

Kapitel 8

Formatierung mit XSL

Wesentlich mächtiger als CSS sind die Formatierungsmöglichkeiten, die über die Formatierungsobjekte von XSL realisiert werden können.

Während sich XML als Standard für die Strukturierung von Inhalten in einem kompakten Dokument beschreiben lässt, bedarf der X-Standard, der sich mit der Gestaltung von Dokumenten befasst, einer wesentlich umfangreicherer Erläuterung. In dem einen Fall geht es um klare logische Strukturen, im anderen Fall darum, alle Möglichkeiten in den Griff zu bekommen, und um die Frage, wie Textelemente und Bilder auf gedruckten oder am Bildschirm angezeigten Seiten unter Berücksichtigung von Seitenumschriften, Mehrspaltigkeit, variablen Größen und den unterschiedlichen Ausrichtungen etc. dargestellt werden können.

Schon für die CSS-Standards waren umfangreiche Dokumente notwendig, XSL bringt es in der PDF-Darstellung immerhin auf über 400 Seiten. Die Entwickler von kompletten XSL-Prozessoren hatten also einiges zu tun.

8.1 Transformation und Formatierung

XSL ist für eine Verarbeitung konzipiert, die in mehreren Stufen abläuft. Zwei Prozesse werden dabei verkoppelt, wobei der zweite Prozess die Ergebnisse des ersten übernimmt. Abbildung 8.1 zeigt den generellen Ablauf.

Im ersten Schritt übernimmt ein entsprechender XSL-Stylesheet-Prozessor die Baumstruktur des Quelldokuments und wendet darauf die in dem XSL-Stylesheet beschriebenen Transformationen an, um die daraus resultierende Baumstruktur aufzubauen. Dieser Schritt entspricht demjenigen, der im vorangegangenen Kapitel für XSLT beschrieben worden ist. Der Ergebnisbaum kann, wie erläutert, die Quellinhalte in einer anderen Anordnung oder Auswahl liefern, das heißt, das Ergebnis kann sich deutlich von dem zugrunde liegenden Quelldokument unterscheiden. Aus den Quelldaten könnte zum Beispiel zusätzlich ein Inhaltsverzeichnis oder ein Index generiert werden.

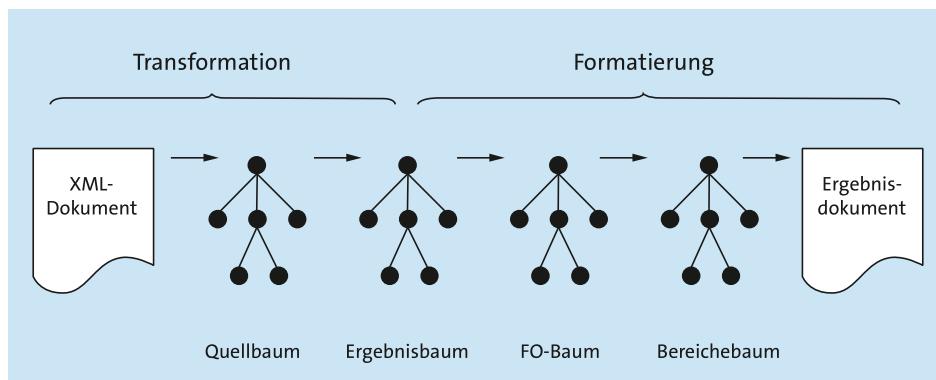


Abbildung 8.1 Der mehrstufige XSL-Prozess

Die gewünschte Formatierung der Datenausgabe wird nun im nächsten Schritt dadurch erreicht, dass in den Ergebnisbaum die dafür notwendigen Formatbeschreibungen eingefügt werden, die im Stylesheet festgelegt sind. Dabei wird ein Katalog von Formatierungsobjekten verwendet, der alles abdecken soll, was in puncto Gestaltung in der Ausgabe möglich ist. Der Ergebnisbaum besteht schließlich wie der Quellbaum aus Element- und Attributknoten, nur bilden diese jetzt entsprechende Formatierungsobjekte und deren Eigenschaften ab. Dieser Ergebnisbaum wird schließlich von einem *XSL Formatter* in dem Zielmedium ausgegeben.

8.2 Formatierungsobjekte

Die Formatierungsobjekte bilden also in dem XSL zugrunde liegenden Modell ebenfalls eine Baumstruktur. Dabei ist beispielsweise eine Seite oder Webseite die Wurzel, die etwa zu bestimmten Textabschnitten oder Tabellen in Form von Ästen verzweigt. Die Blätter schließlich sind die einzelnen Zeichen oder Bilder oder andere unteilbare Objekte.

Die Formatierungsobjekte sind jeweils Instanzen von abstrakten Klassen, die typografische Dinge wie Seite, Absatz, Zeile, Tabelle etc. repräsentieren. Jedes dieser Objekte kann über einen Satz von Eigenschaften im Detail gestaltet werden. Die Formatierungsobjekte werden, wie in einem XML-Dokument üblich, als Elemente dargestellt, die zugehörigen Eigenschaften erscheinen als Attribute dieser Elemente. Der Elementinhalt, der aus den Elementen des Quelldokuments übernommen wird, erscheint hier als Inhalt des Formatierungsobjekts. Alle für die Formatierungsobjekte und die Formatierungseigenschaften vorgegebenen Namen gehören zu dem Namensraum mit dem URI <http://www.w3.org/1999/XSL/Format>, dem konventionell das Präfix `fo` zugeordnet ist.

Der Prozess der Formatierung selbst lässt sich wiederum in drei Schritte gliedern. Im ersten Schritt werden die von der Transformation gelieferten Objekte in Formatierungsobjekte umgewandelt, und der Baum der Formatierungsobjekte wird erzeugt. Dabei werden die einzelnen Zeichen, die den Inhalt der Quelldaten liefern, in `fo:-Zeichenknoten` umgewandelt. Die Knoten, die nur aus Leerzeichen bestehen, werden ignoriert.

In einer zweiten Phase wird dieser noch grobe Formatierungsobjektbaum verfeinert. Dazu gehören die Entfernung überflüssiger Leerzeichen, die Interpretation von unklaren Eigenschaften sowie die Entfernung von Duplikaten.

8.3 Baum aus Bereichen – Areas

Auf dieser Basis wird dann in einem dritten Schritt ein *Area Tree* aufgebaut, also ein Baum von geometrischen Bereichen, aus denen die gesamte Ausgabe zusammenge stellt wird. Dabei müssen die den Formatierungsobjekten zugeordneten Eigenschaften noch in die endgültigen Merkmale umgewandelt werden, die der Formatter darstellen kann.

Diese Merkmale werden *Traits* genannt. In vielen Fällen entsprechen sie den angegebenen Eigenschaften, in anderen Fällen aber müssen sie erst errechnet werden, zum Beispiel wenn Schriftgrößen relativ zu anderen definiert werden. Manchmal ergeben sie sich auch in Folge von Vererbung aus dem übergeordneten Element.

