3 XSLT – Einführung in die Transformationssprache

Dieses Kapitel soll als Einstieg in die Transformationssprache XSLT dienen. Wie bereits in den vorangegangenen Kapiteln gezeigt wurde, ist XSLT für die praktische Arbeit mit XSL-FO unabdingbar. Die nun folgenden Erläuterungen dienen dazu, auf XSL-FO hinzuführen und die späteren Beispiele und Anwendungen zu verstehen. Für ein vertieftes Verständnis von XSLT mit all seinen Facetten sei das Buch „Web-Design mit XML“ von Manfred Knobloch und Matthias Kopp empfohlen, das ebenfalls in der Reihe xml.bibliothek des dpunkt-Verlages erschienen ist.

Um den Einstieg zu erleichtern und die Arbeitsweise von XSLT zu verstehen, wird in diesem Kapitel XSLT genutzt, um XML-Dateien in XHTML zu transformieren. Eine solche Transformation ist einfacher durchzuführen und die Resultate lassen sich mit Hilfe eines Browsers betrachten.

3.1 XSLT-Prozessoren und Hilfsmittel

Im folgendem Abschnitt werden die Prozessoren vorgestellt, die für eine Weiterverarbeitung von XML-Daten notwendig sind. Außerdem wird auf zwei Editoren verwiesen, die bei der Entwicklung von XSLT-Stylesheets hilfreich sein können.

Für die Entwicklung bzw. das Schreiben von XSLT-Skripten seien folgende Hilfsmittel genannt:

Hilfsmittel	Beschreibung
Stylus Studio	Der Stylus Studio der Firma Excelon ist eine Entwicklungsumgebung für XSLT. Die kostenlose Demoversion ist unter http://www.exln.com/products/evaluations/ erhältlich.

Autor: Manuel Montero Pineda

Hilfsmittel	Beschreibung
Cooktop	Cooktop ist ein kostenloser, leicht bedienbarer XML/XSLT-Editor mit einer integrierten Browser-Vorschau. http://xmlcooktop.com/

Da für die Anwendung von XSLT ein Programm zur Verarbeitung notwendig ist, werden drei der gängigsten XSLT-Prozessoren im Folgenden kurz vorgestellt:

XSLT-Prozessor	Beschreibung
Saxon	Kostenloser Prozessor, der sowohl in einer Java- als auch in einer Win32-Version vorliegt. Die Ausgabe erfolgt über die Konsole. http://users.iclway.co.uk/mhkay/saxon
Xalan	Der kostenlose Xalan-Prozessor von Apache ist in einer C++- und in einer Java-Version verfügbar. http://xml.apache.org/
MSXML 4.0	Der Microsoft-Prozessor ist Bestandteil des Internet Explorer seit der Version 5.5. Für ältere IE-Versionen kann er nachgerüstet werden unter: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmsxml/html/xmlparser.asp

Für die folgenden Beispiele wird der MSXML 4.0 in Verbindung mit dem Internet Explorer verwendet werden. Wenn auch die unterschiedlichen XSLT-Prozessoren gleiche Skripte nicht in allen Details gleich verarbeiten, können doch die nachfolgenden Beispiele in allen genannten Prozessoren nachvollzogen werden. Sollten Sie den erzeugten XHTML-Quellcode ebenfalls betrachten wollen, sei die Verwendung von Cooktop empfohlen.

Alle Erläuterungen und Stylesheet-Variationen beziehen sich auf folgende XML-Beispiel-Instanz:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>                    ❶
<EUROPA>
   <LAND>
      <NAME>Deutschland</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">82.4</EINWOHNERZAHL>
      <HAUPTSTADT>Berlin</HAUPTSTADT>
      <KFZ-KENNZEICHEN>D</KFZ-KENNZEICHEN>
      <TEL-VORWAHL>0049</TEL-VORWAHL>
   </LAND>
   <LAND>
      <NAME>Frankreich</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">58.5</EINWOHNERZAHL>
      <HAUPTSTADT>Paris</HAUPTSTADT>
      <KFZ-KENNZEICHEN>F</KFZ-KENNZEICHEN>
```

```
      <TEL-VORWAHL>0033</TEL-VORWAHL>
   </LAND>
   <LAND>
      <NAME>Spanien</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">39.4</EINWOHNERZAHL>
      <HAUPTSTADT>Madrid</HAUPTSTADT>
      <KFZ-KENNZEICHEN>E</KFZ-KENNZEICHEN>
      <TEL-VORWAHL>0034</TEL-VORWAHL>
   </LAND>
</EUROPA>
```

❶ Mit der Verarbeitungsanweisung `<?xml-stylesheet>` lässt sich in der XML-Instanz das zu verwendende XSLT-Stylesheet festlegen. Das Attribut `type` wird hierfür auf `text/xsl` gesetzt (sollten Sie nicht den Internet Explorer verwenden, setzen Sie den Wert auf `text/xml`), das Attribut `href` erfordert eine URI (Pfad und Dateiname), die auf das Stylesheet verweist.

Das Stylesheet für eine Transformation in XHTML wird im Folgenden erstellt.

3.2 Schnellstart

Zunächst werden die Grundform eines XSLT-Stylesheets für die Transformation in XHTML vorgestellt und die einzelnen Elemente erläutert.

```
<?xml version="1.0" encoding="UTF-8"?>                              ❶
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/    ❷
    XSL/Transform">
   <xsl:template match="EUROPA">                                    ❸
      <html>
         <title></title>
         <body><h2>
            <xsl:apply-templates/>                                  ❹
         </h2></body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

❶ Bei einem XSLT-Stylesheet handelt es sich um ein gültiges XML-Dokument. Es beginnt also mit einem XML-Prolog.

❷ Das Wurzelelement eines Stylesheets muss immer `<xsl:stylesheet>` oder `<xsl:transform>` lauten. Dies signalisiert dem Prozessor, dass es sich um ein XSLT-Stylesheet handelt. Das obligatorische Attribut `version` hat den Wert 1.0, der dem Prozessor anzeigt, dass es sich um ein Stylesheet gemäß der Recommendation vom November 1999 handelt. Die Version 2.0 besitzt zur Zeit noch den Status eines Arbeitspapiers. Mit Hilfe des zwingend erforderlichen Namespace-Attributs `xmlns:xsl` wird sicherge-

stellt, dass die XSLT-Anweisungen eindeutig sind und sich von anderen Markups unterscheiden. Der Wert des Attributs ist fix.

Vgl. Abschnitt 4.3

❸ Beim `<xsl:template>`-Element handelt es sich um ein Element, das die Funktion einer Schablone besitzt. Immer wenn über das Attribut `match` ein Treffer erfolgt, wird sein Attributwert in den Ergebnisbaum an die entsprechende Stelle gesetzt. Der Attributwert von `match` ist eine XPath-Anweisung (Adressierung). In unserem einfachen Fall ist es der Name des Wurzelelements `EUROPA`. Wenn der Prozessor, der das Quelldokument auf Übereinstimmungen mit den Templates im Stylesheet untersucht, auf das Element `<EUROPA>` trifft, kopiert er den Inhalt des `<xsl:template>`-Elements in das Zieldokument. So erhalten wir die Grundstruktur eines HTML-Dokuments.

❹ Das Element `<xsl:apply-templates/>` veranlasst den Prozessor, den Inhalt aller Elemente, die sich innerhalb des Elements befinden, auf welches das `<xsl:template>`-Element zutraf, in das Zieldokument zu kopieren. Da es sich in diesem Beispiel um das Wurzelelement `<EUROPA>` handelt, wird der gesamte Inhalt unseres Beispiels an dieser Stelle in das Zieldokument kopiert. Die im Stylesheet vorhandenen Leerzeichen, Tab-Sprünge und Zeilenschaltungen zwischen den HTML-Tags und dem einzufügenden Inhalt werden entsprechend den XML-Regeln automatisch ignoriert, weil sie nicht als Inhalts-Bestandteil gelten.

Um das Ergebnis zu testen, ist das Stylesheet unter dem Namen `style.xsl` im Verzeichnis, das die Beispiel-Instanz `Europa.xml` enthält, zu speichern. Sollte der Internet Explorer 5.5 oder höher verwendet werden, braucht nur noch die Datei `Europa.xml` mit dem Browser aufgerufen zu werden.

Sollten Sie einen anderen Prozessor verwenden, müssen Sie diesen meist über die Konsole (Saxon und Xalan) aufrufen, die URIs des Quelldokuments und des Stylesheets angeben und erhalten das Ergebnis in der Ausgabe. Dieses sollten Sie mit `>` in eine Datei mit der Dateierweiterung `.html` umleiten, die Sie anschließend in einem Browser Ihrer Wahl betrachten können. Bei Saxon lautet der Aufruf wie folgt:

```
saxon Europa.xml style.xsl > Europa.html
```

Als Ergebnis sollten Sie in jedem Fall Folgendes sehen:

```
<html>
   <title></title>
   <body>
      <h2> Deutschland 82.4 Berlin D 0049 Frankreich 58.5 Paris F 0033
           Spanien 39.4 Madrid E 0034 </h2>
   </body>
</html>
```

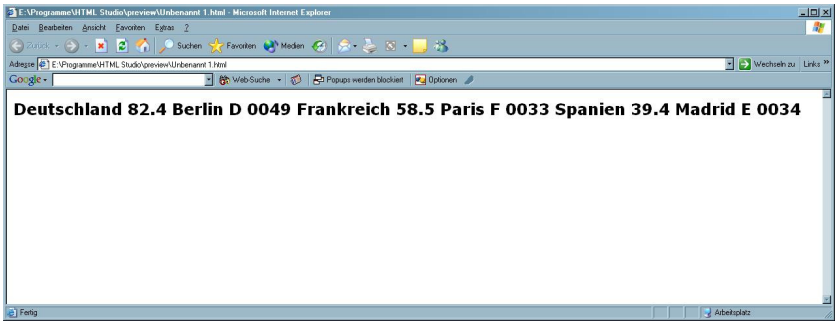


Abb. 3-1
Browser-Ansicht

In den nächsten Abschnitten werden wir uns dem grundsätzlichen Aufbau von XSLT-Stylesheets und den verschiedenen Elementen und ihren Attributen darin widmen.

3.3 Das Output-Element

Mit dem Top-Level-Element `<xsl:output>` lässt sich festlegen, welche Art der Ausgabe generiert werden soll. Als Top-Level-Elemente bezeichnet man solche Elemente, die Kindelemente des Wurzelelements `<xsl:stylesheet>` sein können. Über das Attribut `method` lässt sich angeben, welches Format die Ausgabe haben soll. Mögliche Werte sind hierbei `XML`, `HTML`, `XHTML` und `text`. Die Standardausgabe ist `XML`, es sei denn, es befindet sich ein `<HTML>`-Element im Stylesheet. Da sich in unserem obigen Beispiel ein solches befindet, ist die Angabe des `<xsl:output>`-Elements folglich nicht notwendig. Neben dem `method`-Attribut gibt es noch das `version`-Attribut, das ebenfalls angegeben werden muss und zur Zeit den fixen Wert `1.0` hat. Daneben gibt es einige optionale Elemente, welche u. a. festlegen, ob das XML-Dokument einer DTD folgt oder welches `encoding` (Zeichensatz) verwendet werden soll.

In unserem Beispiel hätte das `<xsl:output>`-Element wie folgt aussehen können:

```
<xsl:output method="html" version="1.0" encoding="UTF-8"/>
```

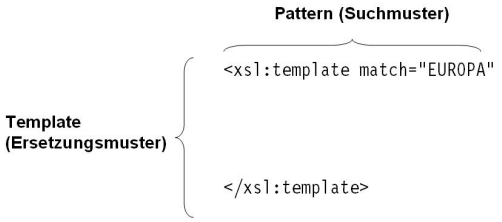
Der Attributwert `UTF-8` bezeichnet den Zeichensatz als Unicode-Zeichensatz im Zeichenumfang von UTF-8 (zur Ausstattung des Zeichensatzes siehe beispielsweise auf Windows-Systemen den Font Arial Unicode MS).

3.4 Template-Regeln

Die Template-Regeln sind Schablonen zur Transformation des Quellbaums (source tree) in den Ergebnisbaum (result tree). Sie bestehen aus zwei Teilen, von denen der eine als Pattern (Suchmuster) bezeichnet

wird, und der andere als Template (Ersetzungsmuster), das ausgeführt wird, wenn das Suchmuster einen entsprechenden Treffer im Quellbaum hat.

Abb. 3-2
Template-Aufbau



Zur Demonstration der Arbeitsweise von Templates wird das Stylesheet-Beispiel um mehrere kleine Templates erweitert.

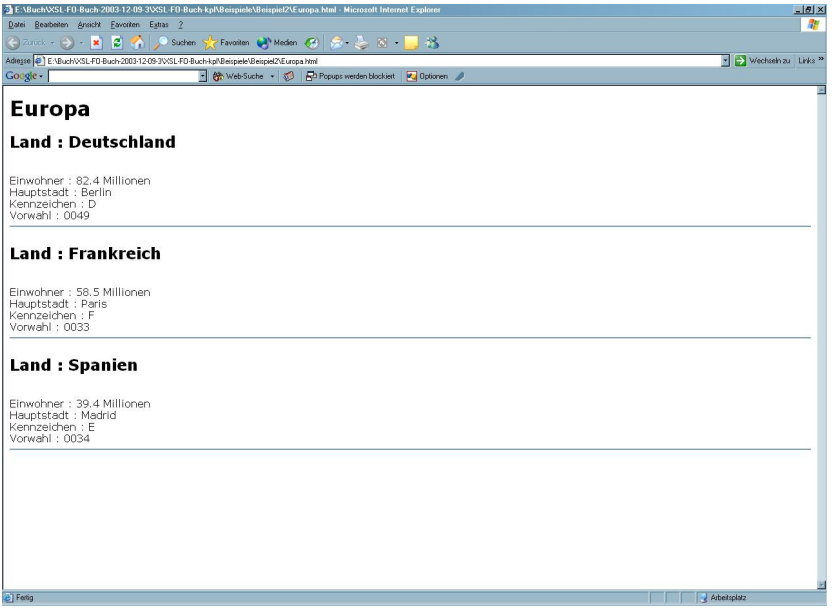
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:template match="EUROPA">
      <html>
         <title></title>
         <body>
            <h1>Europa</h1><xsl:apply-templates/>                    ❶
         </body>
      </html>
   </xsl:template>
   <xsl:template match="NAME">
      <h2>Land : <xsl:apply-templates/></h2>                         ❷
   </xsl:template>
   <xsl:template match="EINWOHNERZAHL">
      <br/>
      Einwohner : <xsl:apply-templates/> Millionen
   </xsl:template>
   <xsl:template match="HAUPTSTADT">
      <br/>
      Hauptstadt : <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="KFZ-KENNZEICHEN">
      <br/>
      Kennzeichen : <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="TEL-VORWAHL">
      <br/>
      Vorwahl : <xsl:apply-templates/>
      <br/><hr/>
   </xsl:template>
</xsl:stylesheet>
```

❶ Der Inhalt des <EUROPA>-Elements wird an an dieser Stelle einschließlich aller Unterelemente mit deren Inhalten hineinkopiert. Der Prozessor sucht nach Unterelementen bis zur bloßen Zeichenbasis und nach möglichen passenden Templates. Da in diesem Beispiel für alle untergeordneten Elemente Templates formuliert sind, wird der Inhalt dieser Templates der Reihe nach in die Zielstruktur kopiert. Gäbe es für ein untergeordnetes Element kein passendes Template, würde der Prozessor dennoch den bloßen Inhalt dieses Elements kopieren.
❷ Neben dem Text Land : wird der Inhalt des <NAME>-Elements in die Zieldatei kopiert. Das Ganze wird in die HTML-Markierungen für <h2> eingekleidet.

Das Ergebnis der Transformation sieht nun wie folgt aus:

```
<html>
   <title></title>
   <body>
      <h1>Europa</h1>                                          ❶
      <h2>Land : Deutschland </h2>                             ❷
      <br/>Einwohner : 82.4 Millionen
      <br/>Hauptstadt : Berlin
      <br/>Kennzeichen : D
      <br/>Vorwahl : 0049<br/><hr/>
      <h2>Land : Frankreich </h2>                              ❷
      <br/>Einwohner : 58.5 Millionen
      <br/>Hauptstadt : Paris
      <br/>Kennzeichen : F
      <br/>Vorwahl : 0033<br/><hr/>
      <h2>Land : Spanien </h2>                                 ❷
      <br/>Einwohner : 39.4 Millionen
      <br/>Hauptstadt : Madrid
      <br/>Kennzeichen : E
      <br/>Vorwahl : 0034<br/><hr/>
   </body>
</html>
```

Abb. 3-3
Browser-Ansicht



In dem gegebenen Beispiel beziehen sich die Templates auf Elemente (Knoten in der XML-Hierarchie des Ausgangsdokuments). Sie werden mit dem Element `<xsl:template>` und dem Attribut `match` spezifiziert. Templates können auch unabhängig von solchen Knoten spezifiziert werden. An Stelle des `match`-Attributs tritt dann das Attribut `name` mit einem beliebigen Namen, z. B. `Vorlage`. Das Element `<xsl:template>` gilt für beide Template-Varianten. Der Aufruf beider Varianten ist unterschiedlich: Knoten-bezogene Templates werden mit dem Element `<xsl:apply-templates>` aufgerufen. Benannte Templates werden mit dem Element `<xsl:call-template>` und dem Attribut `name` aufgerufen, z. B. `<xsl:call-template name="Vorlage">`. Das `match`-Attribut bewirkt, dass nach einem Suchmuster im XML-Dokument gesucht wird und erst bei einer Übereinstimmung das Template ausgeführt wird. Das `name`-Attribut bewirkt, dass lediglich nach dem Template gleichen Namens im Stylesheet gesucht wird, das an der Stelle des Aufrufs ausgeführt wird.

3.5 Auf Knotenwerte zugreifen

Für den Zugriff auf Werte wird das `<xsl:value-of>`-Element mit seinem obligatorischen `select`-Attribut benutzt. Das Template legt die Position im Quellbaum fest. Das `select`-Attribut des Elements `<xsl:value-of>` legt fest, welche Werte in den Ergebnisbaum kopiert werden. Möglich ist u. a. der Zugriff auf den Namen, den Inhalt oder die Attribute des Knotens. Durch die Funktion `name()` als Attributwert von `select` wird der

Name gewählt, durch den Elementnamen als Attributwert wird der Inhalt des Elements gewählt, allerdings ohne die ggf. darin auftretenden Inhalte von Unterelementen (!). Für die Wahl des aktuellen Knotens kann an Stelle des Elementnamens das Zeichen . stehen. Für die Wahl von Attributen steht das @ als Zeichen vor dem Attributnamen zur Verfügung. Beide nun folgenden Templates liefern lediglich in unserem Beispiel das gleiche Ergebnis, weil das Element <EINWOHNERZAHL> keine Unterelemente hat und der Attributwert des Attributs EINHEIT durchgängig gleich Millionen ist.

```
<xsl:template match="EINWOHNERZAHL">
   Einwohner : <xsl:apply-templates/> Millionen
</xsl:template>
<xsl:template match="EINWOHNERZAHL">
   Einwohner : <xsl:value-of select="."/> <xsl:value-of
       select="@EINHEIT"/>
</xsl:template>
```

3.6 Schleifen

Wie in den vorherigen Abschnitten gezeigt, können Inhalte des Quelldokuments über das <xsl:apply-templates>-Element in das Zieldokument kopiert werden. Häufig sollen aber nur bestimmte Elemente daraus kopiert werden. In diesen Fällen bietet sich die Verwendung des <xsl:for-each>-Elements an, beispielsweise für die Erstellung von Inhaltsverzeichnissen. Im folgenden Beispiel werden wir ein solches generieren:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:template match="EUROPA">
      <html>
         <title></title>
         <body>
         <h1>Europa</h1>
         Länder : <br/>
         <xsl:for-each select = "//NAME">                          ❶
            <br/><a>
            <xsl:attribute name="href">#<xsl:value-of              ❷
                select="." /> </xsl:attribute>
            <xsl:value-of select="." /></a>
         </xsl:for-each>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="NAME">
```

```
      <h2> Land :
      <a>
         <xsl:attribute name="name">                                    ❸
            <xsl:value-of select="."/>
         </xsl:attribute>
      </a>
      <xsl:apply-templates />
      </h2>
   </xsl:template>
   <xsl:template match="EINWOHNERZAHL">
      <br/>
      Einwohner : <xsl:apply-templates/> Millionen
   </xsl:template>
   <xsl:template match="HAUPTSTADT">
      <br/>
         Hauptstadt : <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="KFZ-KENNZEICHEN">
      <br/>
         Kennzeichen : <xsl:apply-templates/>
   </xsl:template>
   <xsl:template match="TEL-VORWAHL">
      <br/>
         Vorwahl : <xsl:apply-templates/>
      <br/><hr/>
   </xsl:template>
</xsl:stylesheet>
```

Vgl. Abschnitt 10.21

❶ Das Element `<xsl:for-each>` erzeugt eine Schleife. Bei jedem zutreffenden Ausdruck des `select`-Attributs wird ein Durchlauf erzeugt. Schleifen stellen in der Praxis eine Möglichkeit dar, um Auswahlen zu treffen. Häufig kommen sie bei Inhaltsverzeichnissen und Registern zum Einsatz, in denen bestimmte Informationen aus dem gesamten Dokument gesammelt werden müssen und neu strukturiert werden sollen.

❷ An dieser Stelle wird die Schleife benutzt, um alle Ländernamen anzuzeigen und einen internen Verweis auf diese an der entsprechenden Stelle im Dokument zu erzeugen. Das Element `<xsl:attribute>` erzeugt dabei Attribute im Elternelement, hier für das Element `<a>`. Das obligatorische Attribut `name` gibt dem Attribut im Zieldokument seinen Namen, hier also das `href`-Attribut, das in HTML das Ziel des Links festlegt. Der Inhalt des `<xsl:attribute>`-Elements wird im Zieldokument zum Wert des `href`-Attributs. Das #-Zeichen wird in HTML genutzt, um auf einen Anker zu verweisen, der sich im selben Dokument befindet. Der Name des Ankers wird hier über den Namen des Landes, also dem Inhalt des `NAME`-Elements festgelegt. Das `<xsl:value-of select="."/>`-Element sorgt auch hier dafür, dass die Werte des Kontextknotens

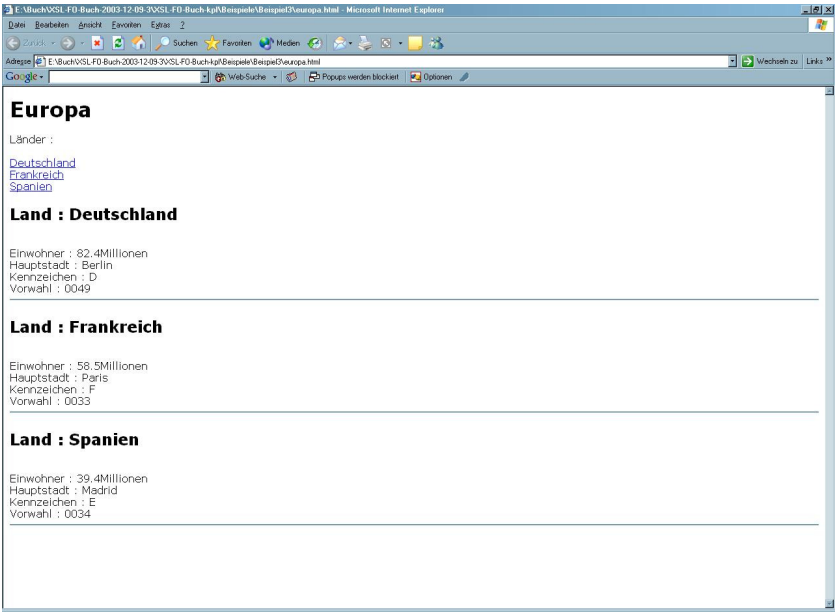
kopiert werden, also diejenigen Werte, die zu dem NAME-Element gehören, das in der Schleife abgefragt wird.

❸ Das an dieser Stelle angewendete Prinzip entspricht der oben erläuterten Vorgehensweise. An dieser Stelle werden die benötigten Anker erzeugt. Auch hier kommt das HTML-Element <a> zum Einsatz. Diesem wird das Attribut name zugeordnet, das den Namen des Ankers, also das Ziel des oben initiierten Links, festlegt. Auch hier wird das <xsl:value-of select="."/>-Element dafür eingesetzt.

Das Ergebnis sieht wie folgt aus:

```
<html>
   <title></title>
   <body>
      <h1>Europa</h1>
      Länder : <br/>
      <br/>
      <a href="#Deutschland">Deutschland</a>                    ❷
      <br/>
      <a href="#Frankreich">Frankreich</a>                      ❷
      <br/>
      <a href="#Spanien">Spanien</a>                            ❷
      <h2> Land :
         <a name="Deutschland"></a>Deutschland                  ❸
      </h2>
      <br/>Einwohner : 82.4Millionen
      <br/>Hauptstadt : Berlin
      <br/>Kennzeichen : D
      <br/>Vorwahl : 0049
      <br/><hr/> <h2> Land :
         <a name="Frankreich"></a>Frankreich                    ❸
      </h2>
      <br/>Einwohner : 58.5Millionen
      <br/>Hauptstadt : Paris
      <br/>Kennzeichen : F
      <br/>Vorwahl : 0033
      <br/><hr/> <h2> Land :
         <a name="Spanien"></a>Spanien                          ❸
      </h2>
      <br/>Einwohner : 39.4Millionen
      <br/>Hauptstadt : Madrid
      <br/>Kennzeichen : E
      <br/>Vorwahl : 0034
      <br/><hr/>
   </body>
</html>
```

Abb. 3-4
Browser-Ansicht



3.7 Bedingungen

Zur Formulierung von Bedingungen stehen in XSLT zwei Elemente zur Verfügung. Das eine ist das `<xsl:if>` und das andere `<xsl:choose>`.

