugegriffen werden. Das Selbe gilt für Attribute die bei Elementen enthalten sind, z.B. Cursor bringen Zeilennummerierungen mit, die als Attribut enthalten sind. Sollte diese Parameter mehrfach verwendet werden muß dieser erst mit einer neuen Variable substituiert werden. Dazu muß eine Variable definiert und ihr ein Wert zugewiesen werden.

Ausgabe von Daten in einer Tabelle wobei eine Spalte die E-Mail-Adresse als Link beinhalten soll

```
<xsl:for-each select="ROWSET/ROW">
      <xsl:variable name="mail"><xsl:value-of select="EMAIL"/></xsl:variable>
…
…
<td> <a href="mailto:{$mail}"><xsl:value-of select="EMAIL"/></a></td>
```

oder

```
<td> <a href="mailto:{$mail}"><xsl:value-of select="$mail"/></a></td>
…
…
</ xsl:for-each select>
```

Die Ansteuerung einer substituierten Variable erfolgt durch {$Variablenname}. Für <xsl:value-of ..> - Tags können auch Variablen verwendet werden.

WICHTIG: Variablendefinition gelten nur innerhalb des HTML-Tags in der die Variable definiert wurde. Das heißt: Wurde ein Variable innerhalb einer Tabellenzeile definiert, z.B. <tr><xsl:variable … /><td></td></tr>, dann gilt die Variable nur innerhalb von <tr></tr>. Deshalb sollten nach Möglichkeit die Variablen bereits im <body> definiert werden. Somit wird die Weiterverwendung der Variable innerhalb des gesamten Dokumentes garaniert.

Nested Cursor

Nested Cursor werden gebraucht, um in Querys die XML-typische Baumstruktur zu erzeugen. Mit "normalen" Querys werden nur Tabellen zurückgegeben.

Im folgenden Beispiel wird erklärt, wie man eine Query mit Cursor erstellen kann.

Beispiel: Erstellen einer Übersicht aller Fachbereiche mit ihren Studiengängen, Semestern und Fächern in einer Baumstruktur.

Eine Query wie: `select fbereich,studiengang,semester,bezeichnung from faecher` erzeugt eine Tabelle

FBEREICH	STUDIENGANG	SEMESTER	BEZEICHNUNG
IMN	IN	1	Grundlagen der Informatik
IMN	IN	1	Analysis I
…	…		…

Mit einem Nested Cursor wird dagegen ein Baum erzeugt

IMN
|-IN
|- 1.Semester
|- Grundlagen der Informatik
|- Analysis I

Ein Nested Cursor kann beliebig tief geschachtelt werden. Deshalb sollte man beim Erstellen sich von außen nach innen durcharbeiten. Als erstes sollte man sich überlegen, was gruppiert werden soll und wie die Hierarchie aufgebaut ist. In unserem Beispiel ist es Fachbereich → Studiengang → Semester → Fach. Die Hierarchie zeigt, dass ein Baum der Tiefe 4 ensteht.

WICHTIG: Um Redundanz im Baum vorzubeugen ist die group by – Klausel zu verwenden.
Dabei werden alle Spaltenbezeichnungen (keine Cursorbezeichnungen) hinter group by geschrieben. Bis zur Kinderebene muß in den Cursor die group by – Klausel verwendet werden.

Tiefe 1: Es erfolgt die gruppierte Auflistung aller Fachbereiche:

```
select f.fbereich from faecher f group by f.fbereich
```

erzeugt:

```
- <ROWSET>
  - <ROW num="1">
      <FBEREICH>IMN</FBEREICH>
    </ROW>
  - <ROW num="2">
      <FBEREICH>ME</FBEREICH>
    </ROW>
</ROWSET>
```

Group by ist hier schon nötig, da jedem Fach ein Fachbereich zugeordnet. Ohne group by würde der Fachbereich so oft wiederholt aufgelistet werden, wie es Fächer dafür gibt.

Beispiel: Es gibt 12 Fächer im Fachbereich IMN. Ohne group by würde folgendes passieren:

```
- <ROWSET>
  - <ROW num="1">
      <FBEREICH>IMN</FBEREICH>
    </ROW>
   - <ROW num="2">
      <FBEREICH>IMN</FBEREICH>
    </ROW>
...
...
- <ROW num="12">
      <FBEREICH>IMN</FBEREICH>
    </ROW>

</ROWSET>
```

Die nächste Stufe ist die Auflistung der Studiengänge für die einzelnen Fachbereiche.

Jetzt wird der erste Cursor hinzugefügt. Es ist zu beachten, dass in der where-Klausel des Cursors eine Referenz auf die höheren Ebenen zu erzeugen ist. Es ist den Cursor **immer** eine Bezeichnung zu geben, da es sonst zu einer Fehlermeldung kommt.