Ähnlich wie CSS formatiert XSL also den Inhalt eines XML-Dokuments mit Hilfe von rechteckigen Bereichen. Jeder Bereich hat eine bestimmte Position auf einer Seite, und zu jedem Bereich ist festgelegt, was darin angezeigt werden soll. Die Bereiche sind die Container, die den Inhalt aus der XML-Quelle aufnehmen.

Diese Bereiche sind nicht selbst die Formatierungsobjekte; die Formatierungsobjekte erzeugen diese Bereiche vielmehr, indem sie Platz im Ausgabemedium belegen, so wie ein Absatz Platz auf einer Seite belegt.

Entscheidend dabei ist, dass die Menge der Bereiche als hierarchische Anordnung verfügbar ist, so dass die einzelnen Bereiche mit ähnlichen Verfahren angesteuert und abgearbeitet werden können, wie es auch mit den Knoten im DOM oder bei XPath möglich ist.

8.4 XSL-Bereichsmodell

Ein Bereich enthält »im Kern« ein Inhaltsrechteck. Das ist der Raum, in dem entweder direkt Text erscheint oder Kindbereiche auftreten. Umgeben ist der innere Kern möglicherweise von einem Rechteck, das im Englischen *Padding* genannt wird, die

»Aufpolsterung« sozusagen, die dafür sorgen soll, dass ein Bereich zu einem folgenden einen sinnvollen Abstand hat. Ganz außen kann sich das Umrundungsrechteck *Border* befinden. Abbildung 8.2 zeigt das zugrunde liegende Bereichsmodell.

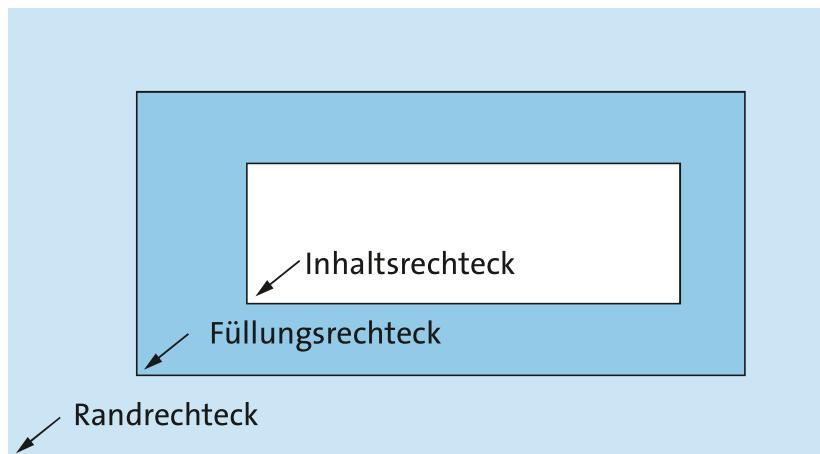


Abbildung 8.2 Bereichsmodell in XSL

Für jeden Bereich ist ein Satz von *Traits* verfügbar, die entweder direkt festlegen, wie der Bereich zu rendern ist – etwa durch die Wahl einer bestimmten Hintergrundfarbe –, oder die gewisse Einschränkungen festlegen, die beim Rendern zu berücksichtigen sind, etwa dass ein bestimmter Abstand zu einem anderen Bereich einzuhalten ist.

8.4.1 Block-Bereiche und Inline-Bereiche

Es werden generell zwei Typen von Bereichen unterschieden: *Block-Areas* und *Inline-Areas*. Sie unterscheiden sich hauptsächlich dadurch, wie sie durch das Formatierungsprogramm zueinander geordnet werden. Dabei sind auch absolut positionierte Bereiche möglich. *Inline-Areas* sind immer in einem *Block* eingeschlossen, sofern sie nicht Kinder einer *Inline-Area* sind. Diese Bereiche werden insbesondere verwendet, um einzelne Zeichen in einer Zeile gesondert zu formatieren. Ein Bereich kann jeweils *Block-* oder *Inline-Areas* als Kinder haben, aber nicht beide gleichzeitig.

8.4.2 XSL und CSS

Die Formatierungsobjekte in XSL haben eine Reihe von Gemeinsamkeiten mit den Optionen, die CSS für die Beschreibung der Formatierung anbietet, insbesondere mit CSS 2. Sie gehen aber teils darüber hinaus, teils verwenden sie eine andere Aufteilung der Objekte, um die es geht. Im Unterschied zu CSS berücksichtigt XSL insbesondere die internationale Gültigkeit des Standards sehr konsequent.

Deshalb wird versucht, auch die Formatierung von Dokumenten zu meistern, die mit einer anderen Schreibrichtung arbeiten, wie etwa arabische Schriften. Dazu ist insbesondere das Bereichsmodell erweitert worden. So werden Bereichszuordnungen nicht mehr mit absoluten Richtungsangaben wie top, bottom, right und left vorgenommen, sondern mit relativen wie before, after, start und end, die je nach Schreibrichtung zu anderen Ergebnissen führen.

Es ist im Rahmen dieses Buches nicht möglich, die komplexen Möglichkeiten von XSL im Detail vorzustellen. Wir wollen Ihnen aber an einem Beispiel wenigstens so weit das Vorgehen verdeutlichen, dass Sie einen soliden Eindruck davon gewinnen können, wie mit den Formatierungsobjekten gearbeitet wird.

8.5 Testumgebung für XSL

Obwohl die W3C-Empfehlung zu XSL inzwischen nicht mehr ganz jung ist, mangelt es doch immer noch an Anwendungen, die mit XSL-Stylesheets etwas anfangen können. Seit der Verabschiedung des Standards im Oktober 2001 sind immerhin verschiedene Prozessoren für XSL-FO entwickelt worden. Zu den bekanntesten gehören der *Antenna House Formatter* (siehe <https://www.antennahouse.com/antenna1/>), die *XEP Engine* von RenderX (siehe <http://www.renderx.com/tools>) und *FOP*, der *Formatting Objects Processor* der *Apache Software Foundation* (siehe <https://xmlgraphics.apache.org>). XML-Editoren von Altova oder Oxygen binden FOP ein, um XSL-FO zu rendern.

Im Dezember 2006 erschien die Version 1.1, die insbesondere die Unterstützung von Änderungsmarkierungen, Indizes, Lesezeichen und multipler Flows hinzufügte.

Wir wollen hier mit einer Kombination von XMLSpy als XSL-Editor und dem FOP aus dem Apache XML Project arbeiten, der über die Website von www.xmlspy.com/download_components.html kostenlos heruntergeladen werden kann. Allerdings muss dazu, da es sich um eine Java-Anwendung handelt, falls nicht schon vorhanden, das Java 2 Runtime Environment ebenfalls installiert werden.

Achten Sie darauf, dass XMLSpy den Pfad zu dem FOP-Prozessor kennt. Über EXTRAS • OPTIONEN finden Sie auf dem Register XSL im unteren Teil das entsprechende Eingabefeld.

FOP ist ein Formatter, der durch XSL-Formatierungsobjekte gesteuert wird und das durch den Baum der FOs gegebene Ergebnis in das gewählte Ausgabeformat rendert. Dabei werden verschiedene Ausgabeformate unterstützt: *PDF*, *PCL* (das Format für HP-Drucker), *PostScript*, *Text*, *XML* zur Wiedergabe des Bereichebaums und *MIF – Marker Interchange Format* – von Adobe FrameMaker. FOP hält sich an die Vorgaben der XSL-Empfehlung, allerdings werden noch nicht alle FOs von FOP unterstützt.

8 Formatierung mit XSL

Unter <https://xmlgraphics.apache.org/fop/compliance.html> finden Sie eine Tabelle mit detaillierten Informationen dazu.

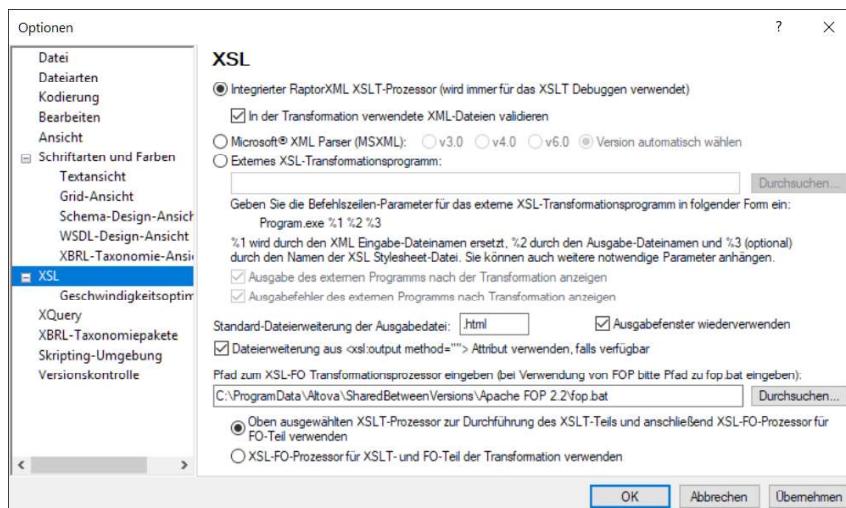


Abbildung 8.3 Auf diesem Register wird die Verknüpfung mit dem FOP-Prozessor festgelegt.

Damit den Entwicklern keine zu großen Gedächtnisleistungen abverlangt werden müssen, bieten XSL-Editoren wie XMLSpy in Fenstern für die Eingabehilfe alle FO-Objekte als Elemente an. Werden Elemente wie `<fo:root>` eingefügt, wird zugleich ein Gerippe der untergeordneten Elemente angeboten, die als Kinder von `<fo:root>` zu erwarten sind, wie Abbildung 8.4 zeigt.

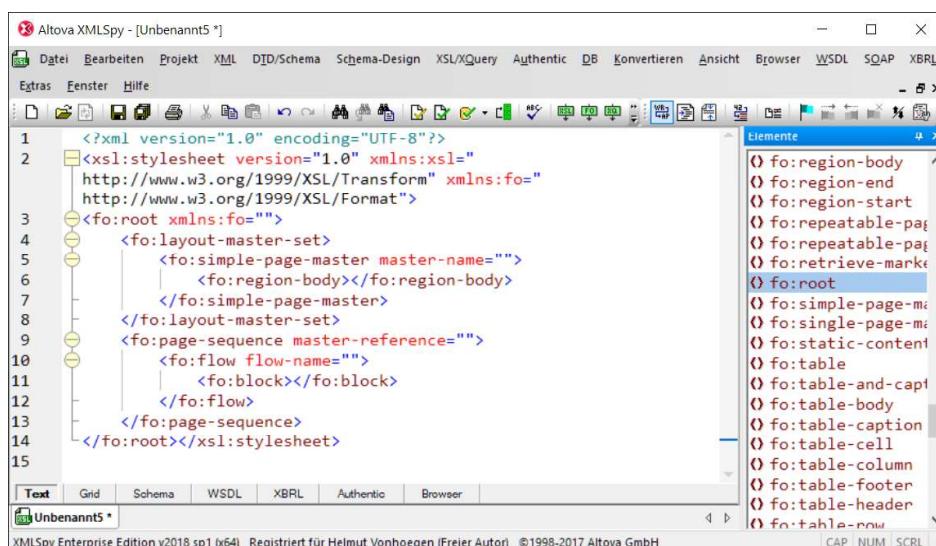


Abbildung 8.4 Elementliste im XSL-Editor in XMLSpy

Sie finden in der Empfehlung zu jedem FO eine vollständige Liste der möglichen Eigenschaften, gute Editoren bieten diese aber auch zur Auswahl an, wenn das Start-Tag des Elements eingetragen ist. Setzen Sie etwa in XMLSpy hinter einem Start-Tag ein Leerzeichen, werden die jeweils möglichen Attribute angeboten.

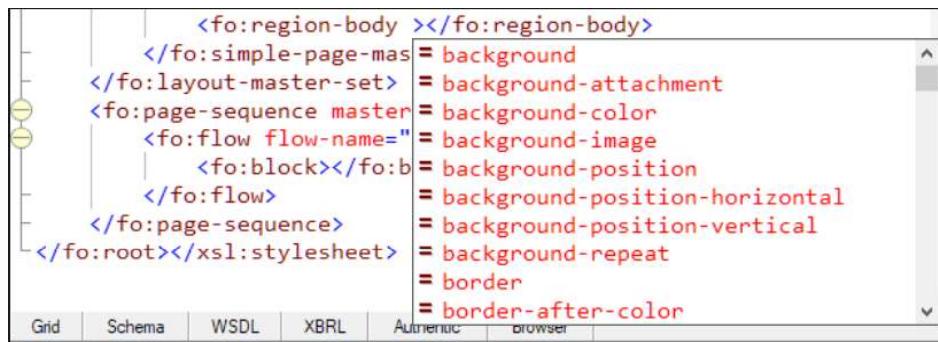


Abbildung 8.5 Attribute im XSL-Editor in XMLSpy

8.6 Aufbau eines XSL-Stylesheets

Als Beispiel soll noch einmal unsere Kursliste verwendet werden. Ein XSL-Stylesheet dazu beginnt zunächst mit den für ein XML-Dokument üblichen Deklarationen. Wenn Sie in XMLSpy eine neue XSL-Datei öffnen, wird automatisch ein `<xsl:stylesheet>`-Element als umgebendes Containerelement eingefügt und ebenso die Namensraumdeklarationen für XSLT und für XSL.

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

Obwohl das W3C die Empfehlung für XSL erst 2001 herausgebracht hat, wird im Namensraum für die Formatierungsobjekte die Jahreszahl 1999 beibehalten, weil der URI für diesen Namensraum bereits 1999 zugeordnet wurde.

Der schon angesprochenen Verknüpfung von Transformation und Formatierung entsprechend, folgt nun zunächst eine Transformationsregel, in der zugleich der Inhalt für die Formatierungsobjekte bereitgestellt wird. Dabei wird die Wurzel des Baums der Formatierungsobjekte gewissermaßen mit dem Dokumentelement des XML-Dokuments verknüpft.

```
<xsl:template match="kursprogramm">
  ...
</xsl:template>
```

8.6.1 Baum der Formatierungsobjekte

In diese Transformationsregel kann nun der Baum der Formatierungsobjekte eingefügt werden. Das erste Formatierungsobjekt, das innerhalb des Elements `<xsl:stylesheet>` zugelassen ist, heißt `<fo:root>`. Es ist die Wurzel des Baums der Formatierungsobjekte. Das erste Element, das von dieser Wurzel abstammen darf und muss, ist das Element `<fo:layout-master-set>`. Parallel zu diesem Element können noch ein Element `<fo:declaration>` (für die Angabe eines Farbprofils) und ein oder mehrere `<fo:page-sequence>`-Elemente eingefügt werden.

Das Element `<fo:layout-master-set>` repräsentiert einen Satz von Layoutvorlagen, die über den Aufbau und die Abfolge der Seiten entscheiden. Jede `page-sequence` steht für eine Folge von Seiten, die in einer bestimmten Weise formatiert wird, also zum Beispiel eine gemeinsame Kopf- oder Fußzeile hat.

Auch hier werden wieder verschiedene `master`-Elemente verwendet, um eine oder auch mehrere Vorlagen für Seiten, Seitenfolgen und für Regionen zu erzeugen. Für die einfache Seitengestaltung ist der `<fo:simple-page-master>` zuständig. Sollen spezielle Vorlagen für Seitenabfolgen zum Einsatz kommen, können entsprechende `page-sequence-master` eingesetzt werden.

8.6.2 Seitenaufbau

Wir wollen uns hier zunächst nur mit dem Element `<fo:simple-page-master>` beschäftigen, um nicht zu viel an Übersichtlichkeit zu verlieren. Damit von anderer Stelle auf das Element Bezug genommen werden kann, wird dem Attribut `master-name` ein passender Wert zugeordnet.

Über die übrigen Attribute, die dem Element beigegeben werden, steuern Sie den Aufbau und das Layout einer Seite. Dabei geht es hauptsächlich um das, was Sie in Anwendungsprogrammen üblicherweise in einem Dialog zur Seiteneinrichtung finden.

Innerhalb der Seite und damit als Kind des Elements `<fo:simple-page-master>` muss nun mindestens ein Bereich angelegt werden, der Inhalte aufnehmen kann. Für die Bildung von Bereichen auf einer Seite stehen insgesamt fünf Formatierungsobjekte zur Verfügung: `<fo:region-body>` ist der zentrale Bereich, `<fo:region-before>` steht für den Kopfbereich, `<fo:region-after>` für den Fußbereich, `<fo:region-start>` für den Bundbereich und `<fo:region-end>` für den Außenbereich.

Für jedes dieser fo:-Elemente stehen nun wiederum zahlreiche Eigenschaften zur Verfügung, mit deren Hilfe exakt festgelegt werden kann, wie die Bereiche gestaltet werden sollen. Dazu gehören die Festlegungen zur Position, zu den Randgrößen, den Abständen sowie den Hintergrundfarben, um nur einige zu nennen.

```
<xsl:template match="kursprogramm">
<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master master-name="Standardseite"
      page-height="400mm" page-width="300mm"
      margin-top="10mm" margin-left="15mm"
      margin-bottom="10mm" margin-right="15mm">

      <fo:region-body>
        <margin-top="0mm" margin-left="5mm"
          margin-bottom="0mm" margin-right="5mm">
        </fo:region-body>
        <fo:region-before extent="15mm"/>
        <fo:region-after extent="15mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    ...
  </fo:root>
</xsl:template>
```

Listing 8.1 Code für den Seitenaufbau

8.6.3 Seitenfolgen

Wenn mit Hilfe des Elements `<fo:simple-page-master>` der generelle Aufbau der Seite geklärt ist, können nun Seitenfolgen definiert werden, die jeweils den zuvor definierten Seitenaufbau oder auch eine spezielle Gestaltung der Seitenabfolge verwenden. Dazu wird das Element `<fo:page-sequence>` eingefügt. Erst wenn der Formatter dieses Element ausführt, werden die Seiten erzeugt. Zu den Attributen gehören die Angabe des Werts, mit dem die Seitennummerierung beginnen soll, sowie Festlegungen zum dafür zu verwendenden Format. Um die Seitengenerierung mit dem durch das `<fo:simple-page-master>`-Objekt bestimmten Seitenaufbau zu verknüpfen, wird ein Bezug auf den Namen dieses Elements hergestellt, und zwar über das Attribut `master-reference`.

8.6.4 Einfügen von Fließtext

Woher kommt nun aber überhaupt Inhalt auf die Seiten? Dafür werden hauptsächlich die `<fo:flow>`-Objekte benutzt, die Kinder des Elements `<fo:page-sequence>` sind. Hier geht es also um Fließtext, der auf die Seiten »ausgegossen« werden soll. Fixierte Texte, wie sie in den Kopf- und Fußzeilen erscheinen sollen, werden dagegen auf derselben Ebene, also als Kinder von `<fo:page-sequence>`, über `<fo:static-content>`-Objekte bereitgestellt. Die Elemente `<fo:flow>` und `<fo:static-content>` haben jeweils nur eine Eigenschaft: `flow-name`. Damit kann der Inhalt des Elements einer der innerhalb von `simple-page-master` eingefügten Regionen zugeordnet werden. Dabei werden reservierte Namen verwendet, etwa `flow-name="xsl-region-before"` oder `"xsl-region-body"`. Die Regionen müssen Sie natürlich anlegen, bevor Sie darauf Bezug nehmen.

8.6.5 Blockobjekte

Wie nun genau die verschiedenen Inhalte auf der Seite erscheinen sollen, geben Sie jeweils über `<fo:block>`-Objekte an, die rechteckige Anzeigebereiche im Dokument definieren. Ein Block kann wiederum Kindblöcke enthalten, *Inline-Areas* oder auch direkt eine Folge von Zeichendaten. Mit dem Element `<fo:block>` legen Sie die Formateigenschaften für die einzelnen Absätze in einem Text oder für eine Überschrift, eine Fußzeile oder sonst einen separaten Bereich fest. Die ganze Fülle der Möglichkeiten, die Sie etwa aus der Absatzformatierung in einem Textprogramm kennen, steht hier bereit, also alles, was die Schriftgestaltung, die Absatzbehandlung, die Ausrichtung, den Zeilenumbruch etc. betrifft.

In dem folgenden Beispiel wird in dem ersten block-Element direkt der Text angegeben, der in der Kopfzeile des Dokuments erscheinen soll. Der Textkörper der Seite dagegen soll die Daten aus dem XML-Dokument aufnehmen. Dazu wird zunächst ein flow-Objekt eingefügt, das eine `<xsl:for-each>`-Schleife enthält, die die einzelnen Kurse abarbeiten soll. Sie enthält hier nur die allgemeine Anweisung `<xsl:apply-templates/>`.

Im unteren Teil des Stylesheets sind mehrere Templates abgelegt, die bestimmen, wie die verschiedenen Elemente, aus denen sich die Daten für einen Kurs zusammensetzen, ausgegeben werden. Dabei soll für jedes Element – `<name>`, `<referent>`, `<termin>` und `<beschreibung>` – jeweils ein eigener Block mit einer speziellen Formatisierung erzeugt werden. Um die einzelnen Elemente auszuwählen, werden entsprechende Suchmuster verwendet, etwa:

```
<xsl:template match="name">
<xsl:template match="referent">
```

Wo erforderlich, wird noch eine Beschriftung für die Daten hinzugefügt. Das Stylesheet kann dann insgesamt so aussehen:

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

<xsl:template match="kursprogramm">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master master-name="Standardseite"
        page-height="400mm" page-width="300mm" margin-top="10mm"
        margin-left="15mm" margin-bottom="10mm"
        margin-right="15mm">
        <fo:region-body margin-top="20mm" margin-left="5mm"
          margin-bottom="0mm" margin-right="5mm"/>
        <fo:region-before extent="15mm"/>
        <fo:region-after extent="15mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="Standardseite"
      initial-page-number="1">
      <fo:static-content flow-name="xsl-region-before">
        <fo:block background-color="lightgray"
          font-family="Helvetica" font-size="22pt" padding="5pt">
          Kursprogramm 2019
        </fo:block>
      </fo:static-content>
      <fo:static-content flow-name="xsl-region-after">
        <fo:block font-family="Helvetica" font-size="18pt">
          Seite:
          <fo:page-number/>
        </fo:block>
      </fo:static-content>
      <fo:flow flow-name="xsl-region-body">
        <xsl:for-each select="//kurs">
          <xsl:apply-templates/>
        </xsl:for-each>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
```

```

<xsl:template match="name">
  <fo:block space-before="15pt" font-size="36pt">
    <xsl:value-of select=". "/>
  </fo:block>
</xsl:template>

<xsl:template match="referent">
  <fo:block font-size="24pt">
    Referent:
    <xsl:value-of select=". "/>
  </fo:block>
</xsl:template>

<xsl:template match="termin">
  <fo:block font-size="18pt">
    Termin:
    <xsl:value-of select=". "/>
  </fo:block>
</xsl:template>

<xsl:template match="beschreibung">
  <fo:block background-color="lightblue" font-size="16pt"
    padding="5pt">
    <xsl:value-of select=". "/>
  </fo:block>
</xsl:template>

</xsl:stylesheet>

```

Listing 8.2 kursliste_FO_1.xsl

8.7 Verknüpfung mit dem Dokument und Ausgabe

Die Verknüpfung des XML-Dokuments mit der XSL-Datei geschieht wieder über eine entsprechende Verarbeitungsanweisung, die in den Prolog eingefügt wird:

```
<?xml-stylesheet href="kursliste_FO_1.xsl" type="text/xsl" ?>
```

Auf das geöffnete XML-Dokument mit der Verknüpfung zum XSL-Stylesheet kann dann in der Umgebung von XMLSpy mit dem Befehl XSL/XQUERY • XSL:FO-TRANSFORMATION das Stylesheet angewandt werden, wobei FOP die Ausgabe in das gewünschte Format vornimmt.

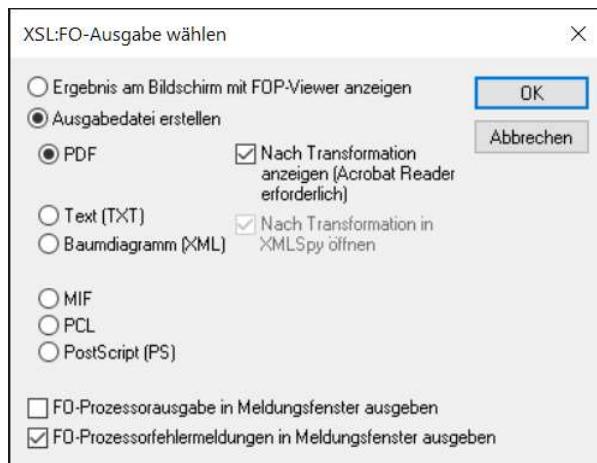


Abbildung 8.6 Ausgabeoptionen von FOP

Sie können in einem Dialog wählen, ob Sie den Output gleich am Bildschirm sehen oder eine Ausgabedatei verwenden wollen. Dafür werden verschiedene Formate wie zum Beispiel PDF oder PostScript angeboten.

Abbildung 8.7 Ausgabe des Kursprogramms als PDF-Datei

8.8 Inline-Formatierungsobjekte

Neben der oben verwendeten Blockformatierung lassen sich auch einzelne Zeichenfolgen innerhalb eines Blocks gesondert formatieren. Dazu wird eine Gruppe von FOs verwendet, die auf der Ebene der Zeile angesiedelt sind. Sie dienen dazu, etwa das

erste Zeichen eines Wortes mit einem fetten Buchstaben auszugeben, die erste Zeile eines Absatzes gesondert zu formatieren oder Seitenzahlen in eine Kopf- oder Fußzeile einzufügen. Im Folgenden erstellen wir ein Beispiel für eine Seitennummerierung im Fuß der Seite, die rot erscheinen soll.

Zunächst muss dafür im layout-master-set eine entsprechende Region eingerichtet sein:

```
<fo:region-after extent="15mm"/>
```

Dann kann dieser Region ein <fo:static-content>-Objekt zugeordnet werden:

```
<fo:static-content flow-name="xsl-region-after">
  <fo:block font-family="Helvetica"
            font-size="18pt">Seite:
    <fo:inline color="red">
      <fo:page-number />
    </fo:inline>
  </fo:block>
</fo:static-content>
```

Wenn Sie das Element <fo:inline> verwenden, können Sie einem Teil eines Textes fast alle Eigenschaften zuordnen, die auch für Blockobjekte möglich sind, also nicht nur eine eigene Schriftgestaltung, sondern auch eigene Text- und Hintergrundfarben, Einrahmungen etc.

8.9 Ausgabe von Tabellen

Bei dem von uns verwendeten Beispiel könnte alternativ zu der bisher benutzten Ausgabe in Form von Textabschnitten auch gut mit einer Tabelle oder einer Liste gearbeitet werden. Für beide Darstellungsformen stehen in XSL Gruppen von Formatierungsobjekten bereit, die wir hier wenigstens kurz vorstellen wollen.

8.9.1 Tabellenstruktur

Um eine Tabelle aufzubauen, stehen neun Objekte zur Verfügung, deren Hierarchie der in Abbildung 8.8 dargestellte Baum zeigt.

In unserem Beispiel wird zunächst ein <fo:table>-Objekt als Kind eines <fo:flow>-Objekts angelegt. Wenn Sie dafür den XMLSpy-Editor verwenden, werden automatisch die notwendigen Kindobjekte eingefügt, wie Abbildung 8.9 zeigt.

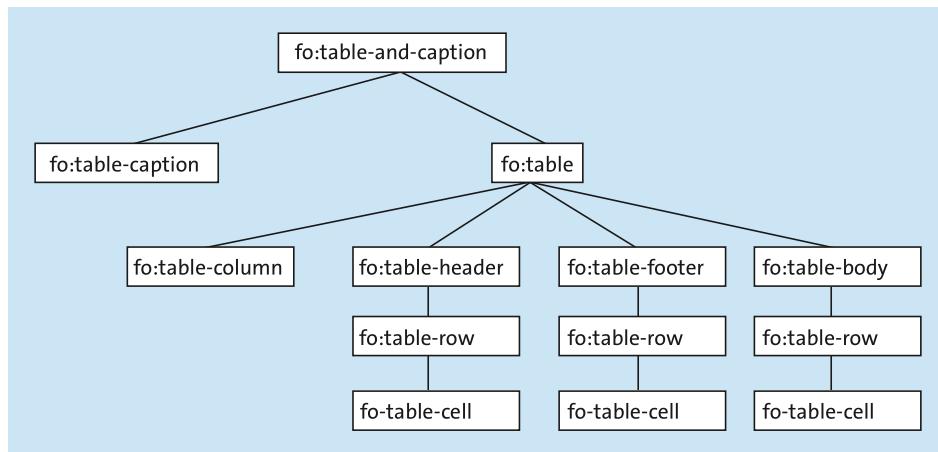


Abbildung 8.8 Baum der Tabellenobjekte

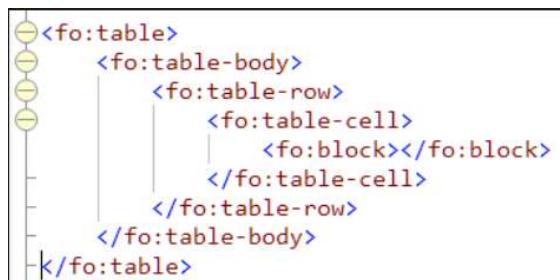


Abbildung 8.9 »table«-Objekt im XSL-Editor von XMLSpy

Zunächst muss für jede vorgesehene Spalte ein `<fo:table-column>`-Objekt angelegt werden. In diesem Fall wird die Spaltenbreite fest vorgegeben. Die Spalten können mit Nummern versehen werden, auf die dann bei den späteren `<fo:table-cell>`-Objekten Bezug genommen werden kann.

Ist das geschehen, folgt ein `<fo:table-body>`-Objekt, als dessen Kinder anschließend die einzelnen Tabellenzeilen `<fo:table-row>` gebildet werden. Diese wiederum setzen sich aus den `<fo:table-cell>`-Objekten zusammen, die schließlich die Inhalte der Tabelle aufnehmen. Die Tabellenzeilen werden in eine `<xsl:for-each>`-Schleife gepackt, die die verschiedenen Kurselmente abarbeitet.

8.9.2 Zellinhalte

Für die Formatierung der Zellinhalte sind in diesem Fall zwei Templates angelegt, die die Elemente `<name>` und `<beschreibung>` innerhalb des Kursprogramms betreffen. Beide enthalten jeweils ein `<fo:block>`-Objekt, das für die Darstellung der Zellinhalte benötigt wird.

Die einzelnen Tabellenzellen werden über die Spaltennummer der gewünschten Spalte zugeordnet. Der Inhalt jeder Zelle wird durch das XSLT-Konstrukt `<xsl:apply-templates>` geliefert. Damit nun aber in jede Zelle nur das hineingeschrieben wird, was dorthin gehört, wird mit Hilfe des `select`-Attributs genau das Element aus dem Quelldokument ausgewählt, das für die jeweilige Spalte vorgesehen ist. Der Ausdruck

```
select="name"
```

wählt zum Beispiel jeweils das aktuelle `name`-Element des Elements `kurs` aus.

Der Code für den Tabellenteil des Stylesheets sieht insgesamt so aus:

```
...
<fo:flow flow-name="xsl-region-body">
  <fo:table>
    <fo:table-column column-width="80mm" column-number="1"/>
    <fo:table-column column-width="175mm" column-number="2"/>
    <fo:table-body>
      <xsl:for-each select="/kursprogramm/kurs">
        <fo:table-row>
          <fo:table-cell column-number="1">
            <xsl:apply-templates select="name"/>
          </fo:table-cell>
          <fo:table-cell column-number="2">
            <xsl:apply-templates select="beschreibung"/>
          </fo:table-cell>
        </fo:table-row>
      </xsl:for-each>
    </fo:table-body>
  </fo:table>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="beschreibung">
  <fo:block space-before="10pt" font-size="12pt">
    <xsl:value-of select=". "/>
  </fo:block>
</xsl:template>
<xsl:template match="name">
  <fo:block space-before="10pt" font-size="18pt">
    <xsl:value-of select=". "/>
  </fo:block>
```

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

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

Listing 8.3 kursliste_FO_2.xsl

Die Ausgabe der Vorschau, die der FO-Prozessor daraus erzeugt, ist in [Abbildung 8.10](#) dargestellt.

Kursprogramm 2019	
XML-Grundlagen	Einführungsseminar für Programmierer, IT-Manager und Contentmanager
XSL-Praxis	Praktische Übungen für Programmierer und Contentmanager. Das Seminar ist in drei Blöcke aufgeteilt, die drei Schwierigkeitstufen entsprechen. Zum Abschluss wird ein Test gefahren, um den Wissenstand zu ermitteln.
XSLT-Einstieg	Einführung in die Transformation von XML-Dokumenten für Programmierer und Contentmanager

Abbildung 8.10 FOP-Output für eine Tabelle

8.10 Listen

Weniger aufwendig als die Konstruktion von Tabellen mit XSL ist der Aufbau von Listen. Hier werden nur vier FOs benötigt. Die komplette Liste wird zunächst in ein Element `<fo:list-block>` eingeschlossen, anschließend werden die einzelnen Listenelemente in `<fo:list-item>`-Elemente verpackt. Um ein Label für eine Listenposition zu erzeugen, kann `<fo:list-item-label>` verwendet werden. Der Inhalt der Position wird schließlich über das Element `<fo:list-item-body>` eingebaut.

Wie beim `table`-Objekt wird auch das `list-block`-Objekt als Kind eines `flow`-Objekts gehandhabt. Und wie bei `table-cell` wird ein `block`-Objekt verwendet, um das `list-item-body`-Objekt darzustellen.

Das Stylesheet für die Kursliste in Listenform sieht dann in dem Listenteil so aus:

```
...
<fo:flow flow-name="xsl-region-body">
  <fo:list-block>
    <xsl:for-each select="/kursprogramm/kurs">
      <fo:list-item>
        <fo:list-item-label>
          <fo:block font-size="18pt">
            <xsl:value-of select="position()"/>
            <xsl:text>. </xsl:text>
          </fo:block>
        </fo:list-item-label>
        <fo:list-item-body>
```

```
<fo:block>
  <xsl:apply-templates select="name"/>
  <xsl:apply-templates select="beschreibung"/>
</fo:block>
</fo:list-item-body>
</fo:list-item>
</xsl:for-each>
</fo:list-block>
</fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="beschreibung">
  <fo:block space-after="20pt" font-size="12pt">
    <xsl:value-of select=".."/>
  </fo:block>
</xsl:template>

<xsl:template match="name">
  <fo:block text-indent="20pt" font-size="18pt">
    <xsl:value-of select=".."/>
  </fo:block>
</xsl:template>
```

Listing 8.4 Code für die Ausgabe als Liste

Wieder wurde eine `<xsl:for-each>`-Schleife verwendet, um die einzelnen Kursdaten auszuwerten. Um die Nummerierung der Kurse automatisch zu erzeugen, wird innerhalb des `list-item-label`-Objekts die Positionsnummer des aktuellen Kursknotens ausgewertet. Um einen Punkt und ein Leerzeichen dahinter zu setzen, wird noch ein entsprechendes `<xsl:text>`-Element eingefügt.

Der Block mit dem Kursnamen wird mit Hilfe des `text-indent`-Attributs etwas eingerückt, damit die Kursnummer nicht überschrieben wird. Der FO-Prozessor gibt die Liste wieder, wie in [Abbildung 8.11](#) dargestellt.

Kursprogramm 2019

1. XML-Grundlagen

Einführungsseminar für Programmierer, IT-Manager und Contentmanager

2. XSL-Praxis

Praktische Übungen für Programmierer und Contentmanager. Das Seminar ist in drei Blöcke aufgeteilt, die drei Schwierigkeitstufen entsprechen. Zum Abschluss wird ein Test gefahren, um den Wissenstand zu ermitteln.

3. XSLT-Einstieg

Einführung in die Transformation von XML-Dokumenten für Programmierer und Contentmanager

Abbildung 8.11 Das Kursprogramm als Liste

Natürlich stehen für die FOs zur Listenerzeugung wieder alle Eigenschaften zur Verfügung, die für die Gestaltung von Listen sinnvoll sind. Wir können hier nur wieder auf die Empfehlung des W3C verweisen, die zu jedem Formatierungsobjekt die definitive Liste der Eigenschaften aufführt.

8.11 Gesucht: visuelle Editoren

Sie werden nach diesen Abschnitten vermutlich zu der Einschätzung kommen, dass XSL Formatting Objects verglichen mit CSS eine ziemlich komplexe Sache geworden ist, auch wenn viele Eigenschaften und Wertoptionen von CSS übernommen worden sind.

So richtig marktfähig wird XSL sicherlich erst werden, wenn den Designern ausgereifte visuelle Editoren zur Verfügung stehen, die dann den doch ziemlich komplexen Code automatisch erzeugen, so wie es heute im Bereich des Webdesigns üblich ist.

8.12 Übersicht über die Formatierungsobjekte von XSL

Die folgenden Tabellen geben einen Überblick über die verfügbaren Formatierungsobjekte, gegliedert nach den unterschiedlichen Funktionen, die diese Objekte bei der Gestaltung des Dokuments erfüllen.

8.12.1 Übergeordnete Objekte

Element	Bedeutung
color-profile	Deklariert ein Farbprofil für ein Stylesheet.
conditional-page-master-reference	fo:conditional-page-master-reference identifiziert einen Page-Master, der gebraucht werden soll, wenn die Bedingungen für seine Verwendung erfüllt sind.
declarations	Wird verwendet, um globale Deklarationen für ein Stylesheet zu gruppieren.
flow	Der Inhalt von fo:flow ist eine Folge von Flow-Objects, die den fließenden Textinhalt liefern, der auf die Seiten verteilt wird.
layout-master-set	fo:layout-master-set ist eine Zusammenstellung aller im Dokument verwendeten Master.

Tabelle 8.1 Liste der übergeordneten Formatierungsobjekte

Element	Bedeutung
page-sequence	fo:page-sequence wird verwendet, um anzugeben, wie eine Sequenz oder Untersequenz von Seiten innerhalb eines Dokuments erzeugt werden soll, zum Beispiel das Kapitel eines Berichts. Der Inhalt dieser Seiten kommt von Flow-Kindern von fo:page-sequence.
page-sequence-master	fo:page-sequence-master gibt eine Reihe von Page-Mastern an, die gebraucht werden, um eine Folge von Seiten zu generieren.
region-after	Diese Region definiert eine Arbeitsfläche, die sich auf der after-Seite von fo:region-body befindet.
region-before	Diese Region definiert eine Arbeitsfläche, die sich auf der before-Seite von fo:region-body befindet.
region-body	Diese Region gibt ein Arbeitsfläche-Referenz-Paar an, das sich im center von fo:simple-page-master befindet.
region-end	Diese Region definiert eine Arbeitsfläche, die sich auf der end-Seite von fo:region-body befindet.
region-start	Diese Region definiert eine Arbeitsfläche, die sich auf der start-Seite von fo:region-body befindet.
repeatable-page-master-alternatives	fo:repeatable-page-master-alternatives gibt eine Untersequenz an, die aus wiederholten Instanzen eines Satzes von alternativen Page-Mastern besteht. Die Anzahl von Wiederholungen kann begrenzt werden oder unbegrenzt sein.
repeatable-page-master-reference	fo:repeatable-page-master-reference gibt eine Untersequenz an, die aus wiederholten Instanzen eines einzelnen Page-Masters besteht. Die Anzahl von Wiederholungen kann begrenzt werden oder unbegrenzt sein.
root	fo:root ist der oberste Knoten eines XSL-Ergebnisbaums. Dieser Baum ist aus Formatierungsobjekten zusammengesetzt.
simple-page-master	fo:simple-page-master wird bei der Erzeugung von Seiten verwendet und gibt die Geometrie der Seite an. Die Seite kann in bis zu fünf Regionen unterteilt werden.
single-page-master-reference	fo:single-page-master-reference gibt eine Untersequenz an, die aus einer einzelnen Instanz eines einzelnen Page-Masters besteht.

Tabelle 8.1 Liste der übergeordneten Formatierungsobjekte (Forts.)

Element	Bedeutung
static-content	fo:static-content enthält eine Sequenz oder einen Baum von Formatierungsobjekten, die in einer einzelnen Region dargestellt und in gleich bezeichneten Regionen auf einer oder mehreren Seiten in der Seitenfolge wiederholt werden sollen. Wird üblicherweise verwendet für gleichbleibende oder fortlaufende Kopf- oder Fußzeilen.
title	fo:title wird verwendet, um einen Titel mit einer gegebenen Seitenfolge zu verbinden. Dieser Titel kann von einem interaktiven Benutzeragenten verwendet werden, um die Seiten zu identifizieren. Zum Beispiel kann der Inhalt von fo:title in einem Titel-Fenster oder in einem Tooltip ausgegeben werden.

Tabelle 8.1 Liste der übergeordneten Formatierungsobjekte (Forts.)

8.12.2 Blockformatierung

Element	Bedeutung
block	fo:block wird üblicherweise für das Formatieren von Absätzen, Titeln, Schlagzeilen, Beschriftungen von Tabellen etc. verwendet.
block-container	fo:block-container, ein Flow-Object, wird verwendet, um einen Referenzbereich auf Blockebene zu generieren.

Tabelle 8.2 Liste der Objekte für die Blockformatierung

8.12.3 Inline-Formatierung

Element	Bedeutung
bidi-override	fo:bidi-override ist ein Inline-Formatierungsobjekt, das verwendet wird, um eine bestimmte Schreibrichtung zu erzwingen, wenn der Unicode-bidirectional-Algorithmus nicht funktioniert. Dieser Algorithmus wird verwendet, um Zeichen von rechts nach links fließen zu lassen, wie etwa in arabischen oder hebräischen Texten.
character	Das fo:character-Flow-Object stellt ein Zeichen dar, dem zur grafischen Darstellung eine Glyphen zugeordnet wird.

Tabelle 8.3 Liste der Objekte für die Inline-Formatierung

Element	Bedeutung
external-graphic	Das fo:external-graphic-Flow-Object wird für eine Grafik verwendet, wenn sich die grafischen Daten außerhalb des FO-Elementebaums befinden.
initial-property-set	fo:initial-property-set setzt die Formateigenschaften für die erste Zeile von fo:block.
inline	fo:inline wird für das Formatieren eines Textteils mit einem Hintergrund oder durch Einschließen in einen Rahmen verwendet.
inline-container	Das Flow-Object fo:inline-container wird verwendet, um einen Inline-Referenzbereich zu generieren.
instream-foreign-object	Das Flow-Object fo:instream-foreign-object wird für eine Inline-Grafik oder ein anderes »generisches« Objekt verwendet, bei dem die Objektdaten als Nachkommen von fo:instream-foreign-object vorliegen.
leader	fo:leader wird benutzt, um Führungszeichen zu generieren, die entweder aus einer Regel oder aus einer Reihe sich wiederholender Zeichen bestehen oder zyklisch Muster von Zeichen wiederholen, die für das Verbinden von zwei Textformatierungsobjekten verwendet werden können.
page-number	fo:page-number wird verwendet, um die aktuelle Seitennummer darzustellen.
page-number-citation	fo:page-number-citation wird verwendet, um auf die Seitennummer der Seite zu verweisen, die den ersten vom angeführten Formatierungsobjekt zurückgegebenen normalen Bereich enthält.

Tabelle 8.3 Liste der Objekte für die Inline-Formatierung (Forts.)

8.12.4 Tabellenformatierung

Element	Bedeutung
table	Das Flow-Object fo:table wird für die Formatierung des tabellarischen Materials einer Tabelle verwendet.
table-and-caption	Das Flow-Object fo:table-and-caption gruppiert die Elemente fo:table-caption und fo:table.

Tabelle 8.4 Objekte für die Formatierung von Tabellen

Element	Bedeutung
table-body	fo:table-body wird verwendet, um den Inhalt des Tabellenkörpers aufzunehmen.
table-caption	fo:table-caption wird verwendet, um Formatierungsobjekte aufzunehmen, die die Überschrift für die Tabelle enthalten; das gilt aber nur, wenn fo:table-and-caption verwendet wird.
table-cell	fo:table-cell wird verwendet, um Inhalte in den Tabellenzellen zu platzieren.
table-column	fo:table-column gibt Merkmale für Tabellenzellen an, die in derselben Spalte liegen.
table-footer	fo:table-footer wird verwendet, um den Inhalt des Tabellenfußes aufzunehmen.
table-header	fo:table-header wird verwendet, um den Inhalt des Tabelenkopfes aufzunehmen.
table-row	fo:table-row gibt Merkmale für Tabellenzellen an, die in derselben Zeile liegen.

Tabelle 8.4 Objekte für die Formatierung von Tabellen (Forts.)

8.12.5 Listenformatierung

Element	Bedeutung
list-block	Das Flow-Object fo:list-block wird verwendet, um eine Liste zu formatieren.
list-item	fo:list-item enthält das Label und den Körper eines Listeneintrags.
list-item-body	fo:list-item-body enthält den eigentlichen Inhalt des Listeneintrags.
list-item-label	fo:list-item-label enthält den Inhalt des Labels für einen Listeneintrag und wird typischerweise verwendet, um die Listeneinträge zu nummerieren, zu identifizieren oder zu schmücken.

Tabelle 8.5 Objekte für die Formatierung von Listen

8.12.6 Formatierung für Verknüpfungen

Element	Bedeutung
basic-link	fo:basic-link wird für die Angabe der Basisadresse eines einfachen Links verwendet.
multi-case	fo:multi-case wird verwendet, um innerhalb von fo:multi-switch jeden alternativen Unterbaum von Formatierungsobjekten aufzunehmen, aus dem dann das Elternelement fo:multi-switch einen Unterbaum zur Darstellung auswählt.
multi-properties	fo:multi-properties wird verwendet, um zwischen zwei oder mehr Eigenschaftssätzen umzuschalten, die mit einem gegebenen Teil des Inhalts verbunden sind.
multi-property-set	fo:multi-property-set wird verwendet, um einen alternativen Satz von Formateigenschaften anzugeben, die abhängig vom Status des Benutzeragenten für den Inhalt gelten sollen.
multi-switch	fo:multi-switch verpackt die Spezifikation alternativer Unterbäume von Formatierungsobjekten (jeder Unterbaum befindet sich innerhalb von fo:multi-case) und kontrolliert das Umschalten (aktiviert über fo:multi-toggle) von einer Alternative zur anderen.
multi-toggle	fo:multi-toggle wird innerhalb von fo:multi-case benutzt, um auf ein anderes fo:multi-case umzuschalten.

Tabelle 8.6 Objekte für die Formatierung von Verknüpfungen

8.12.7 Out-of-Line-Formatierung

Element	Bedeutung
float	fo:float wird typischerweise benutzt, um zu erreichen, dass ein Bild in einer separaten Region zu Beginn der Seite positioniert wird, oder um ein Bild rechts oder links neben einen fließenden Text zu setzen.
footnote	Fo:footnote wird verwendet, um ein Fußnotenzitat und die entsprechende Fußnote zu produzieren.
footnote-body	fo:footnote-body wird verwendet, um den Inhalt der Fußnote zu generieren.

Tabelle 8.7 Objekte für die Out-of-Line-Formatierung

8.12.8 Andere Objekte

Element	Bedeutung
marker	fo:marker wird in Verbindung mit fo:retrieve-marker verwendet, um fortlaufende Kopf- oder Fußzeilen zu produzieren.
retrieve-marker	fo:retrieve-marker wird in Verbindung mit fo:marker verwendet, um fortlaufende Kopf- oder Fußzeilen zu erzeugen.
wrapper	fo:wrapper wird verwendet, um ererbte Eigenschaften für eine Gruppe von Formatierungsobjekten anzugeben. Es hat keine zusätzliche Formatierungssemantik.

Tabelle 8.8 Andere Objekte