<xsl:if>

Mit dem Element `<xsl:if>` lassen sich Bedingungen festlegen. Das Element verfügt über das obligatorische Attribut `test`, das als Wert eine Musterprüfung verlangt. Anders als in den anderen Programmiersprachen üblich, gibt es in XSLT die Verbindung von einem IF mit einem ELSE nicht. Betrachten wir nun ein kleines Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:template match="EUROPA">
      <html>
         <title></title>
         <body>
         <h1>Europa</h1> Länder :
         <br/><br/>
         <xsl:if test="//LAND/NAME='Deutschland'">                    ❶
            Deutschland ist das bevölkerungsreichste Land in Westeuropa.
         </xsl:if>
         <br/> <br/>
         <xsl:if test="//LAND/NAME='Spanien'">                        ❷
            Spanien ist das sonnigste Land in Westeuropa.
```

```
            </xsl:if>
            <br/> <br/>
            <xsl:if test="//LAND/NAME='Frankreich'">                    ❷
               Frankreich ist das flächengrößte Land in Westeuropa.
            </xsl:if>
            <br/>
            </body>
         </html>
      </xsl:template>
</xsl:stylesheet>
```

❶ Wie zu sehen ist, wird der Inhalt des `<xsl:if>`-Elements dann ausgeführt, wenn das Muster im `test`-Attribut eine Übereinstimmung im Quelldokument findet.

❷ An dieser Stelle wird deutlich, dass mit Hilfe des `<xsl-if>`-Elements die Reihenfolge im Ergebnisdokument geändert werden kann. Während im Quelldokument das Element `<NAME>` mit dem Inhalt Spanien an letzter Stelle steht, wird die Reihenfolge im Ergebnisdokument verändert. Der Prozessor geht also nach der Reihenfolge im Stylesheet vor.

Das Ergebnis sieht nun wie folgt aus:

```
<html>
   <title></title>
   <body>
      <h1>Europa</h1> Länder : <br/><br/>
         Deutschland ist das bevölkerungsreichste Land in West-          ❶
            europa.
      <br/><br/>
         Spanien ist das sonnigste Land in Westeuropa.                   ❷
      <br/><br/>
         Frankreich ist das flächengrößte Land in Westeuropa.            ❷
      <br/>
   </body>
</html>
```

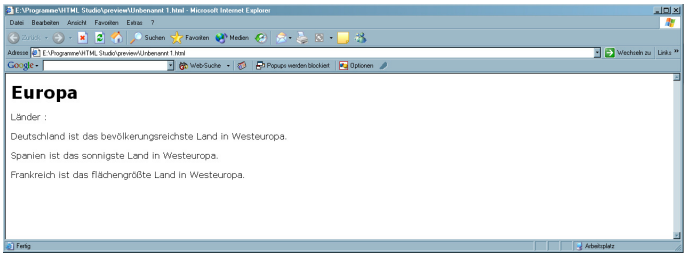


Abb. 3-5
Browser-Ansicht

<xsl:choose>

Das `<xsl:choose>`-Element mit seinen Kindelementen `<xsl:when>` und `<xsl:otherwise>` entspricht dem switch-case-Konstrukt in anderen Programmiersprachen.

Vgl. Abschnitt 10.3

Dieses Konstrukt erlaubt es, mehrere Alternativen anzugeben und schließlich einen Wert, der bei nicht zutreffenden Alternativen stattdessen genommen wird. Das Konstrukt muss mit dem Element `<xsl:choose>` eingeleitet werden. Das Element kann beliebig viele `<xsl:when>`-Kindelemente enthalten und genau ein `<xsl:otherwise>`-Element. Die `<xsl:when>`-Elemente besitzen ebenso wie das `<xsl:if>`-Element ein obligatorisches Attribut `test`. Wenn dessen Muster zutrifft, wird auch hier der Inhalt des Elements in den Ergebnisbaum kopiert. Der eigentliche Unterschied zwischen dem `<xsl:if>`-Element und dem `<xsl:choose>`/`<xsl:when>`-Konstrukt besteht in der Möglichkeit der Verwendung des `<xsl:otherwise>`-Elements, das, wenn keines der `<xsl:when>`-Elemente zutrifft, in Aktion tritt.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
   <title></title>
   <body>
   <h1>Europa</h1> Länder : <br/><br/>
   <xsl:for-each select="//LAND/NAME">                                  ❶
      <xsl:choose>                                                      ❷
         <xsl:when test=".='Frankreich'">                               ❸
            <h2> <xsl:value-of select="."/> ist das flächengrößte Land in
                Westeuropa.</h2> <br/>
         </xsl:when>
         <xsl:otherwise>                                                ❹
            <br/> <xsl:value-of select="."/> ist nicht das flächengrößte
                Land in Westeuropa.
         </xsl:otherwise>
      </xsl:choose>
   </xsl:for-each>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

❶ Die Schleife wird an dieser Stelle benötigt, um den Prozessor dazu zu bewegen, den jeweils aktuellen Knoten in der folgenden Fallunterscheidung zu überprüfen.

❷ Das `<xsl:choose>`-Element umschließt das Konstrukt. Die Elemente `<xsl:when>` und `<xsl:otherwise>` sind nur als Kindelemente des `<xsl:choose>`-Elements erlaubt.

❸ Das `<xsl:when>`-Element wird dann ausgeführt, wenn das Muster im Attribut `test` zutrifft. In diesem Fall wird gefragt, ob der Inhalt des aktuellen Knotens – also `.` – den Wert „Frankreich“ hat. Wenn der Wert

mit dem Muster übereinstimmt, wird der Name des Landes ausgegeben und die Angabe, dass dieses Land das flächengrößte in Westeuropa sei. ❹ Das <xsl:otherwise>-Element kommt dann zum Einsatz, wenn keines der <xsl:when>-Elemente zugetroffen hat.

Als Ergebnis erhalten wir folgenden HTML-Code:

```
<html>
   <title></title>
   <body>
      <h1>Europa</h1> Länder : <br/><br/>
      <br/>Deutschland ist nicht das flächengrößte Land in          ❹
          Westeuropa.
      <h2>Frankreich ist das flächengrößte Land in Westeuropa.      ❸
          </h2><br/>
      <br/>Spanien ist nicht das flächengrößte Land in Westeu-      ❹
          ropa. <br/>
   </body>
</html>
```

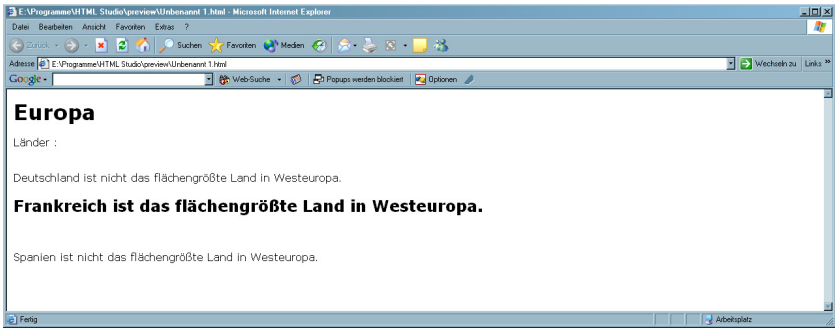


Abb. 3-6
Browser-Ansicht

3.8 Sortierung

Mit Hilfe des Elements <xsl:sort> lassen sich Knoten vor ihrer Ausgabe sortieren. Als Standard ist die Sortierung von Zeichenketten in aufsteigender alphabetischer Reihenfolge vorgegeben. Das Attribut order definiert die Sortierreihenfolge und kann die Werte „ascending“ oder „descending“ einnehmen, während das ebenfalls optionale Attribut data-type festlegt, ob es sich um Zeichenketten („text“) oder Zahlen („number“) handelt. Das <xsl:sort>-Element muss allerdings in ein <xsl:apply-templates>- oder <xsl:for-each>-Element eingebettet sein und darf keinen Inhalt haben.

Im folgenden Beispiel werden die Länder in absteigender alphabetischer Reihenfolge ausgegeben und in einem zweiten Teil der Einwohnerzahl nach geordnet.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
   <title> </title>
   <body>
   <h1>Europa</h1> Länder : <br/><br/>
      <xsl:for-each select="//LAND/NAME">
         <xsl:sort order="descending"/>                                    ❶
         <xsl:value-of select="."/> <br/>
      </xsl:for-each>
      <br/>
      <xsl:for-each select="//LAND/EINWOHNERZAHL">
         <xsl:sort order="descending" data-type="number"/>                 ❷
         <xsl:value-of select="."/> Millionen in
         <xsl:value-of select="../NAME"/> <br/>
      </xsl:for-each>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

❶ Das Element `<xsl:sort>` sorgt dafür, dass alle folgenden Inhalte der Schleife nach dem ersten Buchstaben sortiert, in absteigender alphabetischer Reihenfolge erscheinen.

❷ In diesem zweiten Beispiel wird das `<xsl:sort>` Element dafür genutzt, die Länder nach ihrer Einwohnerzahl geordnet auszugeben. Das Attribut `data-type` muss hierfür auf den Wert `number` gesetzt werden, weil die Grundeinstellung `text` erwartet. Das Muster im `<xsl:value-of>`-Element `"../NAME"` sorgt dafür, dass das `<NAME>`-Element auf der gleichen Ebene wie das Element `<EINWOHNERZAHL>` angesteuert wird. Dadurch wird der Name desjenigen Landes ausgewählt, zu dem diese Einwohnerzahl gehört.

```
<html>
   <title></title>
   <body>
      <h1>Europa</h1> Länder :
      <br/><br/>
      Spanien<br/>                                                         ❶
      Frankreich<br/>                                                      ❶
      Deutschland<br/>                                                     ❶
      <br/>
      82.4 Millionen in Deutschland<br/>                                   ❷
      58.5 Millionen in Frankreich<br/>                                    ❷
      39.4 Millionen in Spanien<br/>                                       ❷
   </body>
</html>
```

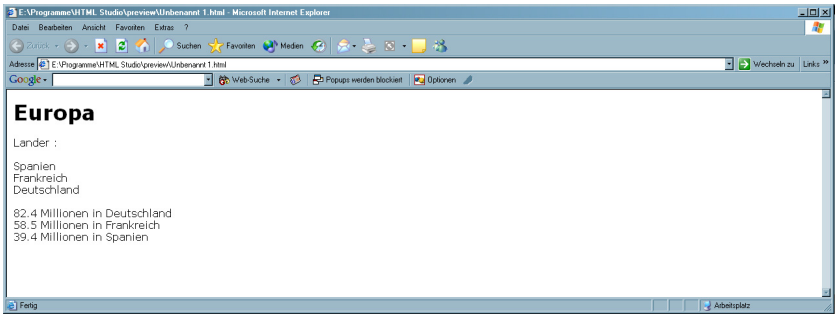


Abb. 3-7
Browser-Ansicht

3.9 Nummerierung

Mit `<xsl:number>` lassen sich Knoten im Ergebnisbaum Sequenzen zuweisen. Es lassen sich so beispielsweise Kapitel, Abschnitte oder einfache Elemente nummerieren. Die Nummerierung in verschiedenen Ebenen ist ebenfalls möglich und auch die Formatierung der Ausgabe lässt sich bestimmen. So kann man mit dem Attribut `format` eine Nummerierung mit arabischen oder römischen Zahlen oder mit Buchstaben generieren.

Vgl. Abschnitt 10.6, 10.7

Das Element besitzt drei in der Praxis gebräuchliche Attribute `count`, `level` und `format`. Das Attribut `count` bestimmt, welche Knoten gezählt werden sollen, `level` bestimmt die Zählweise und `format` das Format der Anzeige.

Auch hier ein kleines Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
      <title></title>
      <body>
         <h3>
            <xsl:apply-templates/>
         </h3>
      </body>
   </html>
</xsl:template>
<xsl:template match="NAME">
   <br/>
   <xsl:number/>                                              ❶
   <xsl:text> </xsl:text>                                     ❻
   <xsl:value-of select="."/> <br/>
</xsl:template>
<xsl:template match="EINWOHNERZAHL">
   <br/>
```

```
   <xsl:number level="multiple" format="1.1" count="LAND | LAND/        ❷
       EINWOHNERZAHL"/>
   <xsl:text> </xsl:text>
   <xsl:value-of select="."/> Millionen Einwohner <br/>
</xsl:template>
<xsl:template match="HAUPTSTADT">
   <br/>
   <xsl:number level="multiple" format="I.I" count="LAND | LAND/        ❸
       HAUPTSTADT"/>
   <xsl:text> Hauptstadt : </xsl:text>                                  ❷
   <xsl:value-of select="."/> <br/>
</xsl:template>
   <xsl:template match="KFZ-KENNZEICHEN">
   <br/>
   <xsl:number level="multiple" format="A.a" count="LAND | LAND/        ❹
       KFZ-KENNZEICHEN"/>
   <xsl:text> Kennzeichen : </xsl:text>
   <xsl:value-of select="."/> <br/>
</xsl:template>
<xsl:template match="TEL-VORWAHL">
   <br/>
   <xsl:number level="any" format="i" count="TEL-VORWAHL"/>             ❺
   <xsl:text> Kennzeichen : </xsl:text>
   <xsl:value-of select="."/> <br/><hr/>
</xsl:template>
</xsl:stylesheet>
```

❶ In diesem Template wird das `<xsl:number>`-Element dazu benutzt, die Ländernamen zu nummerieren. In der Grundeinstellung ohne Attribute erfolgt die Nummerierung in arabischen Zahlen. Gezählt werden dabei alle vorkommenden Elemente.

❷ Das `<xsl:number>`-Element wird hier mit den drei wichtigsten Attributen eingesetzt. Das Attribut `level` gibt an, ob die Zählung einfach `single` oder aber `multiple`, also in der Form (1., 1.1, 1.2, ...) erfolgen soll. Das Attribut `format` legt die Formatierung fest. Mögliche Werte sind 1, a, A, i und I, die kombiniert werden können. Jede Ebene wird mit Hilfe eines Zeichens getrennt. Hier wird der Punkt benutzt, denkbar wären aber auch andere Trennzeichen. Das `count`-Attribut legt fest, auf welcher Ebene welche Elemente gezählt werden sollen. Das Trennzeichen für die Ebenen ist das |-Zeichen.

❸ Das Attribut `format` hat hier den Wert `I.I`, was dafür sorgt, dass die Nummerierung in großen römischen Zahlen erfolgt.

❹ Hier wurde die Zählung kombiniert. In der ersten Ebene werden große Buchstaben verwendet, in der zweiten kleine.

❺ Der Wert `any` des `level`-Attributs bewirkt eine Zählung der Knoten im gesamten Dokument, unabhängig von deren Position in der Struktur.

Der Wert `i` des `format`-Attributs bewirkt die Zählung in kleinen römischen Zahlen.

❻ Das Element `<xsl:text>` erlaubt es, Text in das Zieldokument zu kopieren. Dies ist an dieser Stelle zwar auch ohne dieses Element möglich, allerdings wird es im Zusammenhang mit XSL-FO oft gebraucht, weil damit beispielsweise Leerzeichen als Bestandteil des auszugebenden Textes genau kontrolliert werden können.

Das Ergebnis sieht wie folgt aus:

```
<html>
   <title></title>
   <body>
      <h3><br/>
      1 Deutschland <br/>                                          ❶
      1.1 82.4 Millionen Einwohner <br/>                           ❷
      I.I Hauptstadt : Berlin <br/>                                ❸
      A.a Kennzeichen : D <br/>                                    ❹
      i Kennzeichen : 0049<hr/> <br/>                              ❺
      1 Frankreich <br/>                                           ❶
      2.1 58.5 Millionen Einwohner <br/>                           ❷
      II.I Hauptstadt : Paris <br/>                                ❸
      B.a Kennzeichen : F <br/>                                    ❹
      ii Kennzeichen : 0033<hr/> <br/>                             ❺
      1 Spanien <br/>                                              ❶
      3.1 39.4 Millionen Einwohner <br/>                           ❷
      III.I Hauptstadt : Madrid <br/>                              ❸
      C.a Kennzeichen : E <br/>                                    ❹
      iii Kennzeichen : 0034<hr/>                                  ❺
      </h3>
   </body>
</html>
```

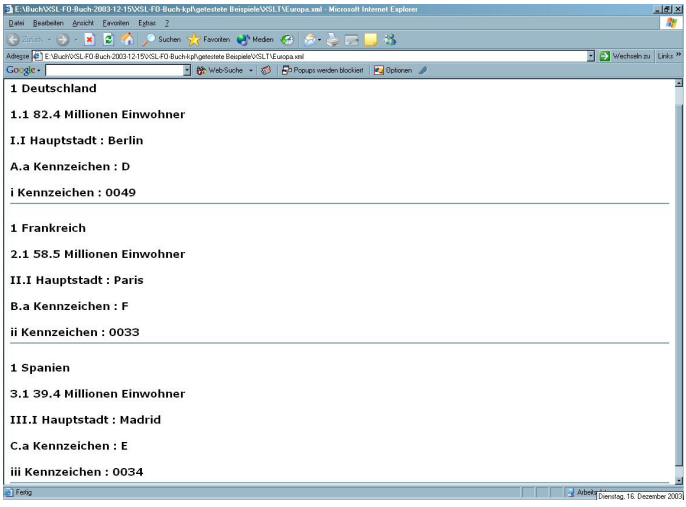


Abb. 3-8
Browser-Ansicht

3.10 Variablen und Parameter

Mit <xsl:variable> lassen sich Variablen definieren. Wird das Element als Top-Level-Element verwendet, so ist dessen Gültigkeitsbereich global. Befindet sich die Variable allerdings innerhalb eines Template-Rumpfes, so handelt es sich um lokale Variablen. Globale Variablen sind sogar auch dann gültig, wenn sie in andere Stylesheets importiert werden. Der Variablen kann man sowohl mit Hilfe des select-Attributs als auch mit dem Inhalt des Elements den gewünschten Wert zuweisen, der allerdings ein String (Zeichenfolge) sein muss. Der Aufruf des Werts erfolgt mittels des $-Zeichens, das dem Namen der Variablen vorangestellt werden muss.

Dieses Variablen-Konzept ist für XSL-FO besonders wichtig. Denn in den Stylesheets gibt es eine Vielzahl von Informationen, die so zentral verwaltet werden können. Als Beispiel sind hier Schriftgrößen zu nennen. Generell nimmt die Bedeutung solcher Variablen mit dem Umfang des Stylesheets zu. Die Verwendung von <xsl:param> entspricht in den später folgenden XSL-FO Stylesheets der von <xsl:variable>, die zu verwendenden Attribute sind in beiden Elementen identisch. Auf die kleinen Unterschiede in der Funktion beider Elemente wird in diesem Buch nicht eingegangen.

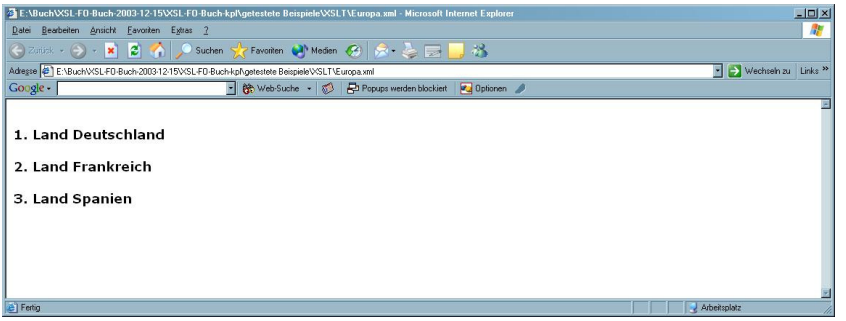
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:variable name="Struktur">                                    ❶
      <html>
      <title></title>
      <body>
         <h3>
            <xsl:apply-templates select="//NAME"/>                  ❹
         </h3>
      </body>
      </html>
   </xsl:variable>
   <xsl:template match="/">
      <xsl:copy-of select="$Struktur"/>                             ❷
   </xsl:template>
   <xsl:template match="NAME">
      <xsl:variable name="Nation" select="'Land'"/>                 ❸
      <br/>
      <xsl:number level="any" format="1."/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="$Nation"/>                              ❸
      <xsl:text> </xsl:text>
      <xsl:value-of select="."/>
      <br/>
```

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

❶ Das Element `<xsl:variable>` hat das obligatorische Attribut, das den Namen der Variablen als Wert hat. Dieser Name kann später beliebig oft aufgerufen werden. Da das Element `<xsl:variable>` hier ein Top-Level-Element ist, d. h. ein Kindelement des Elements `<xsl:stylesheet>` ist, handelt es sich um eine globale Variable, die im gesamten Stylesheet gültig ist. Der Inhalt der Variable – in diesem Fall die HTML-Struktur – kann über den Aufruf an beliebiger Stelle hineinkopiert werden.
❷ Das Element `<xsl:copy-of>` erlaubt, tiefe Strukturen in den Ergebnisbaum hineinzukopieren. Im Unterschied dazu wird das Element `<xsl:copy>` benutzt, um flache Strukturen zu kopieren. Diese Elemente sind für XSLT-Stylesheets oft wichtig. In XSL-FO kommen sie in der Praxis sehr selten zum Einsatz und werden hier nicht weiter ausgeführt. Das $-Zeichen vor dem Variablennamen ist unbedingt erforderlich, da so dem Prozessor mitgeteilt wird, dass es sich um eine Variable handelt und nicht um ein Element.
❸ Hier handelt es sich um eine lokale Variable, die nur innerhalb des Templates gültig ist, in dem sie definiert ist. Da sie das Attribut `select` enthält, wird der Wert dieses Attributs als Ersetzungstext verwendet. Das heißt, dass der Wert des `select`-Attributs den Inhalt der Variable festlegt. Da es sich beim Inhalt der Variable „Nation“ um einen reinen Text – hier `'Land'` – und keine Struktur handelt, erfolgt der Aufruf mittels des `<xsl:value-of>`-Elements. Auch hier muss das $-Zeichen dem Variablennamen vorangestellt werden.
❹ Das hier gebrauchte `<xsl:apply-templates>`-Element hat ein `select`-Attribut, das die Ausgabe genau steuert. Es werden hier nur diejenigen Elemente gesucht und in den Ergebnisbaum kopiert, die im Element `<NAME>` enthalten sind.

```
<html>                                                    ❷
   <title></title>
   <body>
      <h3><br/>
      1. Land Deutschland<br/>                            ❸
      2. Land Frankreich<br/>                             ❸
      3. Land Spanien                                     ❸
   </h3>
   </body>
</html>
```

Abb. 3-9
Browser-Ansicht



3.11 Modularisierung

Die Elemente `<xsl:import>` und `<xsl:include>` erlauben es, Stylesheets oder Stylesheet-Fragmente zu importieren. Sie unterscheiden sich im Hinblick auf die Import-Präzedenz (Rangordnung). Bei der Verwendung von `<xsl:import>` wird dem importierten gegenüber dem importierenden Stylesheet eine niedrigere Präzedenz gegeben. Sollten zwei Templates auf das gleiche Element verweisen, so wird dasjenige ausgeführt, welches nicht importiert wurde. Bei der Verwendung von `<xsl:include>` kommen dagegen die Standard-Präzedenzregeln zum Zuge, wie sie auch innerhalb eines Dokuments gelten. Das heißt, es wird dasjenige Template angewendet, das präziser (z. B. unter Verwendung eines absoluten Pfades) auf das Element in seinem XPath-Ausdruck verwiesen hat. Sollte dies gleichwertig sein, wird das Letztere verwendet. Die beiden leeren Elemente besitzen lediglich ein Attribut `href`, dessen Wert die URI zum einzubeziehenden Stylesheet enthält.

Ein solcher Aufruf könnte wie folgt aussehen:

```
<xsl:include href="style.xsl"/>
```

4 XPath – Einführung in die Technik des Zugriffs auf die XML-Strukturen

Bei XPath handelt es sich um die Navigationssprache für XML-Dokumente und ist damit das zweite Teilkonzept innerhalb des XSL-Konzepts. Sie erlaubt, auf Strukturen und die Inhalte eines XML-Dokuments zuzugreifen. Neben der bloßen Navigation besteht auch die Möglichkeit, mittels definierter Funktionen bestimmte Operationen durchzuführen. Wir werden uns hier auf die für unsere Zwecke gebräuchlichsten und wichtigsten Funktionen beschränken.

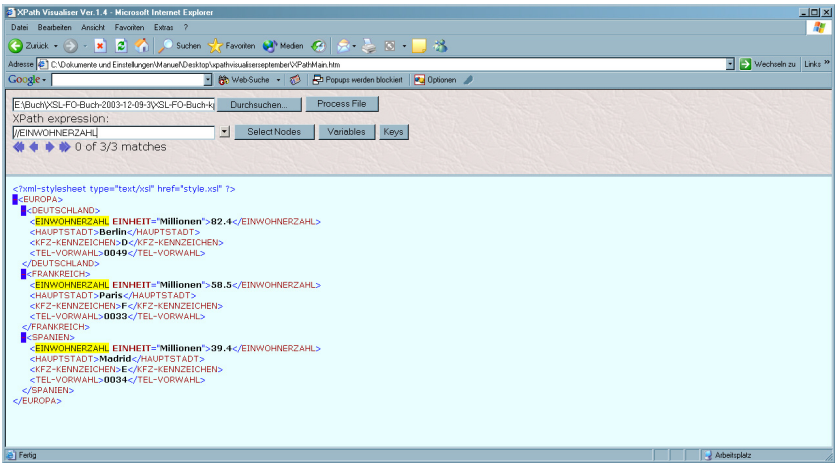
Vgl. Abschnitt 1.1

Für das Testen von XPath-Ausdrücken empfehlen wir das kostenfreie Programm XPathvisualizer. Dieses Programm ist unter dieser Adresse http://www.vbxml.com/xpathvisualizer/ allgemein verfügbar. XPath-Ausdrücke können in der Praxis sehr komplex und deswegen auch sehr fehleranfällig sein. Meist wirken sich Fehler in XPath-Ausdrücken dadurch aus, dass das Ziel nicht gefunden wird, d. h. kein Resultat erscheint oder ein Resultat erscheint, auf das man nicht abgezielt hat. Innerhalb des Visualisierers werden die Ziele von XPath-Ausdrücken im Kontext des XML-Dokuments angezeigt. So lässt sich einfach entscheiden, ob der Ausdruck zu dem gewünschten Ziel führt.

Der folgende Bildschirm zeigt eine beispielhafte Nutzung des XPathvisualizer.

Autor: Manuel Montero Pineda

Abb. 4-1
XPathvisualizer



4.1 Die Knotentypen

Die Bestandteile von XML-Dokumenten werden als Knoten bezeichnet. Dabei lassen sich folgende 7 Knotentypen unterscheiden, deren Namen selbsterklärend sind:

- Root node (Wurzelknoten): Dieser darf nicht mit dem Wurzelelement selbst verwechselt werden. Vielmehr ist dieser der virtuelle Elternknoten (parent node) des Wurzelelements.
- Element node (Elementknoten)
- Attribute node (Attributknoten)
- Text node (Textknoten)
- Namespace node (Namensraumknoten)
- Processing instruction node (Verarbeitungsanweisungsknoten)
- Comment node (Kommentarknoten)

Jeder dieser Knoten lässt sich mit XPath ansteuern und ermöglicht so eine Weiterverarbeitung mit XSL.

4.2 Die Achsen

Die Navigation in einem XML-Dokument mittels XPath erfolgt von einem Kontextknoten (context node) – in den folgenden Darstellungen als SELF bezeichnet – aus. Der Kontextknoten ist immer der jeweilige Ausgangspunkt, in dem sich der XSLT-Prozessor gerade befindet. Die Achsennamen bezeichnen hierbei – wie bereits in XML gebräuchlich – Verwandtschaftsverhältnisse. In den folgenden Darstellungen werden elf der insgesamt dreizehn unterschiedlichen Achsen veranschaulicht. Neben den dargestellten Achsen, welche die Navigation durch die Elementkno-

ten eines Dokumentbaumes erlauben, gibt es noch die Attributknoten und den Namensraumknoten eines Elements, die mittels der Schlüsselwörter `attribute` bzw. `namespace` angesprochen werden.
Die Self-Achse enthält den Kontextknoten.

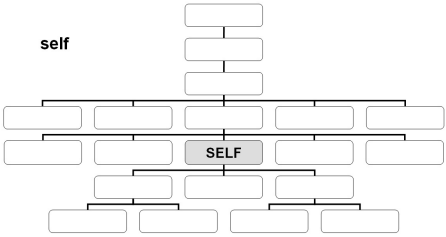


Abb. 4-2
Self-Achse

Die Kind-Achse enthält alle Kindelemente des Kontextknotens.

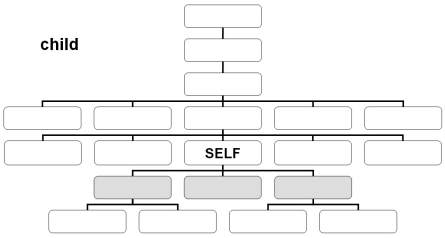


Abb. 4-3
Child-Achse

Die Descendant-Achse enthält alle Nachkommen des Kontextknotens.

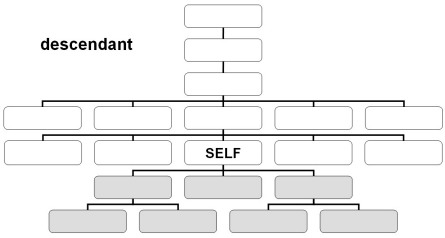


Abb. 4-4
Descendant-Achse

Die Descendant-or-self-Achse enhält alle Nachkommen und den Kontextknoten selbst.

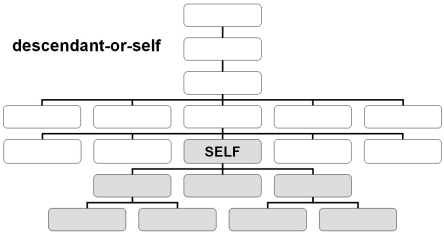
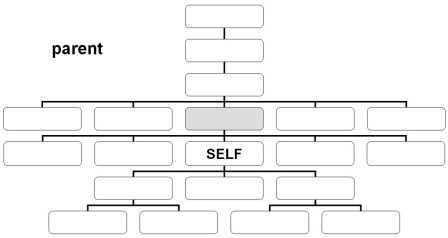


Abb. 4-5
Descendant-or-self-Achse

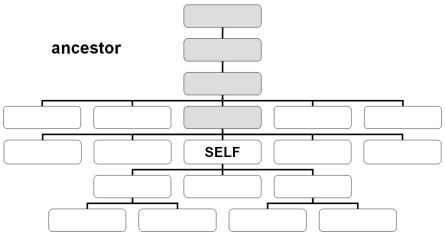
Die Parent-Achse enthält das Elternelement des Kontextknotens.

Abb. 4-6
Parent-Achse



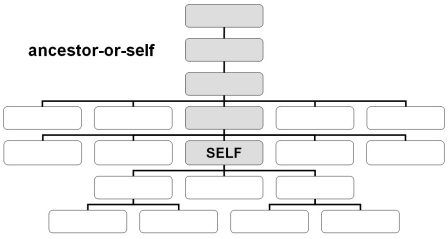
Die Ancestor-Achse enthält alle Vorfahren des Kontextknotens.

Abb. 4-7
Ancestor-Achse



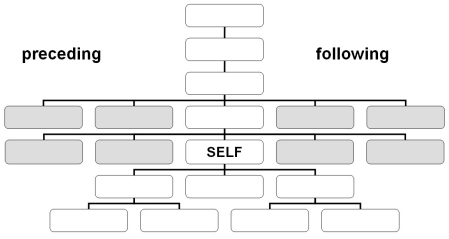
Die Ancestor-or-self-Achse enthält alle Vorfahren und den Kontextknoten selbst.

Abb. 4-8
Ancestor-or-self-Achse



Die Preceding-Achse enthält alle Vorgänger des Kontextknotens, die keine Vorfahren sind. Die Following-Achse enthält alle Nachfolger des Kontextknotens, die keine Nachfahren sind.

Abb. 4-9
Preceding-Achse und Following-Achse



4.3 Die Adressierung

Unter der Adressierung ist eine Formulierung zu verstehen, die es dem Prozessor erlaubt, Knoten oder Knotenmengen anzusteuern bzw. diese zu selektieren.

Solche Adressierungen werden auch Lokalisierungspfade genannt und können in unterschiedlichen Erscheinungsformen auftreten:

- in verkürzter oder ausführlicher Schreibweise,
- als relative oder absolute Lokalisierungspfade.

4.3.1 Verkürzte oder ausführliche Schreibweise

Vgl. Abschnitt 12.1

Für die hier bereits genannten Knoten und Achsen existiert eine verkürzte und eine ausführliche Schreibweise. Beide Schreibweisen werden in der Praxis gemischt verwendet. Die verkürzte ist immer dann üblich, wenn es sich um sehr häufig angesteuerte Knoten oder Achsen handelt, während die ausführliche bei den selteneren Knoten und Achsen üblich ist. In den folgenden Abschnitten werde ich die ausführliche Schreibweise darstellen und dazu die verkürzte, wenn sie häufig verwendet wird. Im Referenzteil finden sich alle ausführlichen und verkürzten Schreibweisen mit einer kurzen Erläuterung.

Ein Beispiel in ausführlicher Notation:

```
/child::buch/child::autor/child::name/attribut::nachname
```

in verkürzter Notation:

```
/buch/autor/name/@nachname
```

In der verkürzten Notation wird auf die Achsenbezeichnung `child::` verzichtet und der Attributknoten mit dem Zeichen `@` bezeichnet. Hierzu in den späteren Beispielen mehr.

4.3.2 Relative und absolute Pfade

Es ist möglich, einen Lokalisierungspfad relativ oder absolut (vom Wurzelelement ausgehend) zu schreiben.

Genau genommen gehen die absoluten Lokalisierungspfade vom Wurzelknoten aus – das ist der Knoten, der direkt oberhalb des Wurzelelements liegt. Würde man diese Unterscheidung nicht machen und lediglich vom Wurzelelement ausgehen, dann könnte man beispielsweise Prozessinstruktionen oder Kommentare, die außerhalb des Wurzelelements liegen, nicht adressieren können.

Die Lokalisierungspfade setzen sich aus mehreren Einzelschritten zusammen, die durch Schrägstriche voneinander getrennt werden.

Ein Beispiel:

```
/child::EUROPA/child::LAND/child::NAME
```

In diesem Fall handelt es sich um einen absoluten Pfad, der vom Wurzelknoten ausgeht. Der Wurzelknoten wird mit dem ersten Schrägstrich angegeben. Von dort aus wird das Wurzelelement <EUROPA> selektiert, welches ein Kindelement <LAND>, das wiederum ein Kindelement <NAME> haben muss, damit es zu einem Treffer kommt. Die Achsen werden unter Angabe des Namens gefolgt von zwei Doppelpunkten angegeben. Im Falle von child:: kann die Achsenangabe auch entfallen. Folglich ist der Pfad /EUROPA/LAND/NAME gleichwertig (in verkürzter Schreibweise).

Relative Pfade hingegen erlauben die Suche nach einer festgelegten Kombination im gesamten Dokument. Sie werden mit doppelten Schrägstrichen eingeleitet.

Beispiel:

```
//child::LAND/child::NAME
```

Der Prozessor sucht im gesamten Dokument nach dem Element <LAND> und dessen Kindelement <NAME>. Ob es also zu einem Treffer kommt, hängt nun davon ab, ob diese Kombination im Dokument vorkommt.

4.3.3 XPath-Beispiele im Kontext eines XSLT-Stylesheets

Es folgt ein XSLT-Beispiel mit XPath-Adressierungen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">                                              ❶
   <html>
      <title></title>
      <body>
         <h3>
            <xsl:apply-templates/>
         </h3>
      </body>
   </html>
</xsl:template>
<xsl:template match="EUROPA">                                         ❷
   <xsl:if test="LAND/NAME">                                          ❷
      TREFFER : /EUROPA/LAND/NAME
      <br/>
   </xsl:if>
   <xsl:if test="LAND/EINWOHNERZAHL">                                 ❸
      TREFFER : /EUROPA/LAND/EINWOHNERZAHL
      <br/>
   </xsl:if>
```

```
   <xsl:if test="LAND/HAUPTSTADT">                                          ❹
      TREFFER : /EUROPA/LAND/HAUPTSTADT
      <br/>
   </xsl:if>
   <xsl:if test="child::EINWOHNERZAHL">                                     ❺
      TREFFER : /EUROPA/EINWOHNERZAHL
      <br/>
   </xsl:if>
   <xsl:if test="descendant::EINWOHNERZAHL">                                ❻
      TREFFER : /EUROPA/descendant::EINWOHNERZAHL
      <br/>
   </xsl:if>
   <xsl:if test="LAND/NAME/following-sibling::EINWOHNERZAHL">               ❼
      TREFFER : /EUROPA/LAND/NAME/following-sibling::EINWOHNERZAHL
      <br/><br/><br/><br/>
   </xsl:if>
   <xsl:if test="LAND/NAME/following-sibling::EINWOHNERZAHL/                ❽
       parent::LAND/descendant::HAUPTSTADT">
      TREFFER : /EUROPA/LAND/NAME/following-sibling::EINWOHNERZAHL/
          parent::LAND/descendant::HAUPTSTADT
      <br/><br/><br/><br/>
   </xsl:if>
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="LAND">                                                 ❾
   <xsl:if test="HAUPTSTADT">                                               ❾
      TREFFER : /EUROPA/LAND/HAUPTSTADT
      <br/>
   </xsl:if>
   <xsl:if test="parent::EUROPA">                                           ❿
      TREFFER : /EUROPA/LAND/parent::EUROPA
      <br/>
   </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

❶ Das Zeichen / bezeichnet den Wurzelknoten.

❷ Das Element `<xsl:template>` bezieht sich auf das Wurzelelement `<EUROPA>`. Für die nun nachfolgenden `<xsl:if>`-Elemente des Templates ist dies der Kontextknoten. Das heißt, dass alle nun folgenden Lokalisierungspfade von diesem Ausgangspunkt ausgehen. Im ersten Beispiel also `LAND/NAME`. Da es ein Kindelement `<LAND>` von `<EUROPA>` gibt, das selbst ein Kindelement `<NAME>` hat, werden die Anweisungen, die sich innerhalb des `<xsl:if>`-Elements befinden, ausgeführt. Es erscheint in der Anzeige der absolute Pfad: `/EUROPA/LAND/NAME`.

❸ Ein weiteres Beispiel zur Demonstration der Position des Kontextknotens.

❹ Ein weiteres Beispiel zur Demonstration der Position des Kontextknotens.
❺ Hier erfolgt keine Ausgabe, weil es vom Kontextknoten <EUROPA> aus betrachtet kein Kindelement namens <EINWOHNERZAHL> gibt. Hier hätte es auch genügt, EINWOHNERZAHL zu schreiben, da nach der Konvention dieses stellvertretend für child::EINWOHNERZAHL stehen kann.
❻ Hier erfolgt wieder eine Ausgabe, da es einen descendant (Nachkommen) <EINWOHNERZAHL> von <EUROPA> aus gibt.
❼ An dieser Stelle sucht der Prozessor nach einem nachfolgenden Geschwisterelement (following-sibling::) <EINWOHNERZAHL> von <NAME>, das ebenfalls <LAND> als Elternelement hat. Da das Element <EINWOHNERZAHL> tatsächlich nach <NAME> steht, erfolgt die Ausgabe.
❽ Das hier gezeigte Suchmuster soll zeigen, dass die Komplexität beliebig zu erhöhen ist. Hier wird nach einem Element gesucht, das ein Kindelement mit dem Namen <LAND> vom Kontextknoten <EUROPA> ist. Dieses soll wiederum ein Kindelement <NAME> enthalten, welches ein nachfolgendes Geschwisterelement <EINWOHNERZAHL> hat. Bis hierher entspricht diese Abfrage der vorangegangenen und wir wissen, dass diese zutrifft. Das Element <EINWOHNERZAHL> soll nun als Elternelement das Element <LAND> haben, welches einen Nachfahren <HAUPTSTADT> haben soll. Wie in der Ausgabe zu sehen, treffen alle diese Voraussetzungen zu.
❾ Das neue Template sorgt an dieser Stelle dafür, dass der Kontextknoten ein anderer ist. Nun beziehen sich die Suchmuster innerhalb des Templates auf <LAND>. An dieser Stelle wird gefragt, ob das Element <LAND> ein Kindelement <HAUPTSTADT> hat.
❿ Hier wird nach einem Elternelement <EUROPA> des Kontextknotens <LAND> gesucht.

Wir erhalten nun folgendes Ergebnis:

```
<html>
   <title></title>
   <body>
      <h3>
      TREFFER : /EUROPA/LAND/NAME                                          ❷
      <br>
      TREFFER : /EUROPA/LAND/EINWOHNERZAHL                                 ❸
      <br>
      TREFFER : /EUROPA/LAND/HAUPTSTADT                                    ❹
      <br>
      TREFFER : /EUROPA/descendant::EINWOHNERZAHL                          ❻
      <br>
      TREFFER : /EUROPA/LAND/NAME/following-sibling::EINWOHNERZAHL         ❼
      <br><br><br><br>
      TREFFER : /EUROPA/LAND/NAME/following-sibling::EINWOHNERZAHL/        ❽
          parent::LAND/descendant::HAUPTSTADT
```

```
    <br><br><br><br>
    TREFFER : /EUROPA/LAND/HAUPTSTADT                    ❾
    <br>
    TREFFER : /EUROPA/LAND/parent::EUROPA                ❿
    <br>
    TREFFER : /EUROPA/LAND/HAUPTSTADT                    ❾
    <br>
    TREFFER : /EUROPA/LAND/parent::EUROPA                ❿
    <br>
    TREFFER : /EUROPA/LAND/HAUPTSTADT                    ❾
    <br>
    TREFFER : /EUROPA/LAND/parent::EUROPA                ❿
    <br>
    </h3>
  </body>
  </html>
```

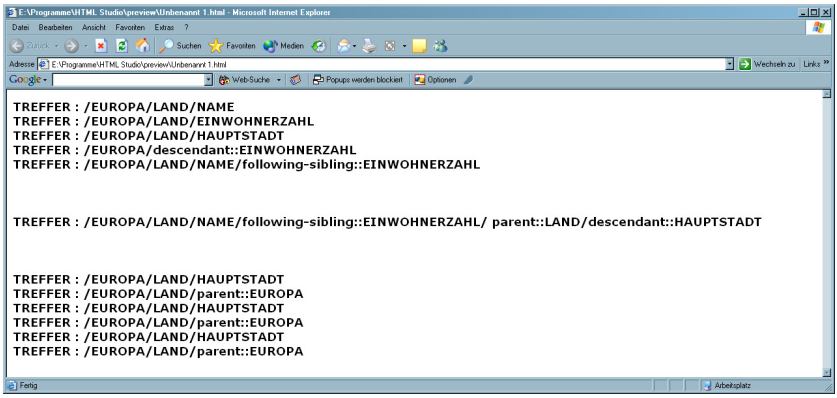


Abb. 4-10
Browser-Ansicht

Zur weiteren Einübung hier eine kleine Sammlung von Pfaden:

- `child::EUROPA` gibt die Kindelemente vom Element `<EUROPA>` zurück.
- `child::node()` gibt die Kindelemente des Kontextknotens zurück.
- `attribute::EINHEIT` ist die ausgeschriebene Schreibweise für das gebräuchliche `@EINHEIT` und gibt den Inhalt des Attributs `EINHEIT` des Kontextknotens zurück.
- `descendant::EUROPA` gibt die Abkömmlinge vom Element `<EUROPA>` zurück.
- `self::EUROPA` gibt sich zurück, falls der Kontextknoten gerade das Element `<EUROPA>` ist.
- `/` gibt das Wurzelelement zurück.
- `//*/@*` gibt beliebige Attribute eines beliebigen Elements zurück.
- `/child::comment()` gibt alle Kommentarknoten zurück, die Kinder des Wurzelknotens sind.

4.4 Die XPath-Prädikate und -Funktionen

Vgl. Abschnitt 12.3

Mit Hilfe von Prädikaten und Funktionen ist eine beliebige Selektion von Knoten in einem XML-Dokument möglich. Für einen kleinen Vorgeschmack auf die potenzielle Komplexität dieser Ausdrücke folgen die wichtigsten Prädikate und Funktionen mit Beispielen.

Prädikate

Folgende, aus anderen Programmiersprachen bekannte Prädikate sind in XPath erlaubt:

- Arithmetische Ausdrücke: `+`, `·`, `*`, `div`, `mod`.
- Boolesche Ausdrücke: `or` (abgekürzt `|`), `and`, `not()` (abgekürzt `!`).
- Vergleichsoperatoren: `<` (Schreibweise: `<` für `<`), `>`, `<=`, `>=`, `=`.

Funktionen

Es gibt eine Vielzahl von Funktionen, die auch in die verkürzte Notation Einzug gehalten haben. Zu den wichtigsten zählen:

- `last()`, Position des letzten Elements in einer Folge von gleichen Elementen.
- `position()`, Position des aktuellen Elements in einer Folge von gleichen Elementen.
- `count(node-set)`, Anzahl gleicher Elemente in einer Folge.
- `name(node-set)`, Name des selektierten Knotens.
- `string(object)`, konvertiert das Objekt in eine Kette von Textzeichen.
- `starts-with (String1, String2)`, gibt `True` (zutreffend) zurück, wenn die Zeichenkette 1 mit der Zeichenkette 2 beginnt. Ein zutreffendes Beispiel ist `starts-with(ABC, A)`, ein unzutreffendes `starts-with(ABC, AX)`.

Die übrigen Funktionen befinden sich im Abschnitt 12.3.

Zur weiteren Einübung hier eine kleine Sammlung von Pfaden kombiniert mit Funktionen:

- `child::EINWOHNERZAHL[attribute::EINHEIT="Millionen"][position ()="3"]` entspricht abgekürzt `EINWOHNERZAHL[@EINHEIT="Millionen"] [3]` und bedeutet, dass das Element `<EINWOHNERZAHL>` selektiert wird, dessen Attribut `EINHEIT` den Wert „Millionen“ hat **und** das dritte in einer Folge gleicher Elemente ist.
- `A[preceding-sibling::table][1]`. Hier wird ein Element `<A>` gesucht, das ein vorangegangenes Geschwisterelement `<table>` hat und

erstes Element einer Folge von <A>-Elementen ist, mit anderen Worten das erste Element <A> ist, das unmittelbar auf eine Tabelle folgt.

- `Beitrag[@ID='Referenz']//table/tbody/tr/td[position()=1] | Beitrag[@ID='Referenz']//table/tbody/tr[position()>1]/td` ist ein Lokalisierungspfad aus dem Stylesheet für dieses Buch. Der Zweck ist die unterschiedliche Behandlung von Tabellen, abhängig von ihrer Position. Es wird abgefragt, ob die Tabelle – hier das Element <table> – ein beliebiger //-Nachfahre von <Beitrag> ist, das als Attributwert `Referenz` des Attributs `ID` hat. Zusätzlich wird gefordert, dass der aktuelle Knoten das erste <td>-Element von <tr> ist, das ein Kindelement von <tbody> ist und ein Enkelelement von <table>. Oder es handelt sich um ein <td>-Element mit den gleichen Voraussetzungen wie eben, nur dass es an beliebiger Stelle eines Elternelements <tr> ist, das aber nicht an erster Stelle sein darf.
- `<xsl:variable name="Ebene" select="count( ancestor-or-self::Beitrag | ancestor-or-self::Abschnitt )" />` Hier werden die „Ebenen“ im Buch gezählt und als Wert einer Variable mit dem Namen `Ebene` übergeben. Die XPath-Funktion zählt alle Knoten, auf die die Eigenschaften `ancestor-or-self::Beitrag` oder `ancestor-or-self::Abschnitt` zutreffen. Im Buch befindet sich z. B. die Überschrift zu diesem Kapitel im folgenden Pfad: `/Beitrag/Abschnitt/Titel`, d. h., diesem Titel wird der Wert `2` zugewiesen, da es einen Knoten gibt, der `Beitrag` und einen, der `Abschnitt` heißt, und von denen beiden er abstammt. Dieser Ausdruck wird benötigt, um im Inhaltsverzeichnis den Überschriften mit gleichem Elementnamen unterschiedliche Schriftgrößen auf den beiden Ebenen zuzuweisen.

5 XSL-Funktionen

Dieses Kapitel beschäftigt sich mit den wichtigsten XSL-Funktionen, ungeachtet, ob es sich dabei um XSLT- oder XPath-Funktionen handelt. Im Referenz-Teil befinden sich jeweils kurze Beschreibungen zu sämtlichen XSL-Funktionen. Hier werden wir lediglich auf die Funktionen näher und mit beispielhaften Anwendungen eingehen, die in XSL-FO-Stylesheets typischerweise und häufig benutzt werden.

Vgl. Abschnitt 11.2, 12.3

Um einen ersten Überblick zu geben, welche Funktionen welcher Recommendation zuzuordnen sind, folgt hier eine Liste sämtlicher Funktionen:

Zu XSLT gehören:

- `current()`
- `document()`
- `element-available()`
- `format-number()`
- `function-available()`
- `generate-id()`
- `key()`
- `unparsed-entity-uri()`

Zu XPath gehören thematisch geordnet:

- Knotenmengen-Funktionen
  - `count()`
  - `id()`
  - `last()`
  - `name()`
  - `namespace-uri()`
  - `position()`
- Zeichenketten-Funktionen
  - `concat()`

Autor: Manuel Montero Pineda

- `contains()`
- `normalize-space()`
- `starts-with()`
- `string()`
- `string-length()`
- `substring()`
- `substring-after()`
- `substring-before()`
- `translate()`

❑ Zahlen-Funktionen

- `ceiling()`
- `floor()`
- `number()`
- `round()`
- `sum()`

❑ Boolesche Funktionen

- `boolean()`
- `false()`
- `lang()`
- `not()`
- `true()`

5.1 generate-id()

Die XSLT-Funktion `generate-id()` weist einem Knoten eine eindeutige Kennung in Form einer Zeichenfolge (string) zu.

```
string generate-id(node)
```

Vgl. Abschnitt 10.19

Die Funktion wird im Zusammenhang mit dokumentinternen Links benötigt. Dabei wird dem Zielknoten ein Anker gegeben, der mit Hilfe der Funktion generiert wird, und dieser wird an anderer Stelle als Ziel angegeben. Da der automatisch generierte Anker für jeden Knoten eindeutig ist, ist es nicht notwendig, diese Kennung zu kennen, sondern es ist ausreichend, ebenfalls mittels der Funktion diese zu referenzieren.

Ein kleines HTML-Beispiel:

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

```
        <body>
           <xsl:for-each select="//NAME">
              <h3>
              <a href="#{generate-id()}">                              ❶
                 <xsl:value-of select="."/>
              </a>
              </h3>
           </xsl:for-each>
           <xsl:apply-templates select="//NAME"/>
        </body>
     </html>
  </xsl:template>
  <xsl:template match="NAME">
     <p>
     <a name="{generate-id(.)}">                                       ❷
        <xsl:value-of select="."/>
     </a>
     </p>
  </xsl:template>
</xsl:stylesheet>
```

❶ An dieser Stelle werden die internen Verweise generiert, der XSLT-Prozessor weist dabei jedem Knoten eine eindeutige Kennung zu.
❷ Zum Vergleich sieht man hier die Anker, die die entsprechende Knoten-Kennung haben.

Als Ergebnis erhalten wir folgenden HTML-Code:

```
<html>
  <title></title>
  <body>
     <h3><a href="#d0e6">Deutschland</a></h3>                          ❶
     <h3><a href="#d0e24">Frankreich</a></h3>                          ❶
     <h3><a href="#d0e42">Spanien</a></h3>                             ❶
     <p><a name="d0e6">Deutschland</a></p>                             ❷
     <p><a name="d0e24">Frankreich</a></p>                             ❷
     <p><a name="d0e42">Spanien</a></p>                                ❷
  </body>
</html>
```

5.2 unparsed-entity-uri()

Die XSLT-Funktion `unparsed-entity-uri()` erlaubt es, auf ungeparste Entitäten einer DTD – meist Binärdaten, wie Bilder – zuzugreifen. Auch hier ein kleines HTML-Beispiel:

Vgl. Abschnitt 10.16

Hierzu muss die Instanz um eine Doctype-Deklaration und die Entity-Deklarationen für die Bilder darin ergänzt werden. Die DTD selbst ist als existierend vorausgesetzt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EUROPA SYSTEM "europa.dtd"[
<!ENTITY Beispiel1 SYSTEM "Bilder\Beispiel1.tif" NDATA TIF>
<!ENTITY Beispiel2 SYSTEM "Bilder\Beispiel2.tif" NDATA TIF>
<!NOTATION TIF SYSTEM "image/tif">
]>
<EUROPA>
   <LAND>
      <NAME BILD="Beispiel1">Deutschland</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">82.4</EINWOHNERZAHL>
      <HAUPTSTADT>Berlin</HAUPTSTADT>
      <KFZ-KENNZEICHEN>D</KFZ-KENNZEICHEN>
      <TEL-VORWAHL>0049</TEL-VORWAHL>
   </LAND>
   <LAND>
      <NAME BILD="Beispiel2">Frankreich</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">58.5</EINWOHNERZAHL>
      <HAUPTSTADT>Paris</HAUPTSTADT>
      <KFZ-KENNZEICHEN>F</KFZ-KENNZEICHEN>
      <TEL-VORWAHL>0033</TEL-VORWAHL>
   </LAND>
   <LAND>
      <NAME>Spanien</NAME>
      <EINWOHNERZAHL EINHEIT="Millionen">39.4</EINWOHNERZAHL>
      <HAUPTSTADT>Madrid</HAUPTSTADT>
      <KFZ-KENNZEICHEN>E</KFZ-KENNZEICHEN>
      <TEL-VORWAHL>0034</TEL-VORWAHL>
   </LAND>
</EUROPA>
```

Das Stylesheet, in dem die Verarbeitung der Bild-Entitäten spezifiziert wird, sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:template match="EUROPA">
      <html>
         <title></title>
         <body>
            <xsl:for-each select="//NAME">
               <img src="{unparsed-entity-uri(@BILD)}"/>          ❶
            </xsl:for-each>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Als Ergebnis erhalten wir folgenden HTML-Code:

```
<html>
   <title></title>
```

```
   <body>
      <img src="Bilder\Beispiel1.tif"/>                                    ❶
      <img src="Bilder\Beispiel2.tif"/>                                    ❶
   </body>
</html>
```

5.3 count()

Die XPath-Funktion `count()` erwartet eine Knotenmenge, zählt die Anzahl der Knoten und gibt sie als Zahl zurück:

```
number count(node-set)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
   <xsl:template match="/">
      <html>
         <title></title>
         <body>
            <h3>
               Wie viele Länder werden hier vorgestellt?<br/>
               Antwort:
               <xsl:value-of select="count(//LAND)"/> <br/>               ❶
               Wie viele Eigenschaften eines Landes werden vorgestellt?
                   <br/>
               Antwort:
               <xsl:value-of select="count(//LAND/*)div(count(//          ❷
                   LAND))"/>
            </h3>
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

❶ Die Funktion `count()` hat hier alle vorkommenden `<LAND>`-Elemente gezählt, als Zahlenwert dem Element `<xsl:value-of>` übergeben und als arabische Zahl dargestellt.

❷ An dieser Stelle wird eine kleine Rechnung durchgeführt und die Gesamtzahl der Kindelemente aller `<LAND>`-Elemente durch die Anzahl der `<LAND>`-Elemente dividiert.

Als Ergebnis erhalten wir folgenden HTML-Code:

```
<html>
   <title></title>
   <body>
      <h3> Wie viele Länder werden hier vorgestellt?<br>
      Antwort: 3                                                           ❶
      <br><h3> Wie viele Eigenschaften eines Landes werden hier
          vorgestellt?<br>
```

```
      Antwort: 5                                                    ❷
   </body>
</html>
```

5.4 position()

Die XPath-Funktion `position()` gibt die Position des Kontextknotens in einer Knotenmenge zurück.

`number position()`

Meist wird dies eingesetzt, um bestimmte Knoten in einer Folge gleicher Knoten (Elemente) unterschiedlich zu behandeln. Sehr oft wird auch die Kurzschreibweise verwendet. So entspricht `Europa[position()=3]` der Abkürzung `Europa[3]`. Auch hier ein kleines HTML-Beispiel für die Verwendung:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
      <title></title>
      <body>
         <xsl:apply-templates select="//LAND"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="LAND">
   <xsl:if test="position()=1">                                     ❶
      <h1>
         <xsl:value-of select="position()"/>. <xsl:value-of        ❷
             select="."/>
      </h1>
   </xsl:if>
   <xsl:if test="position()=2">                                     ❶
      <h2>
         <xsl:value-of select="position()"/>. <xsl:value-of        ❷
             select="."/>
      </h2>
   </xsl:if>
   <xsl:if test="position()=3">                                     ❶
      <h3>
         <xsl:value-of select="position()"/>. <xsl:value-of        ❷
             select="."/>
      </h3>
   </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

❶ An dieser Stelle wird geprüft, in welcher Position sich der Kontextknoten innerhalb aller für das Template zutreffenden <Land>-Elemente befindet.
❷ Trifft der Test zu, wird der Wert der Position als Zahl ausgegeben und der Inhalt des Elements eingefügt.

Dieser Code führt zu folgendem Ergebnis:

```
<html>
   <title></title>
   <body>
      <h1>1. Deutschland 82.4 Berlin D 0049</h1>                    ❶
      <h2>2. Frankreich 58.5 Paris F 0033</h2>                      ❶
      <h3>3. Spanien 39.4 Madrid E 0034</h3>                        ❶
   </body>
</html>
```

Im Browser stellt sich das Ergebnis wie folgt dar:

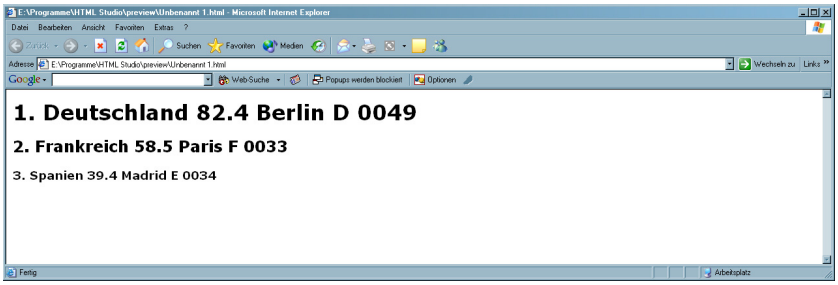


Abb. 5-1
Browser-Ansicht

5.5 last()

Die XPath-Funktion `last()` ermittelt aus einer Knotenmenge den letzten Knoten und gibt dessen Position als Zahl zurück.

`number last()`

Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
      <title></title>
      <body>
         <xsl:apply-templates select="//LAND"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="LAND">
   <xsl:if test="position() &lt; last()">                            ❶
      <h1>
```

```
            <xsl:value-of select="position()"/>. <xsl:value-of select="."/>
         </h1>
      </xsl:if>
      <xsl:if test="position() = last()">                                    ❷
         <h3>
            <xsl:value-of select="position()"/>. <xsl:value-of select="."/>
         </h3>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

❶ Die Abfrage prüft, ob der Kontextknoten eine kleinere Position einnimmt als der letzte Knoten, d. h. ob der Knoten nicht das letzte `<LAND>`-Element ist. Dabei muss das <-Zeichen durch seine Entity-Referenz `&lt;` ersetzt werden, da sonst der XSLT-Prozessor das Zeichen als eine Beginnbegrenzung für einen XML-Starttag halten würde, d. h., dass die Verwendung von <- und >-Zeichen als Vergleichsoperatoren nur mit Hilfe von deren Entity-Referenzen möglich ist.
❷ Hier wird mit Hilfe der `last()`-Funktion das letzte `<LAND>`-Element gesucht.

Dieser Code führt zu folgendem Ergebnis:

```
<html>
   <title></title>
   <body>
      <h1>1. Deutschland 82.4 Berlin D 0049 </h1>                            ❶
      <h1>2. Frankreich 58.5 Paris F 0033 </h1>                              ❶
      <h3>3. Spanien 39.4 Madrid E 0034 </h3>                                ❷
   </body>
</html>
```

5.6 number()

Die XPath-Funktion `number()` wandelt eine Zeichenfolge in eine Zahl um und ermöglicht so das Rechnen mit dieser.

```
number number(string)
```

Bei Zeichenfolgen, die nicht in eine Zahl umgewandelt werden können, wird der Wert `NaN` (Not a Number) zurückgegeben.

5.7 sum()

Die XPath-Funktion `sum()` addiert Zahlen einer Knotenmenge und gibt diese als Zahl zurück.