SQL:

```
select f.fbereich,
      cursor (select fa.studiengang from faecher fa where
      fa.fbereich=f.fbereich group by fa.studiengang) as studgang
from faecher f group by f.fbereich
```

Tiefe 3: Fachbereich → Studiengang → Semester

```
- <ROWSET>
- <ROW num="1">
    <FBEREICH>IMN</FBEREICH>
   - <STUDGANG>
     - <STUDGANG_ROW num="1">
        <STUDIENGANG>IN</STUDIENGANG>
       - <SEMESTER>
         - <SEMESTER_ROW num="1">
            <SEMESTER>1</SEMESTER>
           </SEMESTER_ROW>
         - <SEMESTER_ROW num="2">
            <SEMESTER>2</SEMESTER>
           </SEMESTER_ROW>
        </SEMESTER>
      </STUDGANG_ROW>
    </STUDGANG>
  </ROW>
</ROWSET>
```

Aus Platzgründen wird hier nur eine Teilausgabe angezeigt.

SQL:

```
select f.fbereich,
   cursor (select fa.studiengang,
      cursor (select fae.semester from faecher fae where
              fae.fbereich=f.fbereich and fae.studiengang=fa.studiengang
              group by fae.semester) as semester
    from faecher fa where fa.fbereich=f.fbereich group by fa.studiengang)
    as studgang
from faecher f group by f.fbereich
```

Als letztes muß jetzt noch der Cursor für die Fächer hinzugefügt werden.

```
- <ROWSET>
 - <ROW num="1">
    <FBEREICH>IMN</FBEREICH>
  - <STUDGANG>
     - <STUDGANG_ROW num="1">
        <STUDIENGANG>IN</STUDIENGANG>
       - <SEMESTER>
          - <SEMESTER_ROW num="1">
             <SEMESTER>1 </SEMESTER>
            - <FACH>
               - <FACH_ROW num="1">
                  <BEZEICHNUNG>Grundlagen der Schaltungs- und Digitaltechnik
                  </BEZEICHNUNG>
                 </FACH_ROW>
               - <FACH_ROW num="2">
                  <BEZEICHNUNG>Algebra I</BEZEICHNUNG>
                 </FACH_ROW>
                  ...
                  ...
```

SQL:

```
select f.fbereich,
       cursor (select fa.studiengang,
             cursor (select fae.semester,
                     cursor(select faec.bezeichnung from faecher faec where
                            faec.fbereich=f.fbereich and
                            faec.studiengang=fa.studiengang and
                            faec.semester=fae.semester) as fach
              from faecher fae where fae.fbereich=f.fbereich and
              fae.studiengang=fa.studiengang group by fae.semester)
              as semester
       from faecher fa where fa.fbereich=f.fbereich group by fa.studiengang) as
studgang
from faecher f group by f.fbereich
```

Für die Cursor, die ein leeres Result-Set zurückliefern, kann keine Alternativ-Query angegeben werden. Hier ist es aber möglich parallel dazu einen Cursor zu erstellen, der die Zeilen zählt[1].

Nutzung von PDF in XSQL

Wenn das XSQL-Servlet mit der vorhandenen Anleitung installiert wurde, dann ist das Erstellen von PDFs ohne Probleme möglich. Das dynamische Erstellen der PDFs realisiert die FOP-Engine. Diese wird aktiviert indem folgender Zusatz in die Stylesheet-Definition geschrieben wird:

```
<?xml-stylesheet type="text/xsl" href="stylesheet.xsl" serializer="FOP"?>
```

Um auch in PDF-Stylesheets mit Templates arbeiten zu können, müssen der Namespace für XSL und für FOP vereinbart werden.

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


<xsl:template match="/">
<fo:root >
...
...
</fo:root>
</xsl:template>
```

In einem PDF-Stylesheet wird als erstes das Layout definiert. Dem Layout können Namen zugeordnet werden, so dass mit verschiedenen Seitenlayouts innerhalb eines Stylesheets gearbeitet werden kann.

```
<fo:layout-master-set>
    <fo:simple-page-master master-name="first"
                           page-width="29.7cm"
                           page-height="21cm"
                           margin-top="1cm"
                           margin-bottom="2cm"
                           margin-left="1.7cm"
                           margin-right="1.8cm">
    </fo:simple-page-master>
  </fo:layout-master-set>
```

Page-xxx ist die Definition der Seitenhöhe und -breite. Mit margin-xxx werden die Seitenränder definiert. In PDF-Stylesheets ist es möglich alles millimetergenau zu bestimmen. Die Längenangabe kann in "cm" oder "mm" oder anderen gängigen Maßen angegeben werden. Es muss eine Maßbezeichnung angegeben werden, sonst wird mit "pt" (Points) gearbeitet.

Hinweis: Bei Kommabeträgen muss statt einem Komma der Punkt verwendet werden. Bsp.:1.8 statt 1,8.

[1] Beispiel: 5.4. Dozent löschen

Als nächstes folgen die Tags zur Eröffnung einer Seite.

