XSLT (XSLT)

Lernziele
• Sie haben das Prinzip des XSLT-Transformationsprozesses verstanden. • Sie sind in der Lage, ein XSLT-Stylesheet mit einem XML-Dokument zu verknüpfen. • Sie sind in der Lage, Stylesheets zur Transformation von XML-Dokumenten zu erstellen. • Sie kennen die grundlegenden XSLT-Elemente. • Sie wissen, wie Sie mit XSLT sortieren und nummerieren. • Sie können CSS und Skripte in Ihre XSLT-Stylesheets einbinden. • Sie können XML-Dateien miteinander verlinken. • Sie setzen das erworbene Wissen praktisch in den Übungen ein!

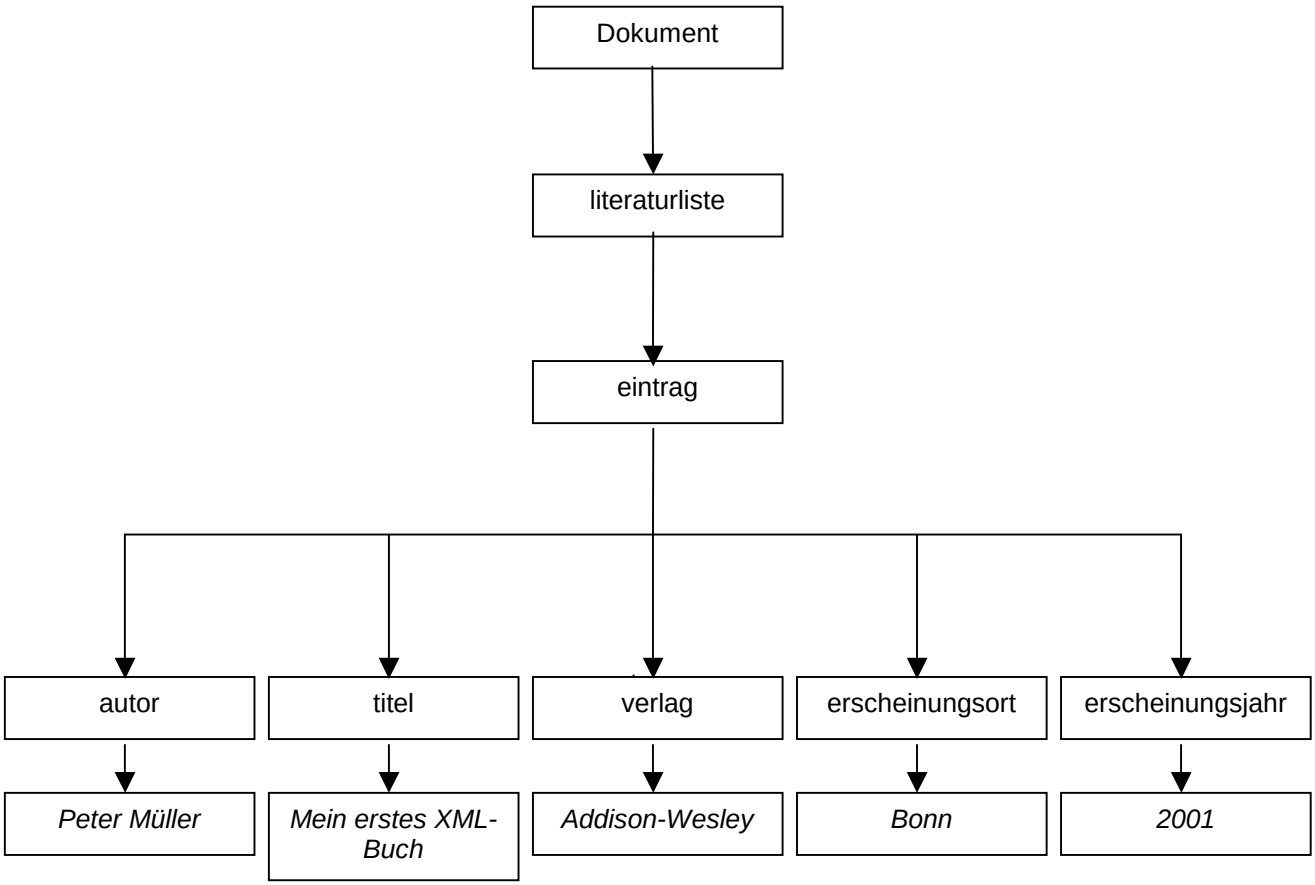
Ein erstes Beispiel

Beispiel XSLT-1: Erstes XSLT-Beispiel

Input

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../xsl/litliste.xsl" ty-
pe="text/xsl"?>
<literaturliste>
  <eintrag>
    <autor>Peter Müller</autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>
</literaturliste>
```

Die Baumansicht des Dokuments:



Stylesheet

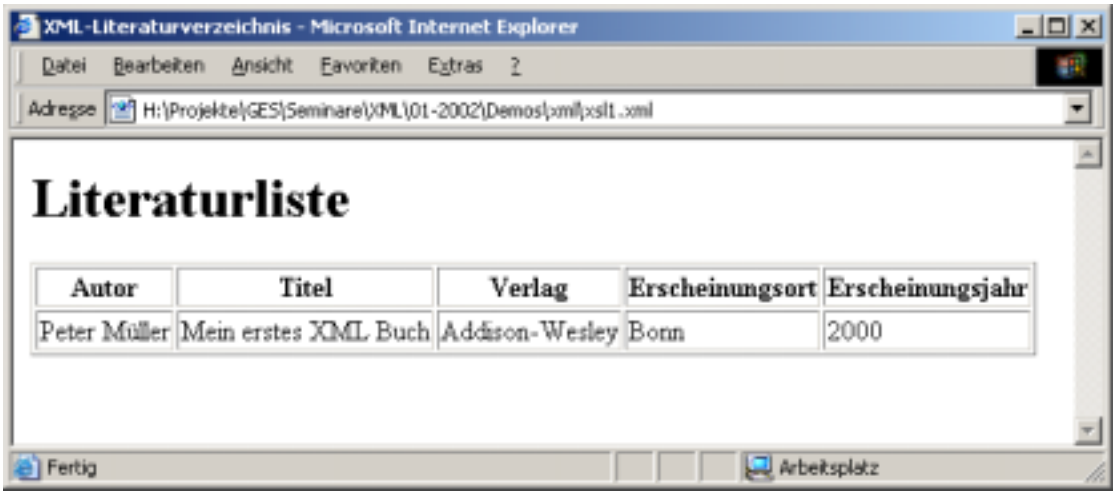
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
     version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/literaturliste/eintrag">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <tr>
            <td>
              <xsl:value-of select="autor"/>
            </td>
            <td>
              <xsl:value-of select="titel"/>
            </td>
            <td>
              <xsl:value-of select="verlag"/>
            </td>
            <td>
              <xsl:value-of select="erscheinungsort"/>
            </td>
            <td>
              <xsl:value-of select="erscheinungsjahr"/>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Das resultierende HTML-Dokument sieht folgendermaßen aus:

Ausgabe

```
<html>
  <head>
    <title>XML-Literaturverzeichnis</title>
  </head>
  <body>
    <h1>Literaturliste</h1>
    <table border="1">
      <tr>
        <th>Autor</th>
        <th>Titel</th>
        <th>Verlag</th>
        <th>Erscheinungsort</th>
        <th>Erscheinungsjahr</th>
      </tr>
      <tr>
        <td>Peter Müller</td>
        <td>Mein erstes XML Buch</td>
        <td>Addison-Wesley</td>
        <td>Bonn</td>
        <td>2000</td>
      </tr>
    </table>
  </body>
</html>
```

Ansicht im Browser



Der Transformationsprozess

Ein XSLT-Prozessor arbeitet mit einem **Quellbaum**, den ein Parser aus dem **Quelldokument** generiert. Das **XSLT-Stylesheet** liegt ebenfalls als Baum vor. (Es handelt sich ja ebenfalls um ein XML-Dokument). Aus beiden erzeugt der XSLT-Prozessor im Transformationsprozess einen **Ergebnisbaum**, der dann im entsprechenden Format als **Ergebnisdokument** ausgegeben wird.