```
number sum(node-set)
```

Beispiel:

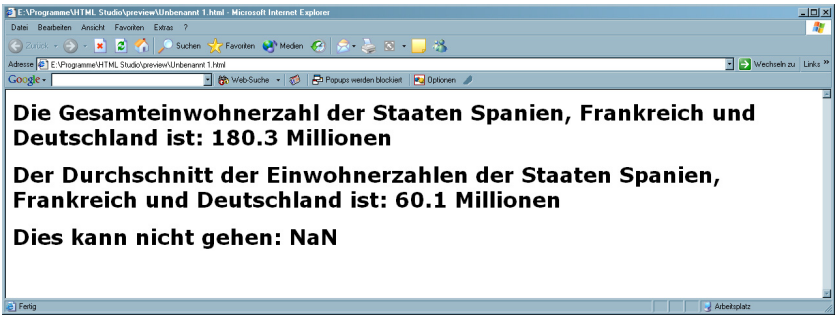
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
      <title></title>
      <body>
         <h1>Die Gesamteinwohnerzahl der Staaten Spanien, Frankreich und
             Deutschland ist:
         <xsl:value-of select="sum(//LAND/EINWOHNERZAHL)"/>                ❶
             Millionen </h1>
         <h1>Der Durchschnitt der Einwohnerzahlen der Staaten Spanien,
             Frankreich und Deutschland ist:
         <xsl:value-of select="sum(//LAND/EINWOHNERZAHL)                   ❶
         div count(//LAND)"/> Millionen </h1>                              ❷
         <h1>Dies kann nicht gehen:
         <xsl:value-of select="sum(//LAND/HAUPTSTADT)"/></h1>              ❸
      </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

❶ Hier wird die Funktion `sum()` dazu genutzt, die Summe aller Inhalte des Elements `<EINWOHNERZAHL>` zu ermitteln.
❷ An dieser Stelle wird eine Weiterverarbeitung vorgenommen und der Durchschnitt aller Einwohnerzahlen ermittelt, wozu die Anzahl aller Einwohner durch die Anzahl der beteiligten Länder dividiert wird.
❸ Hier soll lediglich demonstriert werden, dass bei einer nichtadäquaten Verwendung der Funktion `sum()` das Ergebnis `NaN` erscheint.

Als Ergebnis erhalten wir folgenden Code:

```
<html>
   <title></title>
   <body>
      <h1>Die Gesamteinwohnerzahl der Staaten Spanien, Frankreich       ❶
          und Deutschland ist: 180.3 Millionen </h1>
      <h1>Der Durchschnitt der Einwohnerzahlen der Staaten Spanien,     ❷
          Frankreich und Deutschland ist: 60.1 Millionen </h1>
      <h1>Dies kann nicht gehen: NaN</h1>                               ❸
   </body>
</html>
```

Abb. 5-2
Browser-Ansicht



5.8 not()

Die XPath-Funktion `not()` negiert den in der Klammer enthaltenen Ausdruck.

`boolean not(boolean)`

Folgendes Beispiel demonstriert die Arbeitsweise dieser Funktion:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform">
<xsl:template match="/">
   <html>
      <title></title>
      <body>
         <xsl:apply-templates select="//EINWOHNERZAHL"/>
      </body>
   </html>
</xsl:template>
<xsl:template match="EINWOHNERZAHL">
   <xsl:if test=". &lt; 75">                                              ❶
      Die Einwohnerzahl <xsl:value-of select="."/> Millionen aus
          <xsl:value-of select="preceding-sibling::NAME"/> ist kleiner als
          75 Millionen.<br/>
   </xsl:if>
   <xsl:if test="not(. &lt; 75)">                                         ❷
      Die Einwohnerzahl <xsl:value-of select="."/> Millionen aus
          <xsl:value-of select="preceding-sibling::NAME"/> ist größer als
          75 Millionen.<br/>
   </xsl:if>
   <xsl:if test=". = not(preceding-sibling::NAME='Deutschland')">          ❸
      <xsl:value-of select="preceding-sibling::NAME"/> ist nicht
          Deutschland.<br/>
   </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

❶ Durch das `test`-Attribut wird geprüft, ob der Inhalt des Kontextknotens `<EINWOHNERZAHL>` **kleiner 75** ist.

❷ Der XPath-Ausdruck der vorangehenden <xsl:if>-Anweisung wird durch die Funktion not() negiert, also wird hier nach Werten **größer oder gleich 75** gesucht.
❸ Es wird ein <EINWOHNERZAHL>-Element gesucht, das **kein** vorangegangenes Geschwisterelement <NAME> hat, dessen Wert Deutschland ist.

Als Ergebnis erhält man folgenden Code:

```
<html>
   <title></title>
   <body>
      <h1>
         Die Einwohnerzahl 82.4 Millionen aus Deutschland ist größer    ❷
             als 75 Millionen.<br>
         Die Einwohnerzahl 58.5 Millionen aus Frankreich ist kleiner    ❶
             als 75 Millionen.<br>
         Frankreich ist nicht Deutschland.<br>                          ❸
         Die Einwohnerzahl 39.4 Millionen aus Spanien ist kleiner       ❶
             als 75 Millionen.<br>
         Spanien ist nicht Deutschland.<br>                             ❸
      </h1>
   </body>
</html>
```

6 XSL-FO – Einführung in die Sprache für Seitengestaltung und Umbruch

6.1 Schnellstart

Eine XSL-FO-Datei hat immer das gleiche Grundgerüst. Das Wurzelelement heißt <fo:root>. Kindelemente sind ein <fo:layout-master-set>-, ein optionales <fo:declarations>- und mindestens ein <fo:page-sequence>-Element.

Im <fo:layout-master-set> werden die seitenspezifischen Informationen eingetragen, d. h. die Größe einer Seite, die Ränder, Kopf- und Fußbereich usw.

Die <fo:declarations>-Elemente erlauben, Farbprofile anzulegen, sind aber optional und werden selten eingesetzt, weil von den bisher verfügbaren Formatierern bisher noch nicht unterstützt.

Die <fo:page-sequence>-Elemente legen die Reihenfolge fest, in der die im <fo:layout-master-set> definierten Seiten erscheinen sollen.

Zu Beginn sei das berühmte „Hello World"-Beispiel eingeführt, um den minimalen Seitenaufbau einer XSL-FO-Datei zu demonstrieren.

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">                ❶
   <fo:layout-master-set>                                             ❷
      <fo:simple-page-master master-name="HelloWorld">                ❸
         <fo:region-body/>                                            ❹
      </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-reference="HelloWorld">                   ❺
      <fo:flow flow-name="xsl-region-body">                           ❻
         <fo:block>Hello World!!!</fo:block>                          ❼
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

Autor: Manuel Montero Pineda

Abb. 6-1
Browser-Ansicht

❶ Das Element `<fo:root>` ist immer das Wurzelelement eines FO-Dokuments. Es enthält die Namensraum-Deklarationen (namespace), obligatorisch natürlich die für FO selbst:

```
xmlns:fo="http://www.w3.org/1999/XSL/Format"
```

Sollten Sie mit dem Antenna House Formatter arbeiten, so steht der Namensraum für proprietäre Erweiterungen von Antenna House ebenfalls in diesem Element.

Vgl. Abschnitt 6.2

❷ Im Element `<fo:layout-master-set>` erzeugen Sie die Vorlagen für die Seitengestaltung, die Reihenfolge der Seitenvorlagen, definieren Sie die Ränder und Regionen der Seiten. In diesem Beispiel enthält das Element lediglich einen `<fo:simple-page-master>` und ist damit der denkbar einfachste Fall.

Vgl. Abschnitt 6.2

❸ Der `<fo:simple-page-master>` definiert das Aussehen einer Seite. Hier lassen sich Vorgaben zu Höhe und Breite der Seite machen und verschiedene Ränder und Regionen definieren. In diesem einfachen Beispiel ist keine Spezifikation für die Seitengröße oder die Seitenränder zu entdecken. Diese an und für sich notwendigen Angaben werden durch Vorgabewerte des Formatierers ersetzt. Das Attribut `master-name="HelloWorld"` gibt dieser Seitengestaltung einen Namen, so dass die Vorgabe in der Seitenfolge (`<fo:page-sequence>`) aufgerufen werden kann.

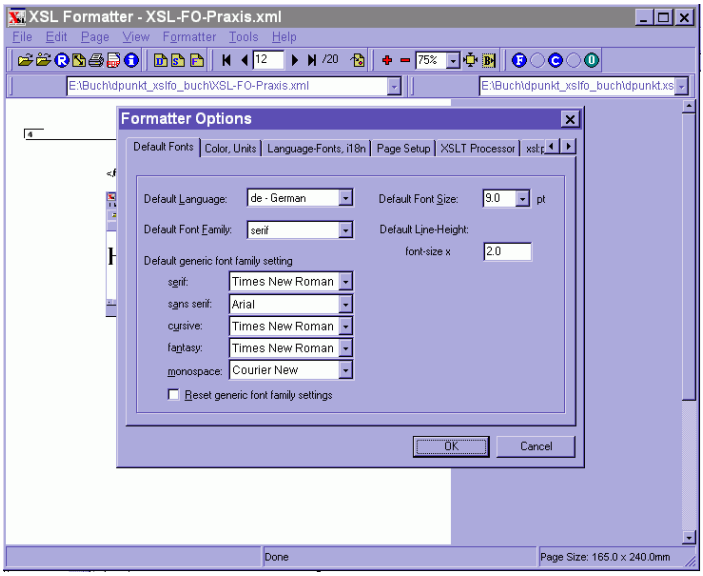


Abb. 6-2
Formatter Options

❹ Das Element `<fo:region-body/>` legt das Aussehen der Region fest, in der sich der laufende Inhalt innerhalb der Seite befindet.

❺ `<fo:page-sequence>` legt die Reihenfolge der oben definierten Seitenvorlagen fest und erlaubt die Einbettung des eigentlichen Inhalts in die Seitenfolge. Um den Bezug auf die Seitenvorlage zu haben, wird dem Attribut `master-reference` der Name dieser Vorlage übergeben, hier also die Seitengestaltung mit dem Namen `HelloWorld`.

❻ Das Element `<fo:flow>` gibt an, dass an dieser Stelle der laufende Inhalt eingefügt wird.

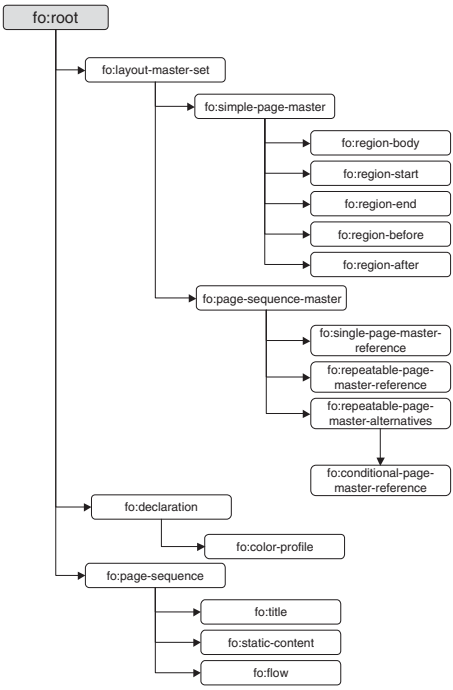
❼ Zur Formatierung dieses Inhalts ist mindestens die Einkleidung in ein Block-Element notwendig (`<fo:block>`). Überschritte dieser Block die Dimensionen der vorgegebenen Body-Region, würde der Formatierer selbstständig den Seitenumbruch durchführen, also den Block zwischen den Seiten teilen. An Stelle eines einfaches Blocks könnten hier auch komplexere Blockstrukturen eingefügt werden, wie Tabellen oder Aufzählungen. In dem einfachen Beispiel enthält dieser Block lediglich den Text `Hello World!!!`. Die an und für sich notwendigen Angaben für Schriftart, Schriftgröße und einiges mehr werden durch die Vorgabewerte des Formatierers ersetzt.

6.2 Das Seiten-Layout

Im vorangegangenen Beispiel wurden schon einige Elemente, die zur Seitengestaltung gebraucht werden, vorgestellt. An dieser Stelle soll das Konzept des Seitenaufbaus und der Seitenreihenfolge eingehend unter die

Lupe genommen werden. Die möglichen Elemente eines Seitenlayouts werden in der folgenden Grafik aufgezeigt und in den folgenden Kapiteln erläutert:

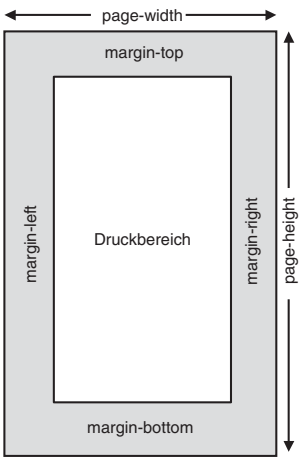
Abb. 6-3
Seiten-Elementstruktur



6.2.1 Der Seitenaufbau

In den folgenden Grafiken ist der Aufbau der Seiten zu sehen:

Abb. 6-4
Seitenaufbau



Für die Definition einer Seite in XSL-FO werden folgende Angaben benötigt:

- Seitenbreite (`page-width`)
- Seitenhöhe (`page-height`)
- Ränder (optional) in vier Bereiche unterteilt:
  - oben (`margin-top`)
  - unten (`margin-bottom`)
  - links (`margin-left`)
  - rechts (`margin-right`)

Alle diese Angaben sind im Element `fo:simple-page-master` mit Hilfe von Attributen zu machen.

Diese Angaben könnten wie folgt lauten:

```
<fo:simple-page-master master-name="HelloWorld"
                       page-height="297mm"
                       page-width="210mm"
                       margin-left="12mm"
                       margin-right="12mm"
                       margin-top="20mm"
                       margin-bottom="20mm">
```

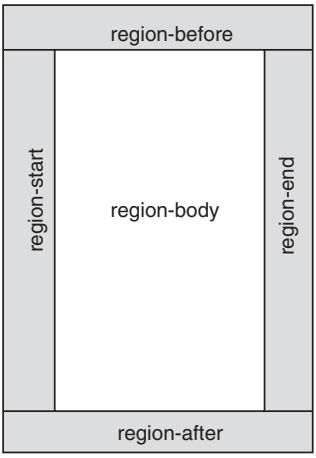
Weitere wichtige Attribute des Elements `<fo:simple-page-master>` zur Ausrichtung des Textes sind:

- `reference-orientation, writing-mode`

6.2.2 Die Druckbereiche

Die Druckbereiche dienen zur weiteren Unterteilung der Seite. Hier werden, wie in der Grafik gezeigt, fünf verschiedene Bereiche unterschieden. Zu beachten ist, dass die Platzierung dieser Bereiche an unsere gewohnte Schreibrichtung gebunden ist (von links nach rechts und von oben nach unten).

Abb. 6-5
Bereiche



- Kopfbereich (`<fo:region-before>`)
- Fußbereich (`<fo:region-after>`)
- linker Bereich (`<fo:region-start>`)
- rechter Bereich (`<fo:region-end>`)
- Hauptbereich (`<fo:region-body>`)

Jeder dieser Bereiche ist ein Kindelement von `<fo:simple-page-master>` und dient dazu, die Eigenschaften dieser Bereiche zu spezifizieren.

Auch hier ein kleines Beispiel:

```
<fo:simple-page-master master-name="HelloWorld">
   <fo:region-before extent="20mm"/>                                    ❶
   <fo:region-after extent="20mm"/>                                     ❶
   <fo:region-body margin-top="10mm" margin-bottom="10mm"/>             ❷
</fo:simple-page-master>
```

❶ Das hier verwendete Attribut `extent` gibt die Ausdehnung dieses Bereiches an, im Falle des Kopf- und Fußbereichs also die Höhe.

❷ Der Hauptbereich bildet insofern eine Ausnahme, weil seine Ausdehnung nicht angegeben wird. Er ist vielmehr derjenige Bereich, der in der Mitte einer Seite übrigbleibt, nachdem alle Ränder und Bereiche definiert wurden. Für die Seitengestaltung im Ganzen und in vielen möglichen Block-Elementen gibt es die Möglichkeit, Rahmen bzw. Randflächen zu reservieren. In diesem Beispiel wird ein 10 mm hoher Rand am unteren und oberen Hauptbereich spezifiziert. Zusammen mit dem Kopf- und Fußbereich bedeutet dies, dass der Inhalt innerhalb des Hauptbereichs jeweils 30 mm vom oberen und unteren Seitenrand entfernt beginnt bzw. endet, aber direkt an die Seitenränder angrenzt, weil die Start- und End-Regionen nicht spezifiziert sind.

Weitere wichtige Attribute der Region-Elemente für ihre Bezeichnung, Rahmen bzw. Ränder, Füllungen und Hintergründe sind:

- region-name
- display-align
- column-count, column-gap
- reference-orientation, writing-mode
- clip
- overflow

6.2.3 Die Schreibrichtung

Die Schreibrichtung für den Text ist – wie bekannt – in vielen Sprachen anders als in den europäischen. XSL-FO unterstützt die für alle Weltsprachen üblichen Schreibrichtungen. Die Schreibrichtung kann für ganze Seiten im Element <fo:simple-page-master> oder in Textblöcken oder sogar im inzeiligen Text mit dem Attribut writing-mode bestimmt werden. So ist es z. B. möglich, ein arabisch-deutsches Wörterbuch zu erstellen, für das die Schreibrichtung innerhalb einer Zeile variiert. Das Attribut writing-mode besitzt drei mögliche Werte: lr-tb (**l**eft**r**ight-**t**op**b**ottom), rl-tb (**r**ight**l**eft-**t**op**b**ottom) und tb-rl (**t**op**b**ottom-**r**ight**l**eft). Die erste Buchstabenkombination gibt hierbei die Schreibrichtung innerhalb von Zeilen vor, die zweite die Richtung der Aneinanderreihung für die Textblöcke.

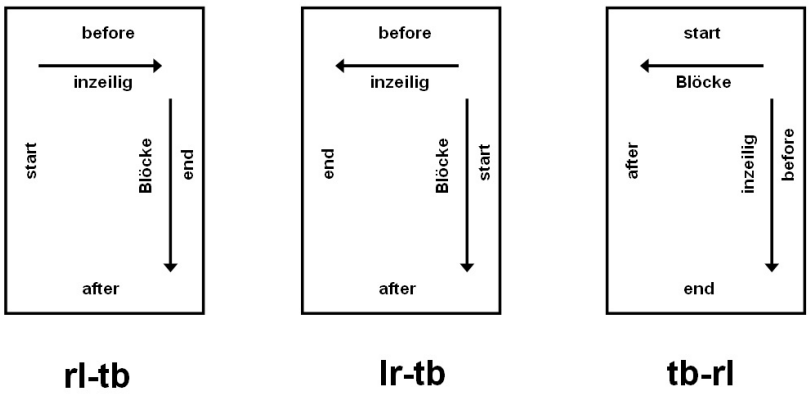


Abb. 6-6
Schreibrichtung

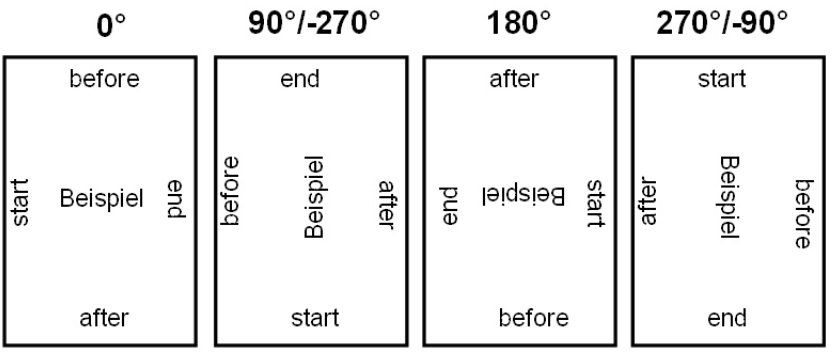
Wie in der Darstellung der Schreibrichtungen zu sehen, bestimmt die Schreibrichtung auch die Positionierung der Seitenbereiche.

Die Schreibrichtung kann zusätzlich durch das Ausrichtungs-Attribut reference-orientation gesteuert werden.

6.2.4 Die Ausrichtung

Mit Hilfe des `reference-orientation`-Attributs lässt sich die Ausrichtung ändern, beispielsweise um Texte zu stürzen. Es kann sieben Werte einnehmen: `0`, `90`, `180`, `270`, `-90`, `-180`, `-270`. Der Wert `0` ist hierbei der Vorgabewert. Bei den Werten handelt es sich um Gradangaben, die das Drehen von Textblöcken oder Zeilen in jeweils 90°-Winkeln erlaubt. Dadurch bedingt, drehen sich ebenso die Bereiche eines Textblocks (z. B. oberer oder linker Rand).

Abb. 6-7
Ausrichtung



Zur Verdeutlichung der Funktionsweise ein kleines Beispiel:

```
...
<fo:block-container reference-orientation="90">                    ❶
   <fo:block font-size="8pt">
      Beispiel für die Textausrichtung.
   </fo:block>
</fo:block-container>
```

❶ Die Änderung der Ausrichtung ist lediglich in dem Container-Element `fo:block-container` zulässig, nicht beispielsweise innerhalb von einfachen Blöcken (`fo:block`). Das Attribut `reference-orientation` sorgt hier für eine 90°-Drehung des Textes.

6.2.5 Maßeinheiten

Die von XSL-FO unterstützten Maßeinheiten sind:

Maßeinheit	Erläuterung
`cm`	Zentimeter
`mm`	Millimeter
`in`	Inch (1 in = 2,54 cm)
`pt`	DTP-Punkte (1 pt = 1/72 in = 0,353 mm); geringfügig abweichend von Didot-Punkten (= 0,376 mm)

Maßeinheit	Erläuterung
pc	Picas (1 pc = 12 pt)
px	Pixel
em	von der Schriftart und -größe abhängige Einheit (Breite eines m); in der Praxis entspricht 1em der Fonthöhe

Der Antenna House XSL-Formatierer unterstützt alle diese Einheiten einschließlich der ersten Nachkommastelle.

6.2.6 Seitenfolgen-Vorlagen definieren

Vgl. Abschnitt 10.24

Das <fo:page-sequence-master>-Element ist neben dem bereits eingeführten <fo:simple-page-master> das zweite Kindelement des Elements <fo:layout-master-set>. Das Element <fo:page-sequence-master> legt fest, in welcher Reihenfolge die Seitengestaltungen (Vorgabeseiten) oder Sequenzen von Seitengestaltungen in das Dokument eingefügt werden. Bei den Seitenfolgen (<fo:page-sequence-master>) handelt es sich letztlich um eine Bündelung von Einzelseiten oder von ganzen Sequenzen. In welcher Reihenfolge diese anschließend im Dokument erscheinen, wird mit dem Element <fo:page-sequence> festgelegt.

In diesem Buch, wie in vielen anderen, müssen beispielsweise linke und rechte Seiten unterschiedlich behandelt, das Vorwort mit einer römischen Seitennummerierung versehen und das Inhaltsverzeichnis in einem spezifischen Layout generiert werden.

Das <fo:page-sequence-master>-Element enthält lediglich ein Attribut, das dessen Referenznamen festlegt (master-name). Die Seitenfolgen selbst werden durch die enthaltenen Elemente <fo:repeatable-page-master-reference>, <fo:repeatable-page-master-alternatives>, <fo:conditional-page-master-reference> und <fo:single-page-master-reference> definiert.

Hierzu als Beispiel die Seitenfolgen für den Hauptteil dieses Buches:

```
<fo:page-sequence-master master-name="Inhalt-Seiten">                    ❶
   <fo:repeatable-page-master-alternatives>                              ❷
      <fo:conditional-page-master-reference master-                      ❸
          reference="PageMaster.Inhalt-rechts-start" page-
          position="first" odd-or-even="odd"/>
      <fo:conditional-page-master-reference master-                      ❹
          reference="PageMaster.Inhalt-links" page-
          position="rest" odd-or-even="even"/>
      <fo:conditional-page-master-reference master-                      ❹
          reference="PageMaster.Inhalt-rechts" page-
          position="rest" odd-or-even="odd"/>
```

```
      <fo:conditional-page-master-reference master-                    ❺
          reference="PageMaster.Teil-links" page-
          position="last" odd-or-even="even" blank-or-not-
          blank="blank"/>
      <fo:conditional-page-master-reference master-                    ❻
          reference="PageMaster.Inhalt-links" page-
          position="last" odd-or-even="even"/>
      <fo:conditional-page-master-reference master-                    ❻
          reference="PageMaster.Inhalt-rechts" page-
          position="last" odd-or-even="odd"/>
   </fo:repeatable-page-master-alternatives>
   <fo:single-page-master-reference master-                            ❼
       reference="PageMaster.Referenz"/>
</fo:page-sequence-master>
```

❶ Das Attribut `master-name` legt hier, wie auch in den `<fo:simple-page-master>`-Elementen den Namen der Seitenfolge fest, die im `<fo:page-sequence-master>`-Element definiert wird.

❷ Das Element `<fo:repeatable-page-master-alternatives>` gibt die Möglichkeit, in `<fo:simple-page-master>`-Elementen definierte Seiten oder in `<fo:page-sequence-master>`-Elementen festgelegte Seitenfolgen zu wiederholen oder unter bestimmten Bedingungen ablaufen zu lassen. Zur Festlegung der Anzahl der Wiederholungen steht das `maximum-repeat`-Attribut bereit.

❸ `<fo:conditional-page-master-reference>` gibt eine Alternative an. Das Attribut `master-reference` referenziert die Seitendefinition bzw. Seitenfolge, die eingesetzt werden soll, wenn es sich um die erste Seite handelt (`page-position="first"`). Außerdem wird festgelegt, dass die erste Seite immer auf einer Seite mit ungerader Seitenzahl stehen soll (`odd-or-even="odd"`). Sollte der Inhalt der vorangegangenen Seiten in einer ungeraden Seite enden, so wird automatisch eine leere Seite eingefügt, damit die neue Seitenfolge mit einer ungeraden Seite beginnen kann. In diesem Buch betrifft dies u. a. alle Kapitel der ersten Ebene. Sie beginnen jeweils auf einer rechten Seite.

❹ An dieser Stelle wird für die übrigen Seiten (`page-position="rest"`) das Aussehen einer geraden bzw. ungeraden Seite festgelegt.

Vgl. Abschnitt 10.25

❺ Hier wird das Aussehen einer geraden (linken) letzten und leeren Seite festgelegt (`page-position="last" odd-or-even="even" blank-or-not-blank="blank"`).

❻ Sollte die letzte Seite nicht leer sein, kommen die hier definierten Seitendefinitionen zum Zug.

❼ Das Element `<fo:single-page-master-reference>` legt eine Seite oder Seitenfolge fest, die in jedem Fall an dieser Stelle (hier nach allen anderen Seitenfolgen) eingesetzt wird.

Die folgende Grafik veranschaulicht den Ablauf der Seiten und Seitenfolgen entsprechend der obigen Definition:

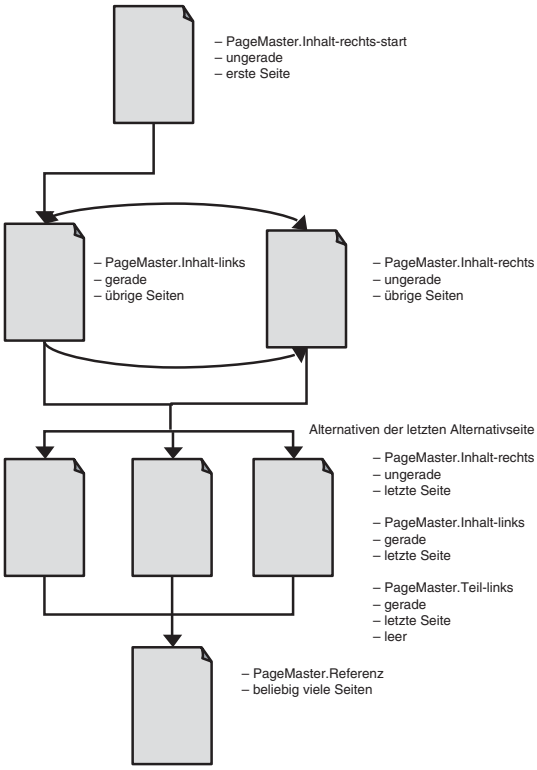


Abb. 6-8
Seitenfolgen

Das Element `<fo:single-page-master-reference>` hat lediglich ein Attribut (`master-reference`).

`<fo:conditional-page-master-reference>` kann folgende Attribute enthalten:

- `master-reference`
- `page-position`
- `odd-or-even`
- `blank-or-not-blank`.

`<fo:repeatable-page-master-alternatives>` kann lediglich mit dem Attribut `maximum-repeat` näher bestimmt werden.

6.2.7 Farbprofile

Das optionale Element `<fo:declarations>` kann ein oder mehrere Kindelemente `<fo:color-profile>` enthalten, mit denen Farbprofile definiert und referenziert werden. Wenn Sie das im Web gebräuchliche Farbprofil RGB (Rot – Grün – Blau) benutzen wollen, brauchen Sie dieses nicht zu

definieren, weil RGB auch ohne jede Deklaration gilt. Im anspruchsvollen Farbdruck kommt man mit dem RGB-Profil jedoch nicht aus. Für die Wahl eines anderen Profils, beispielsweise CMYK, würde die Definition wie folgt aussehen können:

```
<fo:declarations>
   <fo:color-profile src="..." color-profile-name="CMYK"/>
</fo:declarations>
```

Im Attribut `src` ist an Stelle der `...` auf die extern gespeicherten Farbprofildaten zu verweisen.

6.2.8 Die Seitenfolgen festlegen

Vgl. Abschnitt 10.25

Das Element `<fo:page-sequence-master>` ist, wie oben dargestellt, für die Bündelung der Vorgaben für Seiten und Seitenfolgen zuständig. Wie diese Folgen im Satz der XML-Instanz aufgerufen werden, entscheidet das Element `<fo:page-sequence>` bzw. die Reihenfolge mehrerer Seitenfolgen, in der diese verarbeitet werden sollen.

Die weitere Aufgabe des `<fo:page-sequence>`-Elements besteht darin, den im `<fo:simple-page-master>` definierten Bereichen die Inhalte zuzuordnen. Innerhalb des `<fo:page-sequence>`-Elements werden die Regionen, die den eigentlichen Inhaltsbereich umfassen (Kopfbereich, Fußbereich, Startbereich, Endbereich) mit ihren statischen Inhalten gefüllt. Außerdem wird für den Inhaltsbereich spezifiziert, welche Inhalte der XML-Instanz in dieser Sequenz zur Verarbeitung importiert werden sollen.

Im folgenden Beispiel handelt es sich um nicht ausführbare, stark vereinfachte Ausschnitte aus dem Stylesheet, das dem Layout dieses Buches zu Grunde liegt. Hier sollen lediglich die prinzipiellen Zusammenhänge der Elemente `<fo:layout-master-set>`, `<fo:simple-page-master>`, `<fo:page-sequence-master>` und `<fo:page-sequence>` aufgezeigt werden.

```
<fo:root>
   <fo:layout-master-set>                                              ❶
      <fo:simple-page-master margin="{$Seitenraender-Inhalt-           ❷
          rechts}" master-name="PageMaster.Inhalt-rechts">
         <fo:region-body margin="{$Body-Raender-Inhalt-                ❸
             rechts}"/>
         <fo:region-before region-name="Inhalt-rechts-davor"           ❹
             extent="15mm"/>
         <fo:region-after region-name="Inhalt-rechts-danach"/>         ❺
      </fo:simple-page-master>
      ...
      <fo:page-sequence-master master-name="Inhalt-Seiten">            ❻
         <fo:repeatable-page-master-alternatives>                      ❻
```

```
            <fo:conditional-page-master-reference master-                ❻
                reference="PageMaster.Inhalt-rechts-start"
                page-position="first" odd-or-even="odd"/>
            <fo:conditional-page-master-reference master-
                reference="PageMaster.Inhalt-links" page-position="rest"
                odd-or-even="even"/>
            <fo:conditional-page-master-reference master-                ❻
                reference="PageMaster.Inhalt-rechts" page-
                position="rest" odd-or-even="odd"/>
            <fo:conditional-page-master-reference master-                ❻
                reference="PageMaster.Inhalt-links" page-
                position="last" odd-or-even="even"/>
            <fo:conditional-page-master-reference master-                ❻
                reference="PageMaster.Inhalt-rechts" page-
                position="last" odd-or-even="odd"/>
         </fo:repeatable-page-master-alternatives>
      </fo:page-sequence-master>
   </fo:layout-master-set>
   ...
   <fo:page-sequence master-reference="Inhaltsverzeichnis" for-              ❼
       ce-page-count="even" >
   ...
   </fo:page-sequence>
   <fo:page-sequence master-reference="Inhalt-Seiten" force-                 ❽
       page-count="even" >
      <fo:static-content flow-name="Inhalt-rechts-davor-start">
         <fo:block> Erste Seite Kopfbereich </fo:block>
      </fo:static-content>
      <fo:static-content flow-name="Inhalt-rechts-davor">                   ❹
         <fo:block > Rechte Seite Kopfbereich</fo:block>
      </fo:static-content>
      <fo:static-content flow-name="Inhalt-links-davor">
         <fo:block > Linke Seite Kopfbereich </fo:block>
      </fo:static-content>
      <fo:static-content flow-name="Inhalt-rechts-danach">                  ❺
         <fo:block> Rechte Seite Fußbereich </fo:block>
      </fo:static-content>
      <fo:static-content flow-name="Inhalt-links-danach">
         <fo:block>Linke Seite Fußbereich </fo:block>
      </fo:static-content>
      <fo:flow flow-name="xsl-region-body">                                 ❸
         <fo:block>
            <xsl:apply-templates/>                                          ❾
         </fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

❶ Im `<fo:layout-master-set>` werden alle Vorgaben zu Einzelseiten und Seitenfolgen definiert.

❷ Das Element <fo:simple-page-master> definiert die Größe der Seite, Rahmen und Bereiche. In diesem Fall hat die Vorgabeseite den Namen PageMaster.Inhalt-rechts. Dieser Name wird in dem Beispiel im Element <fo:page-sequence-master> wieder aufgegriffen, um diese Seite in eine Seitenfolge zu integrieren. Tatsächlich fehlen in diesem Beispiel etliche weitere <fo:simple-page-master>, die weggelassen wurden, um die Übersicht zu wahren.

❸ Der Bereich <fo:region-body> hat keinen region-name, weil er den festen, vorgegebenen Namen xsl-region-body hat.

❹ Der Bereich <fo:region-before>, also der Kopfbereich der Seite, hat hier den Namen Inhalt-rechts-davor. Dieser muss später in Element <fo:page-sequence> aufgerufen werden, um diesem Bereich einen Inhalt zuzuweisen. Die Zuordnung von Inhalt erfolgt im Kopfbereich mit dem Element <fo:static-content>, das feste Inhalte in diesen Bereich der Seite einfügt. Der feste Inhalt kann einmal darin bestehen, dass er vollkommen statisch ist (beispielsweise bei einem unveränderbaren Titel in allen Seiten der gegebenen Seitenfolge) oder dass der Inhalt generiert wird, wie im Falle einer fortlaufenden Seitennummerierung oder auch eines lebenden Kolumnentitels (beispielsweise die Wiederholung des aktuellen Abschnittstitels im Kopfbereich).

❺ Für den Fußbereich gilt Gleiches wie für den Kopfbereich der Seite.

❻ Im Element <fo:page-sequence-master> werden nun Seitendefinitionen zu Seitenfolgen unter dem Namen Inhalt-Seiten zusammengefasst. Dieser Name wird im Element <fo:page-sequence> aufgerufen, so dass der Formatierer die im Element <fo:page-sequence-master> definierten Seitenfolgen ausführen und mit Inhalt füllen kann.

❼ Mit dem Element <fo:page-sequence> wird nun mindestens eine Seitenfolge, bei Aneinanderreihung mehrerer dieser Elemente werden entsprechend viele Seitenfolgen genau entsprechend ihrer sequenziellen Anordnung im Stylesheet erzeugt. Der darin bezeichnete Inhalt (ist in diesem Beispiel durch ... ersetzt) fließt in die Folge der Seiten ein. Dieses Beispiel hat zwei <fo:page-sequence>-Elemente. Das erste bezieht sich auf einen <fo:page-sequence-master> oder <fo:simple-page-master> mit dem Namen Inhaltsverzeichnis, das zweite auf den <fo:page-sequence-master> mit dem Namen Inhalt-Seiten.

❽ Das Element <fo:page-sequence> enthält hier verschiedene <fo:static-content> Elemente, die die Kopf- und Fußbereiche der verschiedenen linken und rechten Seiten mit einem kurzen Text beschreiben. Dieser Text würde in den jeweiligen Bereichen der Seiten erscheinen. Im Falle des <fo:flow> Elementes handelt es sich um das Element, welches festlegt, an welcher Stelle der fließende Text eingefügt werden soll. Jedes

<fo:page-sequence> Element darf lediglich ein <fo:flow> enthalten. Dieses muss nach allen <fo:static-content>-Elementen innerhalb von <fo:page-sequence> gesetzt werden.

❾ Mit dem <xsl:apply-templates/>-Element wird nun der gesamte Inhalt der Instanz an dieser Stelle eingefügt, dies deshalb, weil der gegebene Stylesheet-Teil innerhalb des Templates für das oberste Element (das Root-Element) der zu setzenden XML-Instanz platziert ist.

<fo:page-sequence> kann folgende Attribute enthalten:

- master-name bezeichnet den Namen der Seitenfolge
- id definiert eine ID für die Seitenfolge
- language gibt die Sprache des Inhalts der Seitenfolge an
- country spezifiziert das Land, in dessen Sprache der Inhalt erfasst ist
- initial-page-number gibt den Startwert für die Seitennummerierung an
- force-page-count legt fest, ob die Seitenfolge mit einer geraden oder ungeraden Seite enden oder beginnen soll
- format legt das Seitennummerierungsformat fest
- grouping-separator, grouping-size erlauben weitere Festlegungen über die Art und das Format der Nummerierung
- letter-value dient zur Formatierung der Seitennummern bei alphanumerischer Zählung.

6.2.9 Textflüsse und statische Inhalte

Innerhalb des Elements <fo:flow> fließen die Inhalte und werden ggf. zu Seiten umbrochen, wenn der Umfang das Aufnahmevermögen einer Seite übersteigt. Der Inhalt dieses Elements bestimmt den Zeilen- und Seitenumbruch und damit die Anzahl der generierten Seiten.

Das Element <fo:static-content> bestimmt die statischen Inhalte von Seitenbereichen außerhalb des Bereichs, der den fließenden Inhalt aufnimmt. Der Inhalt dieser Bereiche ist nicht an Seiten- und Zeilenumbrüchen beteiligt. Ist beispielsweise der Inhalt eines <fo:static-content> im Bereich einer Seite zu groß, so wird der Formatierer keinen Seitenumbruch erzeugen, sondern entweder die Formatierung abbrechen oder eine Warnung ausgeben, die den Überfluss (overflow) in dem gegebenen Bereich anzeigt.

Vgl. Abschnitt 10.10

Beide Elemente <fo:flow> und <fo:static-content> haben nur ein Attribut, flow-name. Der Attributwert verweist auf das Attribut region-name mit dem gleichen Attributwert in einem <fo:simple-page-master>-Element.

Innerhalb einer <fo:page-sequence> darf es lediglich ein <fo:flow> geben, dies muss nach allen <fo:static-content>-Elementen platziert sein.

6.3 Blöcke

Der Block ist eines der zentralen Konzepte von XSL-FO. Er wird benutzt, um alle möglichen Elemente, wie Tabellen, Bilder, Listen, Absätze usw. in rechteckige Anzeigebereiche einzusetzen. Diese Blöcke können beliebig ineinander verschachtelt werden, um mehrere Blöcke einer Ebene zu bündeln.

Mit der Aneinanderreihung der Blöcke füllt sich die Seite oder Spalte. Blöcke, die als Ganzes nicht mehr in den Seiten- oder Spaltenraum passen, werden nach den gegebenen Umbruchregeln zwischen den Seiten geteilt.

An das Element <fo:block> lassen sich Gestaltungsinformationen aller Art durch entsprechende Attribute binden. Zu ihnen gehören Informationen über die zu verwendenden Fonts, Schriftgröße, Rahmen, Einzüge, Farben, Hintergrundfarben und -bilder, Silbentrennung usw. Zu beachten ist, dass die meisten dieser Attribute vererbbar sind. Das heißt, dass sie nicht notwendig dem Block beigegeben sein müssen, der den zu setzenden Inhalt direkt enthält, sondern bereits für Elemente spezifiziert sind, die im Stylesheet dem gegebenen Block übergeordnet sind.

Im folgenden Beispiel werden verschiedene Gestaltungen von <fo:block> gezeigt, die aber die Vielzahl an möglichen Gestaltungen nur sehr eingeschränkt zeigen.

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" >
   <fo:layout-master-set>
      <fo:simple-page-master master-name="HalloWelt">
         <fo:region-body/>
      </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-reference="HalloWelt">
      <fo:flow flow-name="xsl-region-body">
         <fo:block font-family="Arial" font-size="18pt">                ❶
            1. Hallo Welt!!!
         </fo:block>
         <fo:block margin-left="10mm" font-family="Arial"               ❷
         font-size="18pt">
            2. Hallo Welt!!!
         </fo:block>
         <fo:block background-color="blue" font-family="Arial"          ❸
         font-size="18pt">
            3. Hallo Welt!!!
```

```
        </fo:block>
        <fo:block margin-left="10mm" margin-top="20mm" font-          ❹
            family="Times New Roman" font-size="24pt">
           4. Hallo Welt!!!
        </fo:block>
        <fo:block font-weight="bold" font-style="italic" font-       ❺
            family="Arial"
        font-size="18pt">
           5. Hallo Welt!!!
        </fo:block>
        <fo:block border-style="solid" border-width="2pt" margin-    ❻
            left="10mm" margin-right="10mm" font-family="Arial"
        font-size="18pt">
           6. Hallo Welt!!!
        </fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

❶ Die beiden Attribute `font-family` und `font-size` geben die Schriftart und Schriftgröße an.

❷ Hier wird vom Blockrand ein linker Einzug gesetzt: `margin-left`. Solche Einzüge oder Abstände könnten auch oben, unten und rechts gesetzt werden: `margin-top`, `margin-bottom` und `margin-right`.

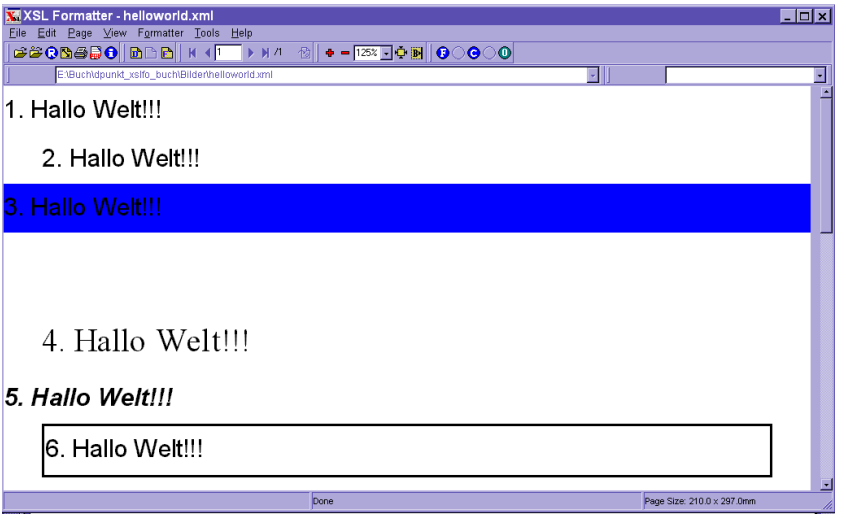
❸ Es stehen eine Vielzahl von Attributen für die Definition von Farben bereit, um wie hier den Hintergrund festzulegen, aber auch um farbige Rahmen zu erzeugen oder farbige Schrift. Hintergrundfarben füllen immer den gesamten Block aus, dessen Dimensionen so sichtbar gemacht werden können.

❹ Hier wird die Verwendung von Einzügen (horizontal) und Vorschüben (vertikal) und einem anderen Font demonstriert.

❺ Die Attribute `font-weight` und `font-style` bestimmen die Fontausprägungen (fett, kursiv, in Kombination fett-kursiv).

❻ Blöcke können mit einem Rahmen versehen werden. Die vier Rahmenteile sind dabei auch einzeln, sowohl in ihrer Gestaltung als auch in der Farbe und Stärke bestimmbar. Das Attribut `border-style` legt hier einen Rahmen in Form einer durchgehenden Linie und `border-width` legt ihre Breite fest. Zu beachten ist, dass der Rahmen den gegebenen Block umfasst, also außerhalb der Blockgrenzen liegt.

Abb. 6-9
Formatierer-Ansicht



Die zulässigen Attribute für das `<fo:block>`-Element sind thematisch geordnet die Folgenden. Die deutschsprachigen Erläuterungen zur Funktionalität, ihrer Attributwerte und ihrer Vererbungseigenschaften sind dem Referenzteil zu entnehmen. Die in der Praxis wichtigen Attribute (Eigenschaften) und ihre funktionalen Eigenheiten werden weiter unten behandelt.

Für Hintergründe:

- `background-attachment`, `background-color`, `background-image`, `background-repeat`, `background-position-horizontal`, `background-position-vertical`

Für Rahmen (außerhalb der Blockgrenzen):

- `border-before-color`, `border-before-style`, `border-before-width`, `border-after-color`, `border-after-style`, `border-after-width`, `border-start-color`, `border-start-style`, `border-start-width`, `border-end-color`, `border-end-style`, `border-end-width`, `border-top-color`, `border-top-style`, `border-top-width`, `border-bottom-color`, `border-bottom-style`, `border-bottom-width`, `border-left-color`, `border-left-style`, `border-left-width`, `border-right-color`, `border-right-style`, `border-right-width`

Für Schriftarten (Fonts) und Schriftausprägungen:

- `font-model`, `font-family`, `font-selection-strategy`, `font-size`, `font-stretch`, `font-size-adjust`, `font-style`, `font-variant`, `font-weight`

Für Silbentrennung:

- `country`, `hyphenate`, `hyphenation-character`, `hyphenation-push-character-count`, `hyphenation-remain-character-count`, `hyphenation-keep`, `hyphenation-ladder-count`, `language`

Zu Zeilen- bzw. Seitenumbruch:

- `break-after`, `break-before`, `keep-together`, `keep-with-next`, `keep-with-previous`

Für Füllungen und Ränder (innerhalb der Blockgrenzen):

- `margin-top`, `margin-bottom`, `margin-left`, `margin-right`, `padding-before`, `padding-after`, `padding-start`, `padding-end`, `padding-top`, `padding-bottom`, `padding-left`, `padding-rightpause-after`, `space-before`, `space-after`

Verschiedenes:

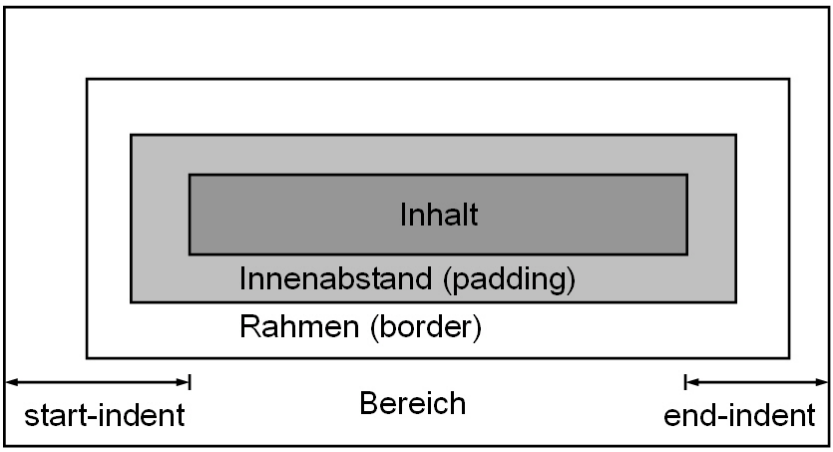
- `color`, `end-indent`, `id`, `intrusion-displace`, `last-line-end-indent`, `linefeed-treatment`, `line-height`, `line-height-shift-adjustment`, `line-stacking-strategy`, `orphans`, `relative-position`, `richness`, `role`, `script`, `source-document`, `span`, `start-indent`, `text-align`, `text-align-last`, `text-altitude`, `text-depth`, `text-indent`, `visibility`, `white-space-collapse`, `white-space-treatment`, `widows`, `wrap-option`

6.4 Rahmen, Ränder, Abstände und Co.

In XSL-FO gibt es mehrere Konzepte zur Positionierung von Inhalten in Blöcken und zur Erzeugung von Abständen, Rahmen, Einrückungen und Rändern. Diese Konzepte können oft alternativ eingesetzt werden. In diesem Kapitel werden die unterschiedlichen Möglichkeiten und ihre Zusammenhänge vorgestellt und auch einige Empfehlungen für die Praxis gegeben, da die unüberlegte Anwendung der Konzepte eine Fehlerquelle ersten Ranges sein kann.

In der folgenden Grafik sind einige der zur Inhaltspositionierung innerhalb eines Blocks geeigneten Konzepte illustriert:

Abb. 6-10
Rahmen und Abstände



6.4.1 Ränder

Ränder, die mit den `margin`-Attributen erzeugt werden, stellen die gebräuchlichste Form zur Erzeugung von Abständen dar. Die Abstände beziehen sich auf die Flächen zwischen den Bereichsgrenzen (Satzspiegelgrenze) und darin liegenden Blöcken sowie Flächen zwischen Blöcken.

Die fünf möglichen Attribute für Ränder sind:

- `margin-top` (oben)
- `margin-bottom` (unten)
- `margin-left` (links)
- `margin-right` (rechts)
- `margin` (oben, unten, links und rechts)

Der Formatierer setzt die Ränder für Blöcke nach folgenden Regeln:

- Werden keine Ränder spezifiziert, erstreckt sich der Block auf Satzspiegelbreite. Aufeinander folgende Blöcke werden ohne Abstand untereinander geschichtet. Diese Regel gilt, weil der in XSL-FO vorgegebene Wert für alle Ränder auf `0pt` gesetzt ist.
- Wird für Links bzw. Rechts ein Rand mit einem positiven Wert spezifiziert, wird der Block links bzw. rechts entsprechend eingezogen.
- Wird für Oben bzw. Unten ein Rand mit einem positiven Wert spezifiziert, entsteht eine entsprechende Fläche über bzw. unter dem Block. Ein ggf. nachfolgender Block wird entsprechend weiter nach unten verschoben.
- Die Werte für Ränder werden nicht vererbt, d. h. untergeordnete Blöcke übernehmen die Werte übergeordneter Blöcke nicht.
- Untergeordnete Blöcke gehen immer von den Rändern des übergeordneten Blocks aus.

- Negative Werte für Ränder links bzw. rechts führen ggf. zur entsprechenden Überschreitung der Satzspiegelgrenzen. Negative Werte für oben bzw. unten werden ignoriert und sollten deshalb nicht gesetzt werden.

Beispiel:

```
<fo:block margin-left="20mm">                                          ❶
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
   <fo:block margin-left="20mm">                                       ❷
      INHALT INHALT INHALT INHALT INHALT INHALT INHALT
   </fo:block>
</fo:block>
<fo:block margin-top="20mm" margin-left="-10mm">                       ❸
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>
```

❶ Das Attribut `margin-left` erzeugt einen Rand zur linken Satzspiegelgrenze. Der Inhalt wird somit um `20mm` eingezogen.
❷ Da dieser Block ein Kindelement des ersten ist, ist der Ausgangspunkt der um `20mm` nach rechts verschobene linke Rand des Elternblocks. Hinzu kommen wiederum `20mm` Rand, so dass der Block insgesamt um `40mm` vom Satzspiegelrand eingezogen ist.
❸ Dieser Block hat einen Rand nach oben hin. Es wird so ein Abstand zum oberen Block erzeugt. Der negative Wert von `margin-left` erweitert den Block um `10mm` nach links und lässt den Block außerhalb der linken Satzspiegelgrenze beginnen.

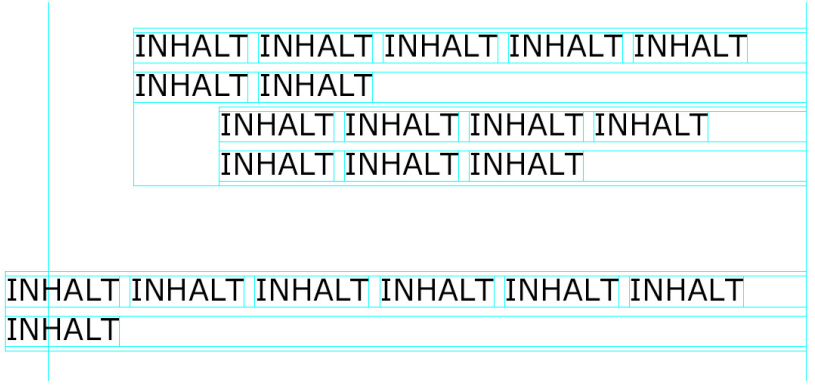


Abb. 6-11 Formatierer-Ansicht

6.4.2 Innenabstände

Mit Hilfe der `padding`-Attribute lassen sich Abstände von den Blockgrenzen nach innen festlegen. Der Block wird durch die `padding`-Attributwerte entsprechend vergrößert.

Die padding-Attribute sind:

- padding-start (bei Schreibrichtung lr-tb – die für europäische Sprachen übliche: links)
- padding-end (bei Schreibrichtung lr-tb: rechts)
- padding-left (links)
- padding-right (rechts)
- padding-before (bei Schreibrichtung lr-tb: oben)
- padding-after (bei Schreibrichtung lr-tb: unten)
- padding-top (oben)
- padding-bottom (unten)
- padding (Kurzform für oben, unten, links und rechts)

Der Formatierer setzt die padding-Attributwerte für Blöcke nach folgenden Regeln um:

- Sind links bzw. rechts Ränder (margin) gesetzt, geht der Innenabstand von den Rändern aus.
- Sind links bzw. rechts keine Ränder gesetzt, wird der Block entsprechend auch über die Satzspiegelgrenzen links bzw. rechts vergrößert.
- padding oben bzw. unten führt zu einem entsprechenden Abstand vom vorausgehenden bzw. nachfolgenden Block. Dafür sollte padding nicht verwendet werden, an dieser Stelle sind margin oder space zu bevorzugen.
- padding zur Satzspiegelgrenze führen nicht zu einer Einrückung des Inhalts, aber zu einer Vergrößerung des Blocks über den Satzspiegel hinaus.
- Die Werte für padding werden nicht vererbt, d. h. untergeordnete Blöcke übernehmen die Werte übergeordneter Blöcke nicht.
- Negative Werte für padding werden ignoriert.

Beispiel:

```
<fo:block padding="10mm 10mm 10mm 10mm">                              ❶
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
   <fo:block padding="10mm 10mm 10mm 10mm">                           ❷
INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>                                                           ❷
</fo:block>
<fo:block padding="10mm 10mm 10mm 10mm">                              ❸
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>
```

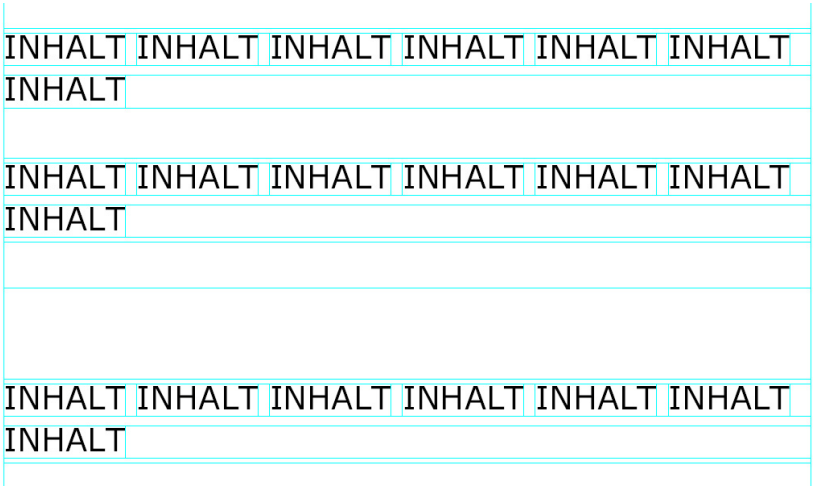


Abb. 6-12
Formatierer-Ansicht

❶ Das `padding`-Attribut erzeugt einen Abstand nach oben und unten zum nächsten Block. In diesem Fall zum dritten Block, da der zweite sich innerhalb der Blockgrenzen des ersten befindet. Die `padding`-Abstände nach links und rechts führen zu keiner Einrückung des Inhalts, da sie auf die Satzspiegelgrenzen treffen.

❷ Das `padding`-Attribut erzeugt einen Abstand zum oberen und unteren Block. Die padding-Abstände nach links und rechts führen zu keiner Einrückung des Inhalts, da sie auch hier auf die Satzspiegelgrenzen treffen.

❸ Das `padding`-Attribut erzeugt einen Abstand zum oberen Block. Der Abstand beträgt `30mm`, da hier die Abstände der beiden oberen Blöcke auf den des unteren treffen und aufaddiert werden.

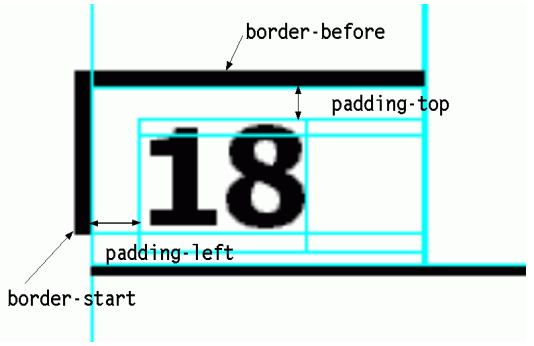
In der Praxis wird das `padding` häufig für mikrotypografische Feinheiten eingesetzt. Hierfür ein Beispiel (die Seitenzahlen in diesem Buch):

```
...
<fo:block line-height="6pt" text-align="left" border-start-                ❶
    style="solid" border-before-style="solid" border-before-width="1pt"
    border-start-width="1pt" margin-top="10.7mm" margin-bottom="0.7mm">
   <fo:block padding-top="0.7mm" padding-left="1mm">                       ❷
      <fo:page-number/>
   </fo:block>
</fo:block>
...
```

❶ Dieser äußere Block dient zur Positionierung des Inhaltsblocks, der einen Block mit den Seitenzahlen enthält. Ebenfalls wird der sichtbare Rahmen (`border-start-style="solid" border-before-style="solid"`) an dieser Stelle festgelegt und dessen Linienstärke (`border-before-width="1pt" border-start-width="1pt"`) bestimmt.

❷ Der innere Block enthält die automatisch generierten Seitenzahlen und die Angaben zum Innenabstand zu dem Rahmen. Es wäre ebenso möglich, diesen inneren Block wegzulassen und auch die Innenabstände im äußeren Block zu spezifizieren. Die Spezifikation des inneren Blocks mit seinen `padding`-Attributen erleichtert die Übersicht.

Abb. 6-13
Seitenzahl Kopfbereich



6.4.3 Rahmen

Rahmen umfassen Blöcke. Entsprechend ihrer Breite fügen sie dem Block, den sie umfassen, außerhalb eine Fläche hinzu. Rahmen werden mit den `border`-Attributen spezifiziert.

Zur Festlegung der Rahmen in den vier Richtungen stehen die folgenden vier Attribute zur Verfügung. Die deutschsprachigen Bezeichnungen gehen von der Schreibrichtung von links nach rechts und von oben nach unten aus.

- `border-before` (oben)
- `border-after` (unten)
- `border-start` (links)
- `border-end` (rechts)

Diese Rahmen in ihrer Grundform können durch weitere Spezifikationen einzeln in ihrer Darstellung bestimmt werden. Hierzu stehen zur Verfügung:

- `color` (Farbe des Rahmens)
- `style` (Erscheinungsform des Rahmens; zulässige Werte sind beispielsweise `solid` (durchgehende Linie oder Fläche), `double` (doppelte Linie) oder `dotted` (punktierte Linie)
- `width` (Linienstärke bzw. Breite des Rahmens)

Beispiel:

```
<fo:block border-style="solid" border-width="1mm" margin-                    ❶
    bottom="10mm" padding="12mm 12mm 12mm 12mm">
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
   <fo:block border-style="solid" border-width="1mm" padding="10mm           ❷
       10mm 10mm 10mm">
      INHALT INHALT INHALT INHALT INHALT INHALT INHALT
   </fo:block>
</fo:block>
<fo:block border-style="solid" border-width="1mm">                           ❸
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>
```

❶ Das Attribut `border-style` mit dem Wert `solid` erzeugt einen Rahmen mit einer durchgezogenen Linie um den Block herum. Die Liniendicke wird mit dem Attribut `border-width` festgelegt. Da hier ein padding-Abstand definiert wurde und der Block einen weiteren enthält, umgibt der Rahmen den Inhalt der beiden Blöcke im Abstand von `12mm` gemessen vom inneren Rand des Rahmens. Das `margin-bottom` erzeugt einen Abstand von `10mm` zum folgenden dritten Block.

❷ Dieser Block ist in den ersteren verschachtelt und somit auch von diesem umgeben.

❸ Da dieser Block keinen `padding`-Abstand hat, liegt der erzeugte Rahmen direkt an der Grenze zum Inhalt auf.

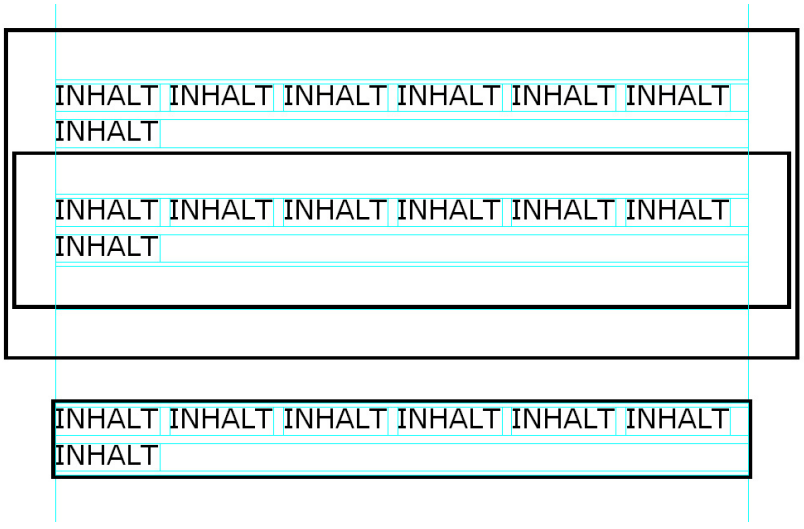
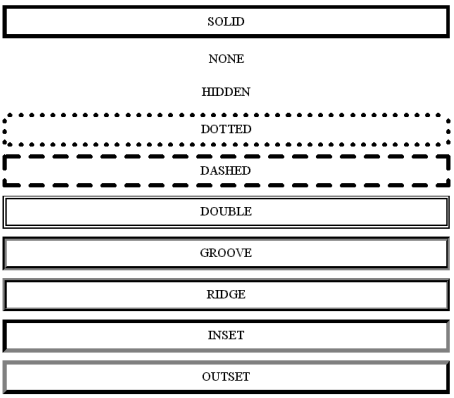


Abb. 6-14
Formatierer-Ansicht

Die folgende Abbildung zeigt die Linienausprägungen, die das Attribut `style` ermöglicht.

Abb. 6-15
Rahmenlinien



Zur Verdeutlichung einige Stylesheet-Beispiele für Rahmen:

```
...
<fo:block text-align="center"
   border-before-style="solid" border-before-width="5mm"                ❶
   border-after-style="double" border-after-width="3mm"                 ❷
   border-start-style="dotted" border-start-width="1mm"                 ❸
   border-end-style="dashed" border-end-width="1mm" >                   ❹
      Einheitliche Farbe, verschiedene Linienbreiten und verschiedene
          Linienarten
</fo:block>
<fo:block text-align="center"
   border-width="2mm" border-style="solid"                              ❺
   border-before-color="grey" border-after-color="red" border-          ❻
       start-color="blue" border-end-color="green">
      Verschiedene Farben, einheitliche Linienbreite und einheitliche
          Linienart
</fo:block>
...
```

❶ Der Attributwert `solid` bewirkt das Aussehen einer durchgehenden Linie und ist in diesem Beispiel am oberen Rand (`before`) des Blockes angesiedelt.
❷ Der Attributwert `double` erzeugt eine doppelte Linie am unteren Rand.
❸ Der Wert `dotted` erzeugt eine gepunktete Linie am linken Rand.
❹ Der Wert `dashed` erzeugt eine gestrichelte Linie am rechten Rand.
❺ Die Attribute `border-width`, `border-style` und `border-color` können eingesetzt werden, wenn alle Rahmenteile des Blockes gleich gestaltet sein sollen.
❻ Mit dem Anhang `color` wird eine von der Grundfarbe abweichende Farbe spezifiziert.

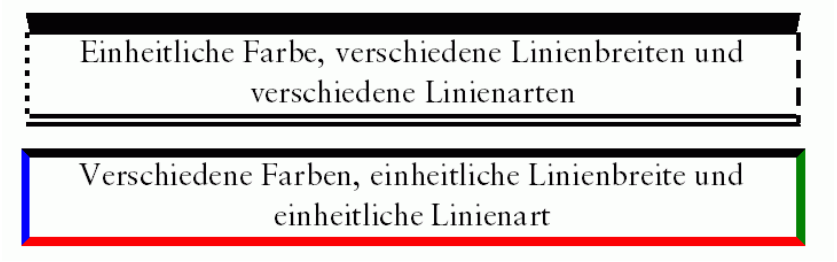


Abb. 6-16
Formatierer-Ansicht

Der Formatierer setzt die `border`-Attributwerte für Blöcke nach folgenden Regeln um:

- Sind links bzw. rechts Ränder (`margin`) gesetzt, geht die äußere Rahmenbegrenzung von den Rändern aus.
- Sind links bzw. rechts keine Ränder gesetzt, ragt der Rahmen entsprechend auch über die Satzspiegelgrenzen links bzw. rechts heraus.
- `border` oben bzw. unten führt zu einem entsprechenden Abstand vom vorausgehenden bzw. nachfolgenden Block.
- Die Werte für `border` werden nicht vererbt, d. h. untergeordnete Blöcke übernehmen die Werte übergeordneter Blöcke nicht.
- Negative Werte für `border` werden ignoriert.

6.4.4 Abstände (vertikale Vorschübe)

Die `space`-Attribute geben eine Reihe von Möglichkeiten zur genaueren Spezifikation von vertikalen Abständen (Vorschüben) zwischen Blöcken zur Hand. Mit Hilfe des `margin`-Attributs lässt sich zwar ebenfalls ein Abstand zu einem folgenden Block erzeugen, jedoch werden die Ränder lediglich aufaddiert, wenn für einen vorausgehenden Block ein Wert `margin-bottom` und für den nachfolgenden Block ein Wert `margin-top` spezifiziert ist. Mit den `space`-Attributen hingegen lassen sich die Abstände aufeinander folgender Blöcke vielfältiger und genauer kontrollieren.

Vgl. Abschnitt 10.5

Für die Attribute `space-after` und `space-before` stehen hierzu folgende Anhänge zur Verfügung:

- `space-after.precedence` bzw. `space-before.precedence` (Vorschub-danach, Vorschub-davor – präferierter Wert).
- `space-after.conditionality` bzw. `space-before.conditionality` (bedingter Vorschub bei Seitenumbrüchen).
- `space-after.optimum` bzw. `space-before.optimum` (optimaler Vorschub – bei Spezifikation von Vorschub-Toleranzen für einen automatisierten vertikalen Keil).

- space-after.minimum bzw. space-before.minimum (minimaler Vorschub).
- space-after.maximum bzw. space-before.maximum (maximaler Vorschub).

Der Anhang .precedence bestimmt beim Aufeinandertreffen von after- und before-Werten, welche Spezifikation bevorzugt (mit Priorität) für den Vorschub gewählt wird. Die Größe des Vorschubs wird durch die Attributwerte in space-after bzw. space-before oder in .optimum, .minimum, .maximum vorgegeben.

Mit dem Anhang .conditionality lässt sich das Verhalten von zwei aufeinander bezogenen Vorschub-Spezifikationen bei Seitenumbrüchen (space-after im vorausgehenden Block, space-before im folgenden Block) durch die zwei möglichen Attributwerte discard und retain bestimmen. Wird der Anhang .conditionality nicht verwendet, so gilt der Vorgabewert discard. Dieser bewirkt, dass bei Seitenumbrüchen der Block auf der neuen Seite immer am oberen Satzspiegelrand beginnt. Im Falle von retain hingegen wird der Abstand zum oberen Satzspiegelrand auf der neuen Seite beibehalten.

Einfaches Beispiel:

```
<fo:block space-after="20mm" border-style="solid" border-                    ❶
    color="#0033CC" border-width="5mm" font-size="30pt">
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>
<fo:block space-before="20mm" padding="5mm 5mm 5mm 5mm" border-              ❶
    style="solid" border-color="#0033CC" border-width="5mm" font-
    size="30pt">
   INHALT INHALT INHALT INHALT INHALT INHALT INHALT
</fo:block>
```

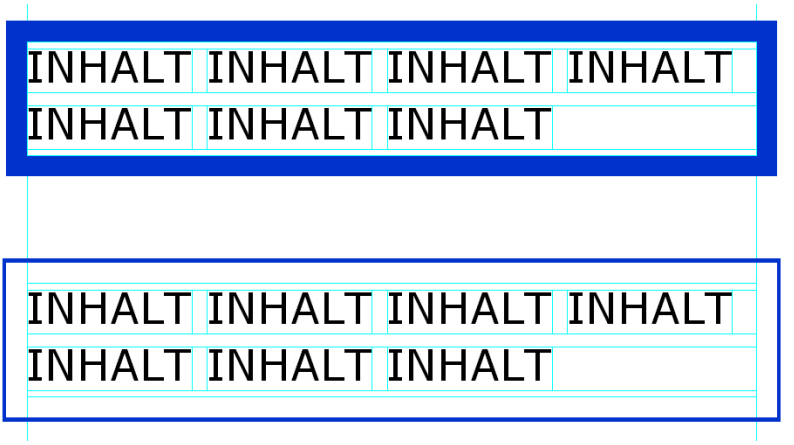


Abb. 6-17 Formatierer-Ansicht

❶ Die Attribute space-after und space-before erzeugen einen Abstand zwischen den beiden Blöcken. Da sie nicht aufaddiert werden, ist der Abstand 20mm, und das Resultat entspricht dem des vorangegangenen Beispiels, in dem ein Rand von 20mm mit Hilfe des margin-bottom-Attributs erzeugt wurde.

Zur Verdeutlichung der Wirkung von Vorschüben ein weiteres, komplexeres Beispiel:

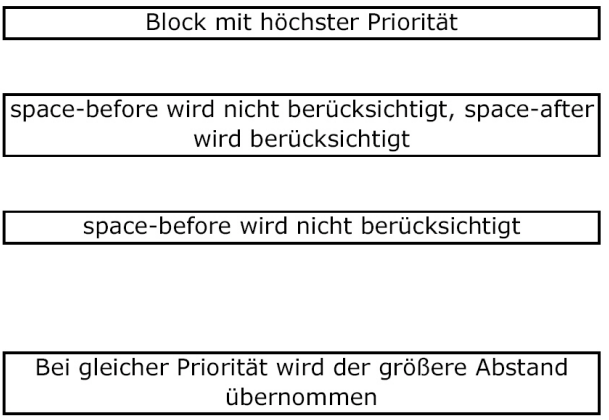
```
...
<fo:block text-align="center" border-width="2pt" border-style="solid"
   space-after.precedence="force" space-after="10mm">                    ❶
      Block mit höchster Priorität
</fo:block>
<fo:block text-align="center" border-width="2pt" border-style="solid"
   space-before="10000mm" space-before.precedence="5"                    ❶
   space-after="10mm" space-after.precedence="5">                        ❷
      space-before wird nicht berücksichtigt, space-after wird
          berücksichtigt
</fo:block>
<fo:block text-align="center" border-width="2pt" border-style="solid"
   space-before="10000mm" space-before.precedence="4"                    ❷
   space-after="20mm">                                                   ❸
      space-before wird nicht berücksichtigt
</fo:block>
<fo:block text-align="center" border-width="2pt" border-style="solid"
   space-before="10mm">                                                  ❸
      Bei gleicher Priorität wird der größere Abstand übernommen
</fo:block>
...
```

❶ Da der Wert force für das Attribut space-after.precedence der höchstmögliche ist, wird der space-after-Abstand eingehalten und der space-before-Abstand des zweiten Blockes ignoriert.

❷ In diesem Fall entscheidet die höhere Priorität über den Einsatz des Vorschubs. Da die space-after.precedence größer ist als die space-before.precedence des folgenden Blocks, wird der space-after-Abstand des ersteren Blocks verarbeitet.

❸ An dieser Stelle gibt es keine expliziten Angaben zur Priorität. Beide Prioritäten haben folglich den Wert "0" und sind gleich. In diesem Fall wird die Spezifikation des größeren Abstands der beiden benachbarten Blöcke verwendet, in diesem Fall der space-after-Vorschub des oberen Blocks.

Abb. 6-18
Formatierer-Ansicht



6.5 Inzeilige Formatierung

Vgl. Abschnitt 10.2

Das Element `<fo:inline>` unterscheidet sich in erster Linie von `<fo:block>` dadurch, dass keine Absätze gebildet werden, sondern die Inhalte inzeilig eingefügt werden. Es sind daher so gut wie alle Attribute, die für das Element `<fo:block>` gültig sind, auch für `<fo:inline>` anwendbar.

Eine Ausnahme bilden die `margin`-Attribute, die für `<fo:inline>` nicht erlaubt und hier unsinnig sind. Da inzeilige Inhalte ebenfalls in Blöcken (in inzeiligen) formatiert werden, können andere Abstands-Attribute wie `space-after`, `space-end`, `padding` u. a. verwendet werden.

Typischerweise wird das Element `<fo:inline>` benutzt, um einzelne Zeichen, Worte und Wortfolgen hervorzuheben. Beispiele in diesem Buch sind die Element- und Attributsbezeichnungen, die in einer `Monodickten-Schrift` erscheinen, aber auch die Hervorhebungen durch **fette**, *kursive* oder unterstrichene Darstellung.

Eine wichtige Anmerkung ist zum Zeilenumbruch innerhalb von `<fo:inline>` zu machen. Mit `<fo:inline>` eingefügte Inhalte werden nicht automatisch zwischen den Zeilen umbrochen! Hierzu ist es notwendig, das Attribut `keep-together.within-line="no"` einzufügen, weil der Vorgabewert für dieses Attribut `yes` ist.

6.6 Typografische Mittel

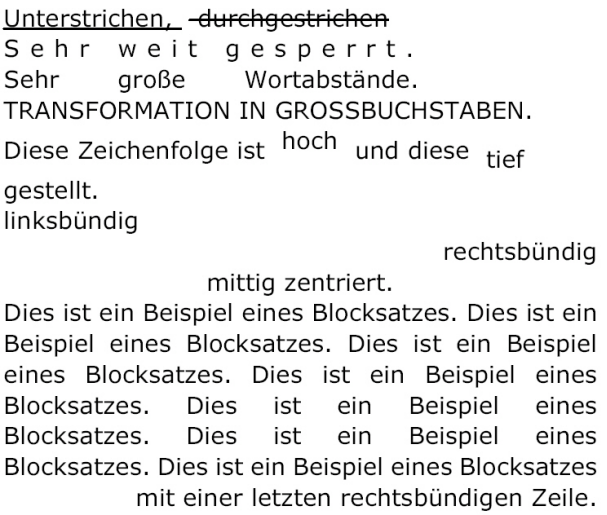
XSL-FO zielt auf die – automatisierte – Gestaltung in einem Satz- und Umbruchsystem. Für die Typografie bietet XSL-FO deshalb die von anderen Satzsystemen gewohnten, teilweise darüber hinausgehenden

Mittel in Form verschiedener Attribute und Elemente. Sie werden in den folgenden Abschnitten behandelt.

An dieser Stelle ein Beispiel für einige typografische Details:

```
...
<fo:block>
   <fo:inline text-decoration="underline">
      Unterstrichen,
   </fo:inline>
   <fo:inline text-decoration="line-through">
      durchgestrichen
   </fo:inline>
</fo:block>
<fo:block letter-spacing="3mm">
   Sehr weit gesperrt.
</fo:block>
<fo:block word-spacing="15mm">
   Sehr große Wortabstände.
</fo:block>
<fo:block text-transform="uppercase">
   Transformation in Grossbuchstaben.
</fo:block>
<fo:block >
   Diese Zeichenfolge ist
   <fo:inline baseline-shift="30%">
      hoch
   </fo:inline>
   und diese
   <fo:inline baseline-shift="-30%">
      tief gestellt.
   </fo:inline>
</fo:block>
<fo:block text-align="left">
   linksbündig
</fo:block>
<fo:block text-align="right">
   rechtsbündig
</fo:block>
<fo:block text-align="center">
   mittig zentriert.
</fo:block>
<fo:block text-align="justify" text-align-last="right">
   Dies ist ein Beispiel eines Blocksatzes. Dies ist ein Beispiel eines
       Blocksatzes. Dies ist ein Beispiel eines Blocksatzes. Dies ist ein
       Beispiel eines Blocksatzes. Dies ist ein Beispiel eines
       Blocksatzes. Dies ist ein Beispiel eines Blocksatzes. Dies ist ein
       Beispiel eines Blocksatzes mit einer letzten rechtsbündigen Zeile.
</fo:block>
...
```

Abb. 6-19
Formatierer-Ansicht



6.6.1 Textausrichtung

Vgl. Abschnitt 10.3

Die Attribute `text-align` und `text-align-last` legen fest, wie der textliche Inhalt innerhalb eines Blocks ausgerichtet werden soll. Die Eigenschaft `text-align-last` ermöglicht eine gesonderte Behandlung für die letzte Zeile eines Blocks. Die möglichen Werte für die Text-Ausrichtung sind:

- `center` (mittig zentriert)
- `left` (linksbündig, rechts flatternd)
- `right` (rechtsbündig, links flatternd)
- `justify` (Blocksatz, jede Zeile wird zwischen den Wörtern auf gleiche Breite ausgetrieben)
- `inside` (zum Bund hin zentriert; linke Seite rechtsbündig – rechte Seite linksbündig)
- `outside` (zum äußeren Beschnittrand zentriert; linke Seite linksbündig – rechte Seite rechtsbündig)
- `start` (Bündigkeit mit dem `start`-Bereich; bei Sprachen mit einer anderen Schreibrichtung als `lr-tb` von Bedeutung)
- `end` (Bündigkeit mit dem `end`-Bereich; bei Sprachen mit einer anderen Schreibrichtung als `lr-tb` von Bedeutung)

6.6.2 Einrückungen

Es wurden die zwei Absatzeinrückungsattribute `start-indent` und `end-indent` vorgestellt. Neben diesen beiden Attributen gibt es noch `text-indent` und `last-line-end-indent`.

Vgl. Abschnitt 6.4.3

Während `start-indent` und `end-indent` die Einrückung eines ganzen Blocks, d. h. aller Zeilen dieses Blocks im Start- bzw. Endbereich bewirkt, greift das Attribut `text-indent` lediglich auf die erste Zeile im Startbereich zu. Das Attribut `last-line-end-indent` bewirkt bei einer positiven Maßzahl die Einrückung des Absatzes im Start-, bei einer negativen Zahl im Endbereich.

Beispiel:

```
<fo:block text-align="justify" start-indent="10mm" end-
    indent="10mm" text-indent="-10mm">
   Hängender Einzug, Hängender Einzug, Hängender Einzug, Hängender Einzug,
       Hängender Einzug, Hängender Einzug, Hängender Einzug, Hängender
       Einzug, Hängender Einzug, Hängender Einzug
</fo:block>
<fo:block space-before="5mm" text-align="justify" text-indent="10mm">
   Einzug der ersten Zeile, Einzug der ersten Zeile, Einzug der ersten
       Zeile, Einzug der ersten Zeile, Einzug der ersten Zeile, Einzug der
       ersten Zeile
</fo:block>
```

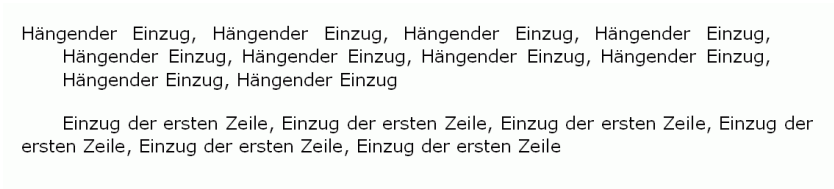


Abb. 6-20 Formatierer-Ansicht

6.6.3 Die Schriftart/Der Font

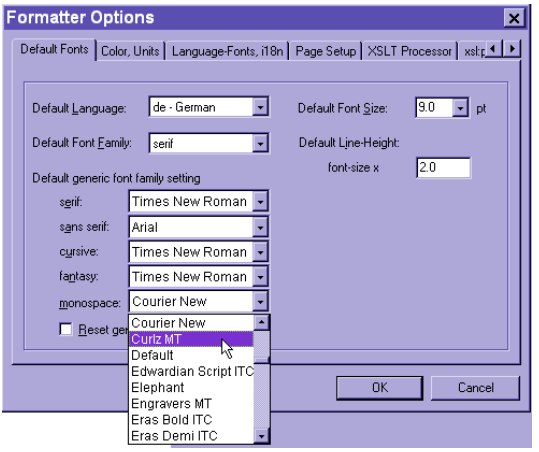
Das Attribut `font-family` bewirkt die Wahl einer Schriftart bzw. eines Fonts. Der Attributwert wird auf die Unterstrukturen im zu verarbeitenden Dokument vererbt.

An dieser Stelle sei auf mögliche Fehlerquellen in der Praxis hingewiesen.

Sollte der Antenna House Formatter verwendet werden, so findet man alle durch das System unterstützten Schriftarten/Fonts mit ihrem genauen Namen unter den Formatter Options.

Vgl. Kapitel 8

Abb. 6-21
Formatter Options



In diesen Optionen lassen sich Fonts als Vorgabewerte festlegen, die im Fall von nicht definierten Schriften im Stylesheet ersatzweise herangezogen werden.

Vgl. Abschnitt 10.1

Es empfiehlt sich jedoch, alle verwendeten Fonts und Schriftgrößen im Stylesheet selbst zu definieren, da so im Falle der Verarbeitung an einem anderen Rechner oder mit einem anderen Formatierer eine mögliche Fehlerquelle eliminiert ist.

6.6.4 Die Schriftgröße

Vgl. Abschnitt 10.1

Mit dem Attribut `font-size` wird die Schriftgröße festgelegt. Zulässig sind alle XSL-FO-Maßeinheiten, darunter die zur Angabe von Schriftgrößen üblichen Punkte unter Angabe der Abkürzung `pt`. Ebenfalls möglich, aber unüblich, ist die Angabe in Prozent, relativ zur vererbten Größe, d. h. der in der gegebenen Umgebung geltenden Schriftgröße.

Nach den deutschen Normen sind alle typografischen Maße in Millimetern anzugeben. Im Hinblick auf die Verwendung in XSL-FO mit den gängigen Formatierern ist darauf hinzuweisen, dass bei Maßen lediglich eine Stelle hinter dem Komma verwertet wird. Bei der Angabe in Millimetern ist also die kleinste Maßzahl ein Zehntel-Millimeter. Punktangaben können deshalb wesentlich genauer sein, weil ein Zehntel-Punkt lediglich etwa ein Drittel des Zehntel-Millimeters darstellt.

Der Attributwert für die Schriftgröße wird auf die Unterstrukturen im zu verarbeitenden Dokument vererbt.

6.6.5 Die Zeilenhöhe/der Zeilenabstand

Das Attribut `line-height` bewirkt die Festlegung der Zeilenhöhe und damit den Zeilenabstand. Auch hier sind alle XSL-FO-Maßeinheiten zulässig, ebenso die Angabe einer Prozentzahl.

In der Praxis empfiehlt es sich, diese Angabe immer mit anzugeben. Bei Nichtspezifikation von Zeilenhöhe und Schriftgröße im Stylesheet besteht die Gefahr, dass diese durch den Vorgabewert des Formatierers ergänzt werden.

Vgl. Abschnitt 10.1

Der Attributwert für die Zeilenhöhe wird auf die Unterstrukturen im zu verarbeitenden Dokument vererbt.

6.6.6 Der Schriftschnitt/Die Fontausprägung

Mit den Attributen `font-style` und `font-weight` lässt sich der Schriftschnitt steuern. Die Attributwerte für die Schriftschnitte werden auf die Unterstrukturen im zu verarbeitenden Dokument vererbt.

Vgl. Abschnitt 10.5

Das Attribut `font-style` ermöglicht unterschiedliche Kursivdarstellungen. Mit dem Attributwert `italic` wird eine Font-Variante bezeichnet, die im Namen den Ausdruck *kursiv* o. Ä. enthält. Mit den Attributwerten `oblique` bzw. `backslant` wird eine Font-Darstellung bezeichnet, in der der normale Schriftschnitt nach rechts bzw. links geneigt ist. Mit dem Attributwert `normal` wird der normale Schriftschnitt bezeichnet. In den meisten Umgebungen werden lediglich die Schriftschnitte `normal` und `italic` verfügbar sein. Bei Fehlen der entsprechenden Schriftschnitte ignoriert der Formatierer die nicht unterstützten Schnitte.

Mit `font-weight` lässt sich die Fettausprägung des Fonts festlegen. Zulässig sind die Werte: `bold`, `bolder`, `lighter`, `100`, `200` usw. bis `900`. In der gängigen Praxis wird lediglich ein Schriftschnitt für `bold` (**fett**) zur Verfügung stehen.

6.6.7 Unterstreichung, Überstreichung und Durchstreichung

Das Attribut `text-decoration` bewirkt die Unter-, Über- oder Durchstreichung von Text. Wird das Attribut auf ein Block-Element angewendet, dann gilt der Attributwert für alle untergeordneten Elemente. Wird es auf ein inzeiliges Element angewendet, dann gilt es lediglich für den Inhalt des gegebenen Elements.

Vgl. Abschnitt 10.19

Das Attribut kann folgende Werte annehmen:

- `underline` (einfache Unterstreichung)
- `overline` (einfache Überstreichung)
- `line-through` (einfache Durchstreichung)

- `none` (keine Unter-, Über- oder Durchstreichung)

`text-decoration` ist kein Font-Merkmal, sondern Leistungsmerkmal des Formatierers.

6.6.8 Silbentrennung

Vgl. Abschnitt 10.4

Die Silbentrennung ist zwar kein direktes typografisches Merkmal, beeinflusst aber in hohem Maße die Qualität des Satzbildes durch enge Wortzwischenräume, einen guten Zeilenumbruch und ein einheitlich dichtes Graubild der Textblöcke. Für den Zeilenumbruch bietet XSL-FO ein differenziertes Instrumentarium zur Spezifikation. Die Qualität des Satzergebnisses ist aber ein Merkmal des verwendeten Formatierers, weil dieser die Spezifikationen nach eigenen Regeln für den Zeilenumbruch umsetzt.

Die Einstellmöglichkeiten für die Silbentrenn-Attribute, die XSL-FO anbietet, sind:

- `hyphenate` (Silbentrennung gewünscht oder unerwünscht; mögliche Attributwerte deshalb `true` und `false`; Vorgabewert `false`)
- `hyphenation-character` (Trennzeichen; Vorgabewert ist der gewohnte Trennstrich)
- `hyphenation-keep` (Silbentrennung am Ende einer Spalte oder Seite; Attributwerte `column`, `page` verhindern die Silbentrennung am Ende einer Spalte bzw. Seite; Vorgabewert ist `auto`, d. h. keine Besonderheit für die Trennung)
- `hyphenation-ladder-count` (Angabe der maximalen Zahl aufeinander folgender Zeilen, für die Silbentrennung erlaubt sein soll; Vorgabewert begrenzt die Zahl der Zeilen nicht)
- `hyphenation-push-character-count` (spezifiziert für ein Wort die Mindest-Buchstabenzahl, die nach dem Trennzeichen in der folgenden Zeile stehen müssen; Vorgabewert ist `2`)
- `hyphenation-remain-character-count` (spezifiziert für ein Wort die Mindest-Buchstabenzahl, die vor dem Trennzeichen stehen müssen; Vorgabewert ist `2`)
- `country` (Attributwert ist eine standardisierte Landeskurzbezeichnung; kein Vorgabewert)
- `language` (Attributwert ist eine standardisierte Sprachkurzbezeichnung; kein Vorgabewert; für Deutsch `de`)

Alle Attribute vererben ihre Attributwerte auf die Unterstrukturen.

Beispiel:

```
<fo:block hyphenate="true"
          hyphenation-character="!"
```

```
            hyphenation-push-character-count="2"
            hyphenation-remain-character-count="3"
            language="de">
Lange Wörter wie Feuerwehrbrandschutzmeister müssen unbedingt getrennt
    werden.
<fo:block>
```

Beim Antenna House Formatter ist die für die Silbentrennung in Deutsch die Spezifikation `hyphenate="true" xml:lang="de"` üblich. Für die oben genannten weiteren Silbentrenn-Attribute gelten die entsprechenden Vorgabewerte von XSL-FO. Für die Eingabe eines Trennzeichens, das nur dann gesetzt wird, wenn das Wort, in dem sich dieses Zeichen befindet, am Ende einer Zeile getrennt wird, wird der „weiche" Trennstrich verwendet (Unicode-Zeichen `&#x00AD;`).

6.6.9 Horizontale Linien in Blöcken

Das Element `<fo:leader>` dient zur Erzeugung von horizontalen Linien innerhalb von Blöcken bzw. Zeilen. Diese werden in der Praxis oft als Führungslinien in Inhaltsverzeichnissen zwischen dem Text und der rechts ausgerückten Seitenzahl eingesetzt. Die Linien können, wie auch die Rahmen, gestrichelt, gepunktet oder durchgehend sein, oder aber aus selbst definierten Zeichen bestehen.

Vgl. Abschnitt 10.21

Zur genauen Festlegung stehen eine Reihe von Attributen zur Verfügung:

- `leader-length` (Länge der Linie; Vorgabewerte sind minimal `0pt`, optimal `12pt`, maximal `100%`, d. h. normalerweise füllt die Linie den textleeren Raum einer Zeile vollständig aus)
- `leader-pattern` (Füllzeichenmuster; mögliche Werte sind: `space` (Leerzeichen), `dots` (punktiert), `rule` (liniiert) oder `use-content` (beliebiges Zeichenmuster, das als Inhalt im `<fo:leader>`-Element enthalten ist)
- `rule-style` (Muster der Linie; mögliche Werte: `dotted` (punktiert), `dashed` (gestrichelt), `solid` (durchgehend), `double` (doppelt), `groove` (s. Beispiel, Abb. 6-22) und `ridge` (s. Beispiel, Abb. 6-22)
- `rule-thickness` (Stärke der Linie; Vorgabewert ist `1pt`)
- `color` (Farbe; Vorgabewert wird durch den Formatierer bestimmt, üblicherweise schwarz)

Darüber hinaus sind eine Reihe weiterer Attribute definiert, die allerdings in der Praxis keine Bedeutung haben oder bisher von keinem Formatierer unterstützt werden.

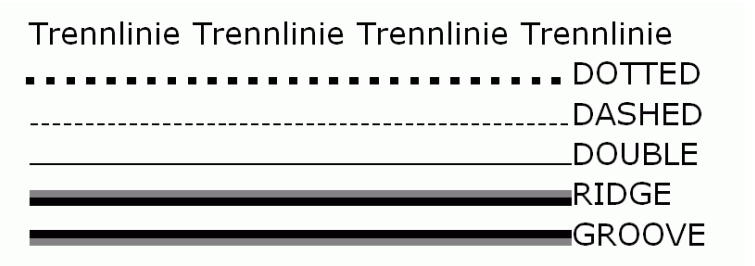
Beispiel:

```
<fo:block>
   <fo:leader leader-pattern="use-content" leader-length="90%">          ❶
      Trennlinie
   </fo:leader>
   <fo:leader leader-pattern="rule" leader-length="70%" rule-            ❷
       style="dotted" rule-thickness="5pt"/>DOTTED
   <fo:leader leader-pattern="rule" leader-length="70%" rule-            ❷
       style="dashed" rule-thickness="1pt"/>DASHED
   <fo:leader leader-pattern="rule" leader-length="70%" rule-            ❷
       style="double" rule-thickness="1pt"/>DOUBLE
   <fo:leader leader-pattern="rule" leader-length="70%" rule-            ❷
       style="ridge" rule-thickness="10pt"/>RIDGE
   <fo:leader leader-pattern="rule" leader-length="70%" rule-            ❷
       style="groove" rule-thickness="10pt"/>GROOVE
</fo:block>
```

❶ Durch Setzen des Attributs `leader-pattern` auf den Wert `use-content` wird der Inhalt des Elements `<fo:leader>` zur Erstellung der Führungslinie herangezogen.

❷ Hier sind verschiedene Muster mit unterschiedlichen Linienstärken aufgelistet.

Abb. 6-22 Formatierer-Ansicht



Typischerweise wird das Element `<fo:leader>` Führungslinie in Inhaltsverzeichnissen verwendet.

Beispiel:

```
<fo:block text-align-last="justify">
   Kapitel: Horizontale Linien
   <fo:leader leader-pattern="rule" rule-style="dotted" rule-
       thickness="1pt"/>
   Seite 18
</fo:block>
```

Abb. 6-23 Formatierer-Ansicht

Kapitel: Horizontale Linien ..... Seite 18

6.6.10 Die Groß- und Kleinschreibung

Das Attribut `text-transform` dient zur Umwandlung von Groß- und Kleinschreibung. Es bietet vier Möglichkeiten:

- `capitalize` wandelt den ersten Buchstaben eines jeden Worts in einen Großbuchstaben um.
- `uppercase` wandelt alle Kleinbuchstaben in Großbuchstaben um.
- `lowercase` wandelt alle Großbuchstaben in Kleinbuchstaben um.
- `none` behält die vorhandene Groß- und Kleinschreibung bei. Dieser Wert ist der Vorgabewert.

6.6.11 Zeichenabstände

Die Sperrung von Zeichen lässt sich mit Hilfe des Attributs `letter-spacing` steuern. Es definiert den Leerraum zwischen zwei Buchstaben innerhalb eines Wortes über den normalen Abstand zwischen den Zeichen hinausgehend. Der Wert wird in absoluten Maßen angegeben und kann sowohl negativ als auch positiv sein. Der Vorgabewert ist `normal`. Negative Werte für den Leerraum werden von den Formatierern unterschiedlich behandelt, üblicherweise ignoriert.

6.6.12 Wortabstände

Das Attribut `word-spacing` reguliert den Abstand zwischen zwei Wörtern über den normalen Wortabstand hinausgehend. Auch hier müssen absolute Maße angegeben werden und auch diese können sowohl positiv wie negativ sein. Der Vorgabewert ist `normal`. Negative Werte für den Wortabstand werden von den Formatierern unterschiedlich behandelt, üblicherweise ignoriert.

6.6.13 Hoch- und Tiefstellung

Vgl. Abschnitt 10.11

Das Attribut `baseline-shift` verschiebt die Grundlinie dieser Textpassage relativ zur Umgebung. Diese Verschiebung kann in Prozent oder absolut, mit einem positiven oder negativen Wert angegeben werden oder aber unter Verwendung der Werte `super` (hochgestellt) und `sub` (tiefgestellt). Der Vorgabewert `baseline` erzeugt keine Verschiebung.

Das Attribut `vertical-align`, für vertikale Ausrichtung, ist in inzeiligen Elementen alternativ einzusetzen, darüber hinaus aber in Tabellenzellen. Seine möglichen Werte sind: `baseline` (Grundlinie, keine Verschiebung; Vorgabewert), `middle` (mittig innerhalb einer Tabellenzel-

le), sub (tiefgestellt), super (hochgestellt), text-top (in einer Tabellenzelle oben angeschlagen), text-bottom (in einer Tabellenzelle unten angeschlagen), ein Prozentwert (positive Werte verschieben nach oben, negative nach unten), ein absoluter Wert (positive Werte verschieben nach oben, negative nach unten), top (in der gegebenen Zeile oben angeschlagen) oder bottom (in der gegebenen Zeile unten angeschlagen).

6.6.14 Zeilen-, Spalten- und Seitenumbruch kontrollieren

Für die gezielte Umbruchsteuerung stehen die von anderen Satzsystemen gewohnten Attribute zur Verfügung. Sie lassen sich zu vier Gruppen ordnen:

- Attribute für die Vermeidung der so genannten „Schusterjungen“ (orphans) und „Hurenkinder“ (widows)
- Umbruchverhindernde Attribute, d. s. keep-together, keep-with-next und keep-with-previous
- Umbrucherzwingende Attribute, d. s. break-after und break-before
- Umbrucherzwingende oder -verhindernde Attribute, alternativ zu benutzen, d. s. page-break-after, page-break-before und page-break-inside

Der Einsatz dieser Attribute und die Wirkungen ihrer Attributwerte:

- Mit dem orphans-Attribut („Schusterjungen“) wird spezifiziert, wie viele Zeilen eines Blocks am Fuß einer Spalte oder Seite mindestens zusammengehalten werden müssen. Der Vorgabewert ist 2.
- Mit dem widows-Attribut („Hurenkinder“) wird spezifiziert, wie viele Zeilen eines Blocks am Kopf einer Spalte oder Seite mindestens zusammengehalten werden müssen. Der Vorgabewert ist 2.
- Die umbruchverhindernden Attribute keep-together (Zusammenhalten des gegebenen Blocks oder inzeiligen Elements), keep-with-next (Zusammenhalten des gegebenen Blocks mit dem folgenden Block bzw. des gegebenen inzeiligen Elements mit dem nachfolgenden Text) und keep-with-previous (Zusammenhalten des gegebenen Blocks mit dem vorausgehenden Block bzw. des gegebenen inzeiligen Elements mit dem vorausgehenden Text) haben die gleiche Anwendungslogik. Ihr Vorgabewert ist .within-line=auto, .within-column=auto, .within-page=auto, d. h. es wird keine Bedingung für das Zusammenhalten gesetzt. Um das Zusammenhalten unter allen Umständen zu erzwingen, wird das Attribut auf den Wert always gesetzt. Ganze Zahlen als Attributwert, z. B. 3, werden kontextab-

hängig vom Formatierer umgesetzt und schwächen den Zwang zum Zusammenhalten ab.

- Die umbrucherzwingenden Attribute `break-after` und `break-before` werden für Blöcke gesetzt und bewirken durch ihre Attributwerte einen Umbruch danach bzw. davor. Der Vorgabewert ist `auto`, d. h. es wird keine Bedingung für den Umbruch gesetzt. Mit den Attributwerten `column` (Spalte), `page` (Seite), `even-page` (linke bzw. geradzahlige Seite) und `odd-page` (rechte bzw. ungeradzahlige Seite) wird ein entsprechender Umbruch danach bzw. davor bewirkt. Ggf. wird eine Leerseite eingefügt.
- Die alternativ einzusetzenden Attribute für die Erzwingung bzw. Vermeidung von Umbrüchen `page-break-after` (Umbruch nach dem gegebenen Block), `page-break-before` (Umbruch vor dem gegebenen Block) und `page-break-inside` (Umbruch innerhalb des gegebenen Blocks) beziehen sich lediglich auf den Seitenumbruch. Sie stellen abgekürzte Schreibweisen für die oben beschriebenen Attribute dar. Der Vorgabewert `auto` für alle diese Attribute weder verbietet noch erzwingt einen Umbruch.

  Das Attribut `page-break-inside` hat lediglich einen alternativen Attributwert `avoid`, mit dem der gegebene Block zusammengehalten wird.

  In `page-break-after` und `page-break-before` hat `avoid` die gleiche Wirkung in Bezug auf den nachfolgenden bzw. vorausgehenden Block. Weitere mögliche Attributwerte sind `always` (Seitenumbruch unter allen Bedingungen), `left` (Seitenumbruch, so dass die folgende bzw. vorausgehende Seite eine linke, geradzahlige Seite ist) und `right` (Seitenumbruch, so dass die folgende bzw. vorausgehende Seite eine rechte, ungeradzahlige Seite ist). Ggf. wird eine Leerseite eingefügt.

6.7 Listen, Aufzählungen und Beschreibungslisten

Vgl. Abschnitt 10.12

XSL-FO bietet ein spezielles Konzept für listenartige Strukturen, mit dem sich so genannte geordnete Listen (jedes Listenelement bekommt eine Ordnungsbezeichnung, typischerweise eine fortlaufende Nummer), ungeordnete Listen bzw. Aufzählungen (jedes Listenelement bekommt ein gleiches Vorzeichen, typischerweise einen Spiegelstrich oder fetten Punkt oder – wie in diesem Buch – ein Kästchen) oder Beschreibungslisten (jedes Listenelement bekommt eine den Inhalt des jeweiligen Listenelements beschreibende Bezeichnung) gestalten lassen.

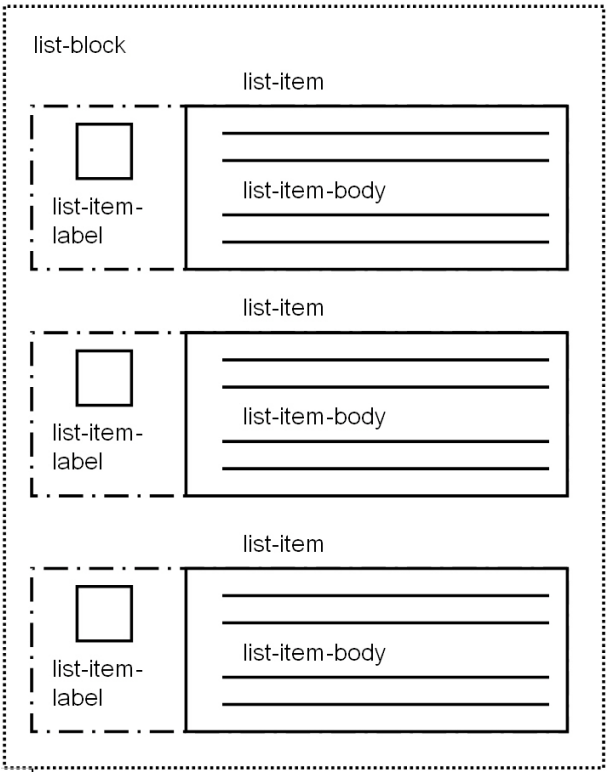
Das XSL-FO-Konzept für Listen besteht aus vier spezifischen Elementen:

- `<fo:list-block>` (Liste)
- `<fo:list-item>` (Listenelement)
- `<fo:list-item-label>` (Listenelement-Etikett)
- `<fo:list-item-body>` (Listenelement-Inhalt)

Innerhalb von `<fo:list-item-label>` und `<fo:list-item-body>` stehen die normalen Blockelemente, typischerweise `fo:block`.

Die hierarchischen Beziehungen dieser listenspezifischen Elemente zeigt die folgende Grafik:

Abb. 6-24
Listenblöcke



Der XSL-FO-Code einer einfachen Liste sieht wie folgt aus:

```
<fo:block>
   <fo:list-block>
      <fo:list-item>
         <fo:list-item-label>
            <fo:block>
               1.
            </fo:block>
         </fo:list-item-label>
         <fo:list-item-body>
            <fo:block>
               Der Text zum ersten Listenelement
```

```
            </fo:block>
         </fo:list-item-body>
      </fo:list-item>
      <fo:list-item>
         <fo:list-item-label>
            <fo:block>
               2.
            </fo:block>
         </fo:list-item-label>
         <fo:list-item-body>
            <fo:block>
               Der Text zum zweiten Listenelement
            </fo:block>
         </fo:list-item-body>
      </fo:list-item>
   </fo:list-block>
</fo:block>
```

Die typografische Umsetzung des hier vorgestellten Codes ist jedoch sehr unansehnlich, da wesentliche Informationen für die Gestaltung fehlen. Beispielsweise wird ohne definierte Abstandsattribute der Inhalt des Listenelement-Etiketts durch den Listenelement-Inhalt überlagert.

In der folgenden Grafik werden die verschiedenen Attribute, die zur Abstandsregelung dienen, grafisch dargestellt:

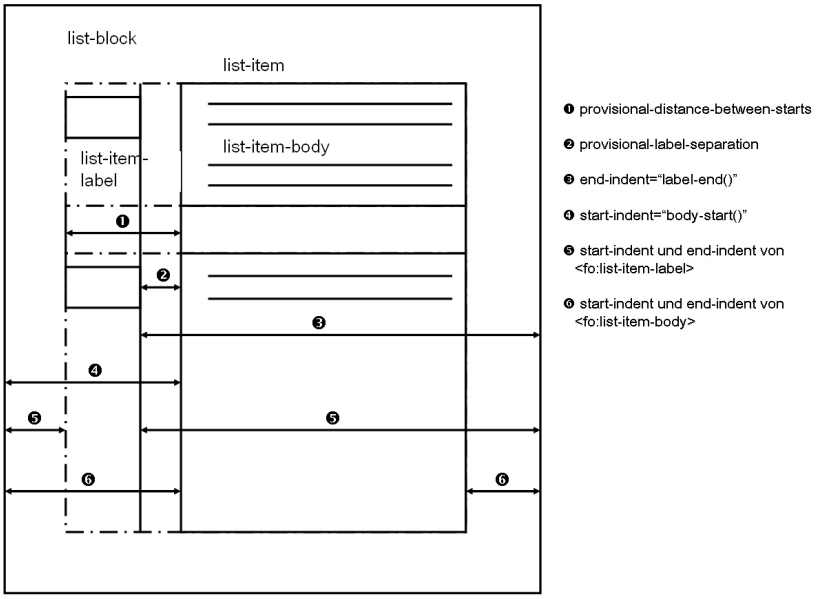


Abb. 6-25
Abstände in Listen

❶ `provisional-distance-between-starts` gibt den Abstand zwischen dem Beginn des Listenelement-Etiketts und dem Beginn des Listenelement-Inhalts an. Als Attributwert wird ein absoluter Wert in einer beliebigen, unterstützten Maßeinheit erwartet.

❷ `provisional-label-separation` gibt den Abstand zwischen dem Ende des Listenelement-Etiketts und dem Beginn des Listenelement-Inhalts an. Als Attributwert wird ein absoluter Wert in einer beliebigen, unterstützten Maßeinheit erwartet.

❸ `end-indent="label-end()"` gibt den Abstand vom Ende des Listenelement-Etiketts bis zum rechten Rand an. Die Funktion `label-end()` ermittelt hierbei diesen Abstand automatisch unter Berücksichtigung der `provisional-distance-between-starts`- und `provisional-label-separation`-Werte.

❹ `start-indent="body-start()"` gibt den Abstand vom Ende des Listenelement-Etiketts bis zum Beginn des Listenelement-Inhalts an. Die Funktion `body-start()` ermittelt hierbei diesen Abstand automatisch unter Berücksichtigung der `provisional-distance-between-starts`- und `provisional-label-separation`-Werte.

❺ `start-indent` und `end-indent` von `<fo:list-item-label>` können ebenso durch Angabe von festen Werten erfolgen.

❻ `start-indent` und `end-indent` von `<fo:list-item-body>` können ebenso durch Angabe von festen Werten erfolgen.

Ein Beispiel mit einer Liste in zwei Ebenen:

```
<fo:block text-align="justify" font-size="15pt">
<fo:list-block provisional-distance-between-starts="10mm" end-          ❶
    indent="10mm" start-indent="10mm">
   <fo:list-item>
      <fo:list-item-label end-indent="label-end()">                     ❷
         <fo:block> 1. </fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">                   ❸
         <fo:block> Bodytext Bodytext Bodytext Bodytext Bodytext Bodytext
             Bodytext Bodytext Bodytext </fo:block>
      </fo:list-item-body>
   </fo:list-item>
   <fo:list-item>
      <fo:list-item-label end-indent="label-end()">                     ❷
         <fo:block> 2. </fo:block>
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">                   ❸
         <fo:block> Bodytext Bodytext Bodytext Bodytext Bodytext Bodytext
             Bodytext Bodytext Bodytext </fo:block>
      </fo:list-item-body>
   </fo:list-item>
   <fo:list-item>
      <fo:list-item-label end-indent="label-end()">                     ❷
      </fo:list-item-label>
      <fo:list-item-body start-indent="body-start()">                   ❸
         <fo:list-block provisional-distance-between-starts="10mm">     ❹
            <fo:list-item>
```

```
                <fo:list-item-label end-indent="label-end()">          ❷
                   <fo:block> 2.1 </fo:block>
                </fo:list-item-label>
                <fo:list-item-body start-indent="body-start()">        ❸
                   <fo:block> Bodytext Bodytext Bodytext Bodytext Bodytext
                       Bodytext Bodytext Bodytext Bodytext </fo:block>
                </fo:list-item-body>
             </fo:list-item>
          </fo:list-block>
       </fo:list-item-body>
    </fo:list-item>
</fo:list-block>
</fo:block>
```

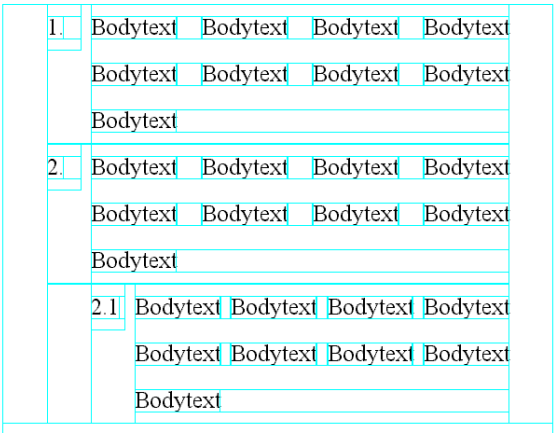


Abb. 6-26
Formatierer-Ansicht

❶ Das Attribut `provisional-distance-between-starts` definiert den Abstand zwischen dem Start-Rand des Listenelement-Etiketts und dem Start-Rand des Listenelement-Inhalts. Die bereits bekannten Attribute `end-indent` und `start-indent` erzeugen Einrückungen am linken und rechten Rand.

❷ Der Wert des Attributs `end-indent` wird hier automatisch durch die Funktion `label-end()` ermittelt und bezeichnet den Abstand zwischen dem rechten Rand und dem Ende des Listenelement-Etiketts.

❸ Der Wert des Attributs `start-indent` wird hier automatisch durch die Funktion `body-start()` ermittelt und bezeichnet den Abstand zwischen dem linken Rand und dem Beginn des Listenelement-Inhalts.

❹ Eine Verschachtelung von Listen wird durch die Erzeugung einer zweiten Liste innerhalb des Elements `<fo:list-item-body>` erzeugt.

Zur Nummerierung von Listenelementen bleibt anzumerken, dass es keine XSL-FO-Elemente oder -Attribute gibt, die eine solche steuern bzw. generieren. Die Nummerierung muss, wenn sie automatisch erfolgen soll, mit dem XSLT-Element `<xsl:number>` erzeugt werden.

Vgl. Abschnitt 5.6

Die in ungeordneten Listen gern verwendeten Zeichen können entweder, wenn sie im Font nicht enthalten sind, aus anderen Fonts entliehen werden oder durch die Verwendung einer Grafik ersetzt werden.

6.8 Fußnoten

Vgl. Abschnitt 10.11

Fußnoten werden in XSL-FO mit dem Element `<fo:footnote>` erzeugt, der Textkörper zur Fußnote mit dem Element `<fo:footnote-body>`. Die beiden Elemente besitzen keine spezifischen Attribute, die zur Steuerung herangezogen werden könnten. Damit ist leider in XSL-FO für die Platzierung der Fußnoten kein Spielraum gegeben. Sie werden automatisch im Fuß des Fließ- oder Hauptbereichs der Seite platziert, in der sie im Text verankert sind. Bei einer mehrspaltigen Seitengestaltung behandeln die Formatierer die Fußnoten unterschiedlich (Überspannung aller Spalten vs. Platzierung unter jede Spalte).

Vgl. Abschnitt 5.6

Eine automatische Nummerierung der Fußnoten muss, wie bei den Listen, mit dem XSLT-Element `<xsl:number>` erzeugt werden.

Es wird davon ausgegangen, dass die Fußnoten im Text an der Stelle stehen, wo sie verankert sein sollen. Der Formatierer setzt die Fußnoten dann in den unteren Bereich des Hauptbereichs der Seite ein.

Beispiel:

```
<fo:block font-size="40pt"> Text </fo:block>
<fo:block font-size="40pt"> Text </fo:block>
<fo:block font-size="40pt"> Text </fo:block>
<fo:block font-size="40pt">
   Text
   <fo:footnote>                                                    ❶
      <fo:inline font-size="15pt" baseline-shift="super">1</        ❷
          fo:inline>
      <fo:footnote-body>                                            ❸
         <fo:block font-size="15pt">
            <fo:inline font-size="10pt" baseline-shift="super">1</
                fo:inline>
            Fußnotentext Fußnotentext Fußnotentext Fußnotentext
                Fußnotentext Fußnotentext Fußnotentext
         </fo:block>
      </fo:footnote-body>
   </fo:footnote>
</fo:block>
<fo:block font-size="40pt">
   Text
   <fo:footnote>                                                    ❶
      <fo:inline font-size="15pt" baseline-shift="super">2</        ❷
          fo:inline>
      <fo:footnote-body>                                            ❸
         <fo:block font-size="15pt">
```

```
            <fo:inline font-size="10pt" baseline-shift="super">2</
                fo:inline>
            Fußnotentext Fußnotentext Fußnotentext Fußnotentext
                Fußnotentext Fußnotentext Fußnotentext
        </fo:block>
      </fo:footnote-body>
   </fo:footnote>
</fo:block>
```

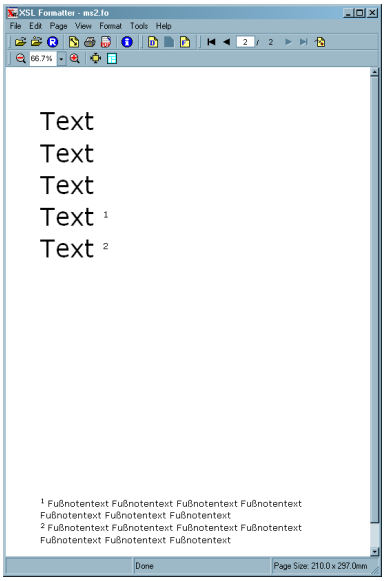


Abb. 6-27
Formatierer-Ansicht

❶ Das Element `<fo:footnote>` ist in seiner Funktion zweigegliedert. Alle Anweisungen vor dem Kindelement `<fo:footnote-body>` dienen als Ankerbezeichnungen, die im Textfluss an entsprechender Stelle erscheinen. Der Inhalt von `<fo:footnote-body>` wird als Fußnote behandelt.

❷ Das Element `<fo:inline>` wird hier eingesetzt, um die Fußnotennummer inzeilig darzustellen.

❸ Das Element `<fo:footnote-body>` enthält den eigentlichen Inhalt der Fußnote.

Typischerweise sollen Fußnoten vom Textbereich durch Linien oder/und größere Abstände getrennt werden. Diese Separatoren werden in XSL-FO mit einem eigenen `<fo:static-content>`-Element innerhalb der Seitenfolge definiert. Das Attribut `flow-name` erhält hierfür den Wert `xsl-footnote-separator`.

Hinweis: Dieses Konstrukt wird nicht von allen Formatierern unterstützt.

Beispiel:

```
<fo:page-sequence master-reference="Beispiel">
   <fo:static-content flow-name="xsl-footnote-separator">          ❶
```

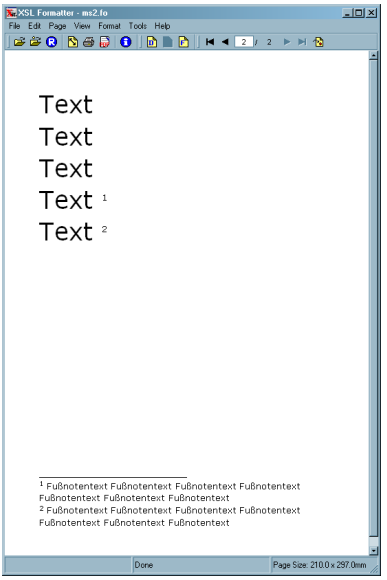
```
    <fo:block text-align-last="justify">
       <fo:leader leader-length="50%" rule-thickness="0.5pt"          ❷
           leader-pattern="rule"/>
    </fo:block>
</fo:static-content>
```

❶ Der Attributwert `xsl-footnote-separator` fügt den Inhalt des Elements `<fo:static-content>` in der gegebenen Seitenfolge jeweils vor den Fußnoten einer Seite ein. Treten in einer Seite keine Fußnoten auf, entfällt dieser Inhalt.

Vgl. Abschnitt 6.6.9

❷ Das Element `<fo:leader>` erzeugt die Trennlinie unter dem Text und über der ersten Fußnote einer Seite. Statt dieser Linie könnte auch beliebiger anderer statischer Inhalt stehen.

Abb. 6-28 *Formatierer-Ansicht*



6.9 Grafiken

Vgl. Abschnitt 10.16

Grafiken und andere bildliche Darstellungen, die außerhalb des XML-Dokuments gespeichert sind, werden in XSL-FO mit dem Element `<fo:external-graphic>` eingebunden. Das Element kann sowohl in einem Block als auch inzeilig verwendet werden.

Die Grafik wird mit Hilfe des Attributs `src` in das Dokument eingebunden. Als Wert erwartet das Attribut den relativen Pfad zur Bilddatei ausgehend von der XML-Instanz.

Zur Größenveränderung der eingebundenen Grafik können die folgenden Attribute genutzt werden:

- `content-height` (Höhenausdehnung der Grafik). Der Vorgabewert ist `auto`, d. h. die Grafikhöhe, wie in der Grafik selbst, wird übernommen. Der Wert `scale-to-fit` bewirkt, dass die Grafik an die Dimensionen eines vorgegebenen Rechtecks angepasst wird. Eine Länge in einer beliebigen, unterstützten Maßzahl als Attributwert führt ggf. zu einer Längenbeschränkung (Verkleinerung der Grafik). Ein Prozentwert als Attributwert führt zu einer entsprechenden Skalierung.
- `content-width` (Breitenausdehnung der Grafik). Die Attributwerte sind die gleichen wie für die Höhe und bewirken das Gleiche.

Wird die Größe bzw. Skalierung lediglich für eine Dimension spezifiziert, dann wird die Größe in der anderen Dimension proportional angepasst. Um Verzerrungen zu vermeiden, sollte jeweils nur eine der beiden Dimensionen spezifiziert werden.

Beispiel:

```
<fo:block>
   <fo:inline>
      <fo:external-graphic src="bilder/bild.bmp"/>                    ❶
   </fo:inline>
   <fo:inline>
      <fo:external-graphic src="bilder/bild.bmp" content-             ❷
          width="30mm"/>
   </fo:inline>
   <fo:inline>
      <fo:external-graphic src="bilder/bild.bmp" content-             ❸
          width="50mm" content-height="20mm"/>
   </fo:inline>
</fo:block>
```

❶ Der relative Pfad zur Datei `bilder/bild.bmp` wird als Wert dem obligatorischen Attribut `src` übergeben. Die Größe des Bildes wird ohne Skalierung aus dem Original übernommen.
❷ Das Attribut `content-width` gibt die Breite des Bildes an. Wird keine Höhe festgelegt, wird diese proportional angepasst.
❸ Hier wird sowohl die Höhe mit `content-height` als auch die Breite mit `content-width` festgelegt, daher kommt es zu einer Verzerrung.

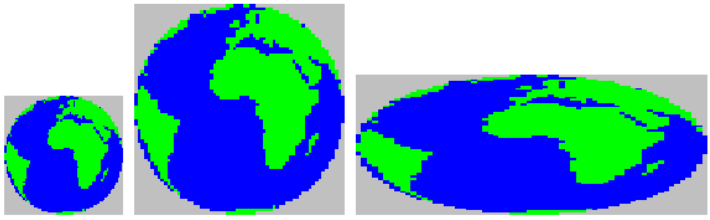


Abb. 6-29
Formatierer-Ansicht

Hintergrundbilder und -flächen

Vgl. Abschnitt 10.17

Zur Einbettung von Hintergrundbildern und -flächen gibt es zur typografischen Steuerung eine Reihe von Attributen, die in Blöcken eingesetzt werden können.

Die Attribute sind:

- `background-image` (Pfad zum Hintergrundbild). Als Wert erwartet das Attribut den relativen Pfad zur Bilddatei ausgehend von der XML-Instanz.
- `background-position-vertical` (vertikale Positionierung innerhalb des gegebenen Block-Rechtecks einschließlich der ggf. definierten `padding`-Fläche). Vorgabewert ist `0%` (beginnend am oberen Rand des Rechtecks). Als Wert kann auch ein Prozentwert zwischen `0%` und `100%` oder ein absoluter Wert in einer beliebigen, zulässigen Maßzahl angegeben werden. Damit verschiebt sich das Hintergrundbild innerhalb des gegebenen Rechtecks von oben nach unten. Weitere zulässige Werte: `top` (entspricht `0%`), `center` (`50%`) und `bottom` (`100%`).
- `background-position-horizontal` (horizontale Positionierung innerhalb des gegebenen Block-Rechtecks einschließlich der ggf. definierten `padding`-Fläche). Vorgabewert ist `0%` (beginnend am linken Rand des Rechtecks). Als Wert kann auch ein Prozentwert zwischen `0%` und `100%` oder ein absoluter Wert in einer beliebigen, zulässigen Maßzahl angegeben werden. Damit verschiebt sich das Hintergrundbild innerhalb des gegebenen Rechtecks von links nach rechts. Weitere zulässige Werte: `left` (entspricht `0%`), `center` (`50%`) und `right` (`100%`).
- `background-position` (abgekürzte Schreibweise zur Positionierung des Hintergrundbildes innerhalb des Rechtecks in beiden Dimensionen. Der erste Wert entspricht `background-position-vertical`, der zweite `background-position-horizontal`.
- `background-repeat` (Wiederholung des Hintergrundbildes innerhalb des Rechtecks). Der Vorgabewert ist `repeat` (das Hintergrundbild wird ggf. horizontal wie vertikal wiederholt). Beim Wert `x-repeat` wird lediglich ggf. horizontal wiederholt, bei `y-repeat` lediglich vertikal und bei `no-repeat` wird das Bild nicht wiederholt.
- `background-color` (Hintergrundfarbe an Stelle eines Hintergrundbildes). Als Attributwert wird eine Farb-Spezifikation erwartet.

Beispiel:

```
<fo:block background-image="bilder/bild.bmp" background-                ❶
    repeat="repeat" font-size="20pt">
   Hintergrundbild Hintergrundbild Hintergrundbild
</fo:block>
```

❶ Das Bild wird als Hintergrundbild für den Block verwendet. Das Attribut `background-repeat` steuert die Wiederholung der Grafik.

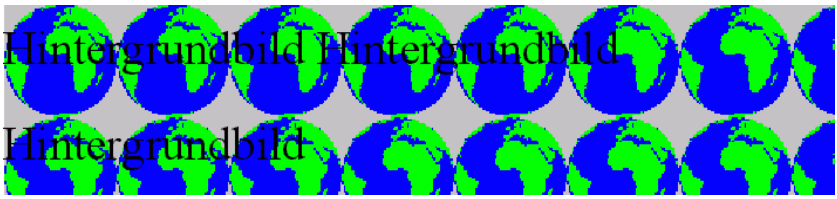


Abb. 6-30
Formatierer-Ansicht

6.10 Tabellen

In XSL-FO wird eine eigene Tabellenstruktur unterstützt, die die typografischen Anforderungen eines Satz-und-Umbruchsystems erfüllt. Diese Tabellenstruktur ist weder identisch mit der der traditionellen CALS-Tabellenstruktur, noch mit der der HTML-Tabellenstruktur. Sie ist allerdings ausgezeichnet geeignet, beide Tabellenmodelle zum Ausgangspunkt für die typografische Umsetzung zu nehmen. Besonders einfach ist das XSL-FO-Tabellenkonzept für den Tabellensatz von so genannten semantischen Tabellendaten zu nutzen, wie sie beispielsweise aus Datenbank-Anwendungen zum Satz exportiert werden.

Vgl. Abschnitt 10.13

Für den Satz von Tabellen stehen neun Elemente und eine Vielzahl von Attributen zur Verfügung. Die folgende Grafik zeigt die Struktur, die sich hieraus ergibt.

Vgl. Abschnitt 10.14

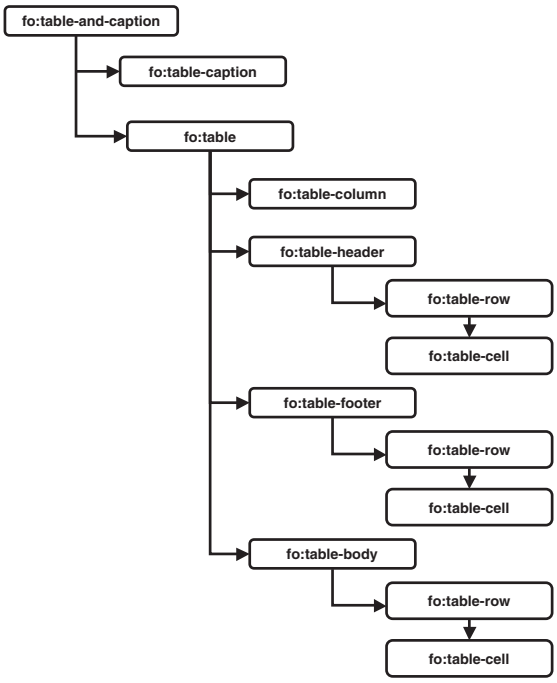


Abb. 6-31
Tabellen-Element-struktur

Die einzelnen Elemente:

- `<fo:table>` ist das zentrale (oberste) und obligatorische Element zur Erzeugung einer Tabelle. Es besitzt eine optionale Spaltendefinition (`<fo:table-column>`), eine oder mehrere optionale Kopf- und Fuß-Tabellenzeilen (`<fo:table-header>` und `<fo:table-footer>`) und den obligatorischen Tabellen-Hauptteil (`<fo:table-body>`).
- `<fo:table-header>` erzeugt den Kopfbereich der Tabelle, der bei einem eventuellen Seitenumbruch automatisch wiederholt wird.
- `<fo:table-body>` enthält den tabellarischen Inhalt der Tabelle, der aus Zeilen (`<fo:table-row>`), diese wiederum aus Zellen (`<fo:table-cell>`) besteht.
- `<fo:table-footer>` erzeugt den Fußbereich der Tabelle, der bei einem eventuellen Seitenumbruch automatisch wiederholt wird.
- `<fo:table-column>` erlaubt die Vorweg-Spezifikation verschiedener Eigenschaften für die Spalten der Tabelle.
- `<fo:table-and-caption>`: Hat die Tabelle eine Überschrift oder Legende, ist dieses Element das oberste und obligatorische Tabellen-Element.
- `<fo:table-caption>` nimmt die Überschrift oder Legende einer Tabelle auf, die als oberstes Element `<fo:table-and-caption>` hat.

- `<fo:table-row>` legt Inhalt und die Eigenschaften der Tabellenzeilen fest.
- `<fo:table-cell>` legt Inhalt und die Eigenschaften der Tabellenzellen fest.

Die Attribute der Tabellenelemente

Da die Elemente mit einer Vielzahl von Attributen typografisch behandelt werden können, werden hier nur die Tabellen spezifischen Attribute näher erläutert und die aus anderen Zusammenhängen bekannten verkürzt dargestellt. In der Praxis irrelevante Attribute werden nicht behandelt.

<fo:table> (Tabellen-Attribute)

- `background`-Attribute, `border`-Attribute, `padding`-Attribute, `margin`-Attribute, `space`-Attribute, `id`, `break-after`, `break-before`, `keep-with-next`, `keep-with-previous`, `keep-together`
- `height` (Höhe), `width` (Breite)
- `border-collapse` definiert das zu verwendende Modell für die benachbarten Zellen-Rahmen. Beim Wert `collapse` fallen die benachbarten Rahmen zusammen, bilden also eine gemeinsame Zellentrennung, bei `separate` bildet jede Zelle ihren eigenen Rahmen. Für die benachbarten Rahmen lässt sich ein Abstand definieren. Diese Flächen außerhalb der Rahmen werden ggf. mit dem für die Tabelle definierten Hintergrund gefüllt.
- `border-separation` definiert den Abstand zwischen zwei benachbarten Rahmen, der ggf. mit dem für die Tabelle definierten Hintergrund gefüllt wird.
- `table-layout` legt einen Algorithmus für die Gestaltung der Tabellenzellen, -zeilen und -spalten fest. Der Vorgabewert ist `auto`. Bei diesem Wert wird – HTML-Tabellen entsprechend – die Breite einer Spalte von den Zelleninhalten in dieser Spalte abhängig gemacht. Bei dem Wert `fixed` wird das Tabellen-Layout allein durch die Breitenangaben für die Tabelle und der darin enthaltenen Spalten, Rahmen und Abständen zwischen den Rahmen bestimmt.
- `table-omit-footer-at-break` legt fest, ob der Fußbereich der Tabelle (`<fo:table-footer>`) bei Umbruch über die gegebene Seite hinaus auf den Folgeseiten wiederholt werden soll. Der Vorgabewert ist `false`, d. h. der Fußbereich wird wiederholt. Nur bei Spezifikation des Wertes `true` wird der Fußbereich nicht wiederholt.

- `table-omit-header-at-break` ist das Wiederhol-Attribut für den Kopfbereich `<fo:table-header>`. Es gilt das für den Fußbereich Gesagte entsprechend.

<fo:table-column> (Spalten-Definitions-Attribute)

- `background`-Attribute, `border`-Attribute, `padding`-Attribute
- `column-number` (Nummer der Spalte). Zulässiger Wert ist eine Zahl, z. B. für die 1. Spalte `1`. Diese Zahl dient der Zuordnung der Spalten-Spezifikationen zu Zellen, die den gleichen Wert für `column-number` haben.
- `column-width` (Breite der Spalte). Zulässige Werte sind beliebige, unterstützte absolute Maße oder Prozentzahlen (Tabellenbreite ist `100%`).
- `number-columns-repeated` (Anzahl der Spalten, für die diese Spezifikation gelten soll). Zulässige Werte sind ganze Zahlen, z. B. `2`.
- `number-columns-spanned` (Anzahl der Spalten, die überspannt werden sollen). Zulässige Werte sind ganze Zahlen, z. B. `2`.

<fo:table-body>, <fo:table-header>, <fo:table-footer> (Attribute für die Tabellenbereiche)

- `background`-Attribute, `border`-Attribute, `padding`-Attribute, `id` (Zuweisung einer eindeutigen Identifikation)

<fo:table-row> (Tabellenzeilen-Attribute)

- `background`-Attribute, `border`-Attribute, `padding`-Attribute, `id`, `break-after`, `break-before`, `keep-with-next`, `keep-with-previous`, `keep-together`
- `height` (Zeilenhöhe)

<fo:table-cell> (Tabellenzellen-Attribute)

- `background`-Attribute, `border`-Attribute, `padding`-Attribute, `id`.
- `height` (Zellenhöhe) und `width` (Zellenbreite)
- `display-align` (vertikale Ausrichtung des Zelleninhalts). Der Vorgabewert ist `auto` und bewirkt, dass der Zelleninhalt behandelt wird, als sei der Wert `before` definiert. Bei üblicher Schreibrichtung bewirkt der Wert `before` die Aliniierung am oberen Zellenrand, `center` eine mittige Ausrichtung und `after` die Aliniierung am unteren Zellenrand.

- `empty-cells` (Rahmen um leere Zellen). Der Vorgabewert ist `show`, d. h. leere Zellen haben ihren Rahmen wie die Zellen mit Inhalten. Alternativer Wert ist `hide`. Bei diesem Wert werden die Rahmen unterdrückt. Haben alle Zellen einer Tabellenzeile den Attributwert `hide`, dann wird die gesamte Tabellenzeile in der Darstellung unterdrückt.
- `ends-row` (Ende einer Tabellenzeile). Der Vorgabewert ist `false`, d. h. die gegebene Zelle beendet keine Tabellenzeile. Mit dem alternativen Wert `true` beendet die gegebene Zelle die Tabellenzeile. Dies gilt nur dann, wenn die Zellen nicht innerhalb von `<fo:table-row>` stehen.
- `starts-row` (Beginn einer neuen Tabellenzeile). Das für `ends-row` Gesagte gilt hier entsprechend.
- `column-number` (Nummer der Spalte). Hiermit wird eine Verbindung zu den Spezifikationen der entsprechenden `<fo:table-column>` hergestellt.
- `number-columns-spanned` (Zahl der zu überspannenden Spalten). Vorgabewert ist 1, d. h. die Zelle füllt gerade eine Spalte. Entsprechend dem ganzzahligen Attributwert werden Spalten in der Schreibrichtung überspannt. Beispielsweise überspannt die Zelle drei Spalten bei einem Attributwert von 3.
- `number-rows-spanned` (Zahl der zu überspannenden Tabellenzeilen). Das zu `number-columns-spanned` Gesagte gilt entsprechend.

Zu beachten: Die horizontale Ausrichtung (Aliniierung) des Zelleninhalts wird nicht als Attribut der Zelle selbst spezifiziert, sondern als Attribut der in der Zelle befindlichen Blöcke!

Für den Einstieg ein einfaches Tabellenbeispiel:

```
<fo:block>
<fo:table>
   <fo:table-body>
      <fo:table-row>
         <fo:table-cell>
            <fo:block> Erste Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell>
            <fo:block> Zweite Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell>
            <fo:block> Dritte Zelle</fo:block>
         </fo:table-cell>
      </fo:table-row>
   </fo:table-body>
</fo:table>
</fo:block>
```

Diese Tabelle stellt die Grundform einer Tabelle dar. Allerdings fehlen einige charakteristische Eigenschaften. Es sind keine Größenangaben für die Tabelle, deren Zeilen oder Zellen angegeben, und die Tabellenzellen sind ohne Rahmen. Das nächste Beispiel soll etwas komplexer sein und diese Eigenschaften enthalten.

```
<fo:block>
<fo:table width="110mm" border-style="ridge" border-width="5pt">          ❶
   <fo:table-body>                                                        ❷
      <fo:table-row>                                                      ❸
         <fo:table-cell width="40mm" border-style="solid" border-         ❹
             width="1pt">
            <fo:block> 1. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell width="40mm" border-style="solid" border-         ❹
             width="1pt">
            <fo:block> 2. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell width="30mm" border-style="solid" border-         ❹
             width="1pt">
            <fo:block> 3. Zelle</fo:block>
         </fo:table-cell>
      </fo:table-row>
      <fo:table-row>                                                      ❸
         <fo:table-cell border-style="solid" border-width="1pt">          ❺
            <fo:block> 1. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell border-style="solid" border-width="1pt">          ❺
            <fo:block> 2. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell border-style="solid" border-width="1pt">          ❺
            <fo:block> 3. Zelle</fo:block>
         </fo:table-cell>
      </fo:table-row>
   </fo:table-body>
</fo:table>
</fo:block>
```

❶ Das Element `<fo:table>` beinhaltet den Inhalt der Tabelle. Das Attribut `width` legt die Breite der Tabelle fest. Das Attribut `border-style` definiert den äußeren Rahmen der Tabelle. Um dies zu veranschaulichen, unterscheidet sich dieser von den Rahmen der Zellen.

❷ `<fo:table-body>` ist der Behälter für den Inhalt der Tabelle, die hier keinen Fuß- oder Kopfbereich hat.

❸ `<fo:table-row>` definiert eine Zeile einer Tabelle. Auch hier sind diverse Attribute erlaubt, auf die aus Vereinfachungsgründen vorerst verzichtet wurde.

❹ `<fo:table-cell>` definiert eine Zelle. Die ersten beiden Zellen haben hier eine Breite von `40mm`, die dritte von `30mm`. Die Zellen besitzen einen kompletten Rahmen (`border-style="solid" border-width="1pt"`).
❺ Die Zellen der zweiten Spalte haben im Unterschied zu denen der ersten keine Breitenangaben. Daher werden die Breitenangaben der ersten Zeile automatisch übernommen.

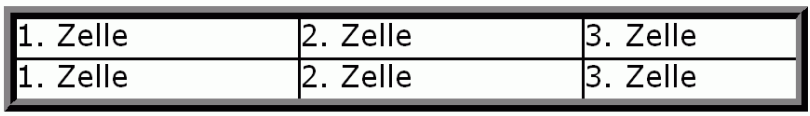


Abb. 6-32
Formatierer-Ansicht

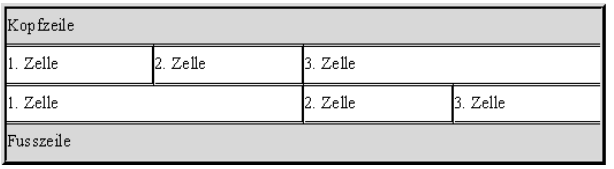
Das folgende Beispiel ist ein wesentlich komplexeres, das die gebräuchlichsten Tabelleneigenschaften nutzt:

```
<fo:block>
<fo:table width="110mm" border-style="outset" border-width="1pt">          ❶
   <fo:table-column column-number="1" column-width="25%" border-           ❷
       style="solid" border-width="1pt"/>
   <fo:table-column column-number="2" column-width="25%" border-
       style="solid" border-width="1pt"/>
   <fo:table-column column-number="3" column-width="25%" border-
       style="solid" border-width="1pt"/>
   <fo:table-column column-number="4" column-width="25%" border-
       style="solid" border-width="1pt"/>
   <fo:table-header background-color="red">                               ❸
      <fo:table-row>
         <fo:table-cell number-columns-spanned="4">                       ❹
            <fo:block> Kopfzeile</fo:block>
         </fo:table-cell>
      </fo:table-row>
   </fo:table-header>
   <fo:table-footer background-color="red">                               ❸
      <fo:table-row>
         <fo:table-cell number-columns-spanned="4">                       ❹
            <fo:block> Fusszeile</fo:block>
         </fo:table-cell>
      </fo:table-row>
   </fo:table-footer>
   <fo:table-body>
      <fo:table-row border-style="double" border-width="1pt">
         <fo:table-cell column-number="1">                                ❺
            <fo:block> 1. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell column-number="2">                                ❺
            <fo:block> 2. Zelle</fo:block>
         </fo:table-cell>
         <fo:table-cell column-number="3" number-columns-                 ❺
             spanned="2">
            <fo:block> 3. Zelle</fo:block>
```

```
            </fo:table-cell>
         </fo:table-row>
         <fo:table-row border-style="double" border-width="1pt">
            <fo:table-cell column-number="1" number-columns-                    ❺
                spanned="2">
               <fo:block> 1. Zelle</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="3">                                   ❺
               <fo:block> 2. Zelle</fo:block>
            </fo:table-cell>
            <fo:table-cell column-number="4">                                   ❺
               <fo:block> 3. Zelle</fo:block>
            </fo:table-cell>
         </fo:table-row>
      </fo:table-body>
   </fo:table>
   </fo:block>
```

❶ Das Attribut `width` legt die Gesamtbreite der Tabelle fest.
❷ Die Definition der Spalten erfolgt mit dem Element `<fo:table-column>`. Jeder Spalte wird eine Nummer mit dem Attribut `column-number` und eine Breite mit dem Attribut `column-width` zugewiesen. Die Breite der Spalte wird hier in Prozent der Gesamtbreite der Tabelle (`110mm`) angegeben.
❸ Die Tabelle verfügt über einen farblich abgehobenen Kopfteil, der mit dem Element `<fo:table-header>`, und einem gleichfarbigen Fußteil, der mit dem Element `<fo:table-footer>` spezifiziert wird. Das Attribut `background-color` erwartet als Wert einen vordefinierten Farbwert wie hier `red` oder aber einen RGB-Dreifarbenwert.
❹ Im Fuß- und Kopfbereich enthält die Tabellenzelle das Attribut `number-columns-spanned` mit dem Wert `"4"`. Dies bewirkt eine Überspannung dieser Zelle über alle vier Spalten.
❺ Die Tabellenzellen im Inhaltsteil (`<fo:table-body>`) werden mit Hilfe des Attributs `column-number` den vordefinierten Spalten zugewiesen.

Abb. 6-33
Formatierer-Ansicht



6.11 Das Float-Konzept

Das Float-Konzept wird in der Praxis für zwei Gestaltungsziele eingesetzt, die auf den ersten Blick nicht viel miteinander zu tun zu haben scheinen. Das eine Ziel ist die Erzeugung von Texten, die andere Objekte (meist Bilder) umfließen, das andere die Platzierung von Marginalien,

d. h. Texten, die aus dem Textfluss (am Rand) herausgestellt sind. Für letztere Nutzung ist das Float-Konzept nur bedingt geeignet, allerdings notgedrungen dafür zurechtgebogen, weil es in XSL-FO kein eigenständiges Marginalien-Konzept gibt.

Das fließende Element wird in den Block eingefügt, in dem es fließen soll. Dafür wird es in ein `<fo:float>`-Element eingekleidet. Mit dem gleichnamigen Attribut `float` wird spezifiziert, wohin das Fließ-Objekt im umgebenden Block fließen soll. Der Attributwert:

- `start` setzt das Fließ-Objekt an den Start-Rand, bei der bei uns üblichen Schreibrichtung links.
- `end` setzt das Fließ-Objekt an den End-Rand, bei der bei uns üblichen Schreibrichtung rechts.
- `left` setzt das Fließ-Objekt an den linken Rand.
- `right` setzt das Fließ-Objekt an den rechten Rand.

Für den Einsatz des Antenna House Formatters hat das Attribut noch zwei weitere, proprietäre Werte:

- `inside` setzt das Fließ-Objekt an den inneren Rand (auf der linken Seite rechts, auf der rechten Seite links).
- `outside` setzt das Fließ-Objekt an den äußeren Rand (auf der linken Seite links, auf der rechten Seite rechts).

Im umgebenden Block muss entsprechender Raum für das Fließ-Objekt freigeschlagen werden. Dies wird als Attribut des umgebenden Block-Elements mit dem Attribut `intrusion-displace` gesteuert.

Das Attribut kann folgende Werte enthalten:

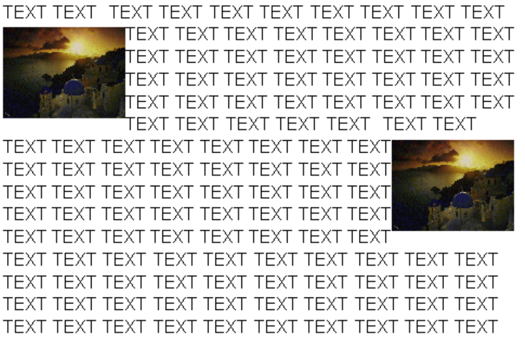
- `line` umfließt den Inhalt des `<fo:float>`-Elements unter Berücksichtigung von `start-indent`.
- `indent` umfließt den Inhalt des `<fo:float>`-Elements unter der Berücksichtigung von `text-indent` und `start-indent`.
- `block` erzeugt einen Block neben dem Inhalt des `<fo:float>`-Elements unter der Berücksichtigung von `text-indent`. Das heißt, der umgebende Block wird ab dem oberen Rand des Fließ-Objekts eingezogen.

Ein Beispiel für umfließenden Text (`line`):

```
<fo:block intrusion-displace="line" >
   TEXT TEXT TEXT TEXT ... TEXT TEXT TEXT
   <fo:float float="left">
      <fo:block>
         <fo:external-graphic src="hellas.bmp" content-width="30mm"/>
      </fo:block>
   </fo:float>
```

```
   TEXT TEXT TEXT ... TEXT TEXT TEXT
   <fo:float float="right">
      <fo:block >
         <fo:external-graphic src="hellas.bmp" content-width="30mm"/>
      </fo:block>
   </fo:float>
   TEXT TEXT TEXT ... TEXT TEXT TEXT
</fo:block>
```

Abb. 6-34 Formatierer-Ansicht



Ein Beispiel für einen eingezogenen Block:

```
<fo:block intrusion-displace="block">
   <fo:float float="right">
      <fo:block margin-left="3mm">
         MARGINALIE
      </fo:block>
   </fo:float>
   TEXT TEXT TEXT ... TEXT TEXT TEXT
</fo:block>
```

Abb. 6-35 Formatierer-Ansicht

```
TEXT TEXT TEXT TEXT TEXT TEXT    MARGINALIE
TEXT TEXT TEXT TEXT TEXT TEXT
TEXT TEXTTEXT TEXT TEXT TEXT
TEXT TEXT TEXTTEXT TEXT TEXT
TEXT TEXT TEXT TEXT TEXT TEXT
TEXT TEXT TEXT TEXT TEXTTEXT
TEXT TEXT TEXT TEXT TEXT TEXT
```

6.12 Querverweise/Hyperlinks

Vgl. Abschnitt 10.18

In XSL-FO besteht die Möglichkeit, Querverweise/Hyperlinks zu erzeugen. Diese können entweder auf ein Ziel innerhalb des Dokuments verweisen oder aber auf externe Quellen.

Die Nutzung solcher Querverweise/Hyperlinks unterscheidet sich, je nachdem welches Medium zur Ausgabe eingesetzt wird: PDF in einem interaktiven Reader wird die Hyperlink-Funktion nutzen, um beispielsweise vom Link-Ursprung durch Klicken zum Link-Ziel zu navigieren. Im

Druck wird man die Querverweis-Funktion nutzen, um beispielsweise an der Stelle des Verweisursprungs die Bezeichnung/den Titel des Verweisziels erscheinen zu lassen. Basis beider Funktionen ist das Element `<fo:basic-link>`.

Zunächst muss das Verweisziel bezeichnet werden. Dies geschieht dadurch, dass dem Block, der das Verweisziel beinhaltet (ein Kapitel, eine Grafik, eine Tabelle, eine Überschrift u. a.), das Attribut `id` beigefügt wird und mit einem eindeutigen Identifikator versehen wird. Das Attribut `id` kann Blöcken jeder Art und sogar inzeiligen Elementen zugeordnet werden.

An der Stelle des Verweisursprungs wird das Element `<fo:basic-link>` gesetzt.

Das Element `<fo:basic-link>` kann eine Fülle von Attributen enthalten. Für die Demonstration dieses Konzepts sind allerdings nur drei Attribute von Bedeutung:

- Im Fall eines Dokument-internen Verweisziels wird das Attribut `internal-destination` gesetzt. Dessen Wert verweist auf das Verweisziel mittels eines XPath-Ausdrucks.
- Im Fall eines Dokument-externen Verweisziels wird das Attribut `external-destination` gesetzt. Dessen Wert verweist auf eine Stelle außerhalb des Dokuments. Dies kann eine Web-Adresse oder ein anderes Dokument sein.
- Zu spezifizieren ist außerdem das Verhalten des Links/Verweises: Dies geschieht mit dem Attribut `show-destination` (Anzeige des Verweisziels). Der Vorgabewert ist `replace`, d. h. das Verweisziel wird in der gegebenen Sicht (das ist im Falle des Drucks immer die Druckseite) angezeigt. Der alternative Wert `new` öffnet das Verweisziel in einer neuen Sicht, hat also für die Druckausgabe keine Bedeutung.

Zu beachten ist, dass die Hyperlink-Funktionalität in der PDF-Ausgabe abhängig davon ist, ob der die Formatierung übernehmende PDF-Generator auf Grund der XSL-FO-Codierung automatisch übernimmt oder zusätzliche Eingriffe in das Formatier-Ergebnis verlangt.

Das folgende Beispiel hat drei Komponenten, weil in der Verarbeitung die Auflösung des Verweises/Links durch den XSLT-Prozess erfolgt, nicht also erst im Formatier-Prozess geschieht: Das zu formatierende XML-Dokument, das XSL-Stylesheet und das resultierende XSL-FO-Dokument.

Ausschnitt aus dem zu formatierenden XML-Dokument:

```
<titel id="links">Querverweise/Hyperlinks</titel>                    ❶
...
text ... <ref idref="links"/> ... text ...                           ❷
```

❶ Dies ist das Verweisziel im XML-Dokument. Der Titel hat für das Attribut `id` den Wert `"links"`.
❷ Dies ist der Verweisursprung. Das Attribut `idref` verweist durch seinen Wert `"links"` auf den Titel mit dem gleichen Wert im Attribut `id`.

Ausschnitt aus dem XSL-Stylesheet:

```
<xsl:template match="titel">                                          ❶
   <fo:block id="{generate-id()}" ...>                                ❷
      <xsl:apply-templates/>
   </fo:block>
</xsl:template>
...
<xsl:template match="ref">                                            ❸
   <xsl:for-each select="//titel[contains(@id,current()/@idref)]">    ❹
      <fo:inline text-decoration="underline">                         ❺
         <fo:basic-link internal-destination="{generate-id()}"        ❻
             show-destination="replace">
            <xsl:value-of select="."/>                                ❼
         </fo:basic-link>
      </fo:inline>
   </xsl:for-each>
</xsl:template>
```

❶ Dies ist das Template für das Element `<titel>`.
❷ Dem Block, der den Inhalt des Titels aufnehmen soll, wird ein Attribut `id` beigefügt, dessen Wert durch den XSLT-Prozessor generiert werden soll und mit der Funktion `generate-id()` spezifiziert ist.
❸ Dies ist das Template für das Element `<ref>`.
❹ Das `<xsl:for-each>`-Element bestimmt für jedes Auftreten gleicher `id`-`idref`-Attributwerte für Verweisziel `<titel>` und Verweisursprung `<ref>` das in den folgenden Zeilen Spezifizierte. Der Attributwert von `select` ist ein XPath-Ausdruck, der das Verweisziel so beschreibt: Ein Element `<titel>` in jeder Umgebung, der für das Attribut `id` den gleichen Wert (`"links"`) hat wie das Attribut `idref` des hier gegebenen Elements `<ref>`.
❺ Das Attribut `text-decoration` wird lediglich gesetzt, um den Link-Ursprung im PDF als solchen kenntlich zu machen.

Vgl. Abschnitt 5.1

❻ Der Attributwert von `internal-destination` soll ebenfalls mit der Funktion `generate-id()` im XSLT-Prozess generiert werden (s. die Parallele für `<titel>`). Das Attribut `show-destination` mit seinem Wert `replace` ist lediglich der Klarheit wegen definiert.
❼ Durch diesen XSLT-Ausdruck wird der Inhalt des `<titel>`-Elements in das `<fo:basic-link>`-Element eingefügt. Würde man nicht den Inhalt des Elements `<titel>`, sondern die Seitenzahl, auf der der Titel erscheint, einfügen wollen, hieße der Ausdruck in dieser Zeile `<fo:page-number-citation ref-id="{generate-id()}"/>`.

Ausschnitt aus dem resultierenden XSL-FO-Dokument:

```
...
<fo:block id="IDAHGBZB" ...>Querverweise / Hyperlinks</fo:block>          ❶
...
...<fo:inline text-decoration="underline">
fo:basic-link internal-destination="IDAHGBZB" show-                       ❷
    destination="replace">
Querverweise / Hyperlinks                                                 ❸
</fo:basic-link>
</fo:inline>...
```

❶ Der Block mit dem Inhalt des Titels hat jetzt für `id` den durch den XSLT-Prozess generierten Wert `"IDAHGBZB"`.
❷ Das inzeilige Element `<fo:basic-link>` hat für das Attribut `internal-destination` den gleichen Wert `"IDAHGBZB"`. Dadurch wird es später dem PDF-Generator ermöglicht, den Hyperlink zwischen den Seiten zu generieren, der Verweisursprung und Verweisziel miteinander verbindet.
❸ Hier wird der Inhalt des Titels eingefügt, auf den verwiesen ist. Stünde im Stylesheet an Stelle von `<xsl:value-of>` der Ausdruck `<fo:page-number-citation>`, würde an dieser Stelle durch den Formatierer die Seitenzahl eingetragen, auf der der betreffende Titel steht.