```
<fo:page-sequence master-name="first">
  <fo:flow flow-name="xsl-region-body">
...
...
  </fo:flow>
</fo:page-sequence>
```

Im Tag `<fo:page-sequence master-name="first">` wird das Layout definiert. Die Namen des Layout müssen mit denen aus `<fo:layout-master-set>` übereinstimmen.
Mit `<fo:flow flow-name="xsl-region-body">` wird der Body einer Seite eröffnet.
Jetzt wird der sichtbare Teil der PDF-Seite implementiert.
Text:

```
<fo:block font-size="16pt" font-family="sans-serif" line-height="16pt"
font-weight="bold" text-align="center">Text</fo:block>
```

Damit mehrere Leerzeichnen angezeigt werden, muss dem `<fo:block>`-Tag noch `white-space-collapse="false"` hinzugefügt werden. Um automatische Zeilenumbrüche zu erzeugen muss noch `wrap-option="wrap"` eingefügt werden. Zum Erzeugen von Leerraum zwischen 2 Texten oder einer Tabelle und Text muss ein Text in Hintergrundfarbe benutzt werden.
Bsp: weißer Hintergrund - `<fo:block color="white">Thomas Matzke</fo:block>`
Soll innerhalb eines Textes ein Wort fett oder kursiv geschrieben werden, dann muss das Tag `<fo:inline>` verwendet werden.
Bsp:

```
<fo:block>Ein <fo:inline font-weight="bold">fettes</fo:inline> und ein
<fo:inline font-style="italic">kursives</fo:inline> Wort.</fo:block>
```

Tabelle:

TEXT1	TEXT2

```
<fo:table border-width="0.3mm" border-style="solid" font-family="times roman">
          <fo:table-column column-width="1.4cm"/>
          <fo:table-column column-width="1.4cm"/>

<fo:table-body  font-family="times roman">
<fo:table-row line-height="11pt"  font-size="11pt" font-weight="bold"
 text-align="center" >
 <fo:table-cell border="1pt solid black">
<fo:block padding="0.7mm">TEXT1</fo:block></fo:table-cell>
  <fo:table-cell border="1pt solid black">
<fo:block padding="0.7mm">TEXT2</fo:block></fo:table-cell>
</fo:table-row>
</fo:table-body>
</fo:table>
```

Alle Spalten müssen im Tabellenkopf definiert werden. Ebenso die Umrandung für die Zellen muss für jede Zelle einzeln definiert werden. Die Zelleninhalte bestehen aus Text-Blöcken.

Listen:

- 1. Listenelement
- 2. Listenelement

```
<fo:list-block provisional-distance-between-starts="0.5cm">
<fo:list-item> <fo:list-item-label>
<fo:block font-family="Arial" padding="5pt">-</fo:block></fo:list-item-label>
<fo:list-item-body start-indent="body-start()"><fo:block padding="5pt" line-
height="18pt">1. Listenelement</fo:block>
</fo:list-item-body>
</fo:list-item>
<fo:list-item> <fo:list-item-label>
<fo:block font-family="Arial" padding="5pt">-</fo:block></fo:list-item-label>
<fo:list-item-body start-indent="body-start()"><fo:block padding="5pt" line-
height="18pt">2. Listenelement</fo:block>
</fo:list-item-body>
</fo:list-item>
</fo:list-block>
```

Hinweise zu XSL-Syntax

- Single-Tags in HTML wie
 müssen in XSL mit / abgeschlossen werden -->
- alle Attribute müssen in Anführungsstriche gesetzt werden, auch die, die in HTML keine Anführungsstriche brauchen oder bekommen
- Tags für Select-Feld wie <option selected>DropBox-Value</option> werden <option selected="true">DropBox-Value</option> geschrieben
- bei Links mit mehreren Parametern beachten das "&" zwischen den Parametern durch "&" zu ersetzen
- Entities beachten: "<" darf nicht für Vergleiche benutzt werden, da es für Tag-Beginn steht
- stattdessen: "<" verwenden
- JavaScript **nicht** in Kommentare `<!-- -->` setzen, da Kommentare nicht geparst werden
- für Kommentare `<xsl:comment> Kommentar </xsl:comment>` verwenden
- soll innerhalb einer Schleife auf ein Element zugegriffen, welches sich nicht im DOM-Baum der abzuarbeitenden Schleife befindet, erfolgt der Zugriff über das Root-Tag
  Bsp: <xsl:value-of select="/datapage/request/parameters/sws" />

WICHTIG: einführendes "/" nicht vergessen, ohne "/" geht der Parser davon aus, dass sich das Element innerhalb des DOM-Unterbaums der Schleife befindet

Editoren: Seeburger XLStyler
Tutorial: www.selfxml.de; http://selfhtml.teamone.de
XSL-Spezifikation: http://www.w3.org/TR/2001/PR-xsl-20010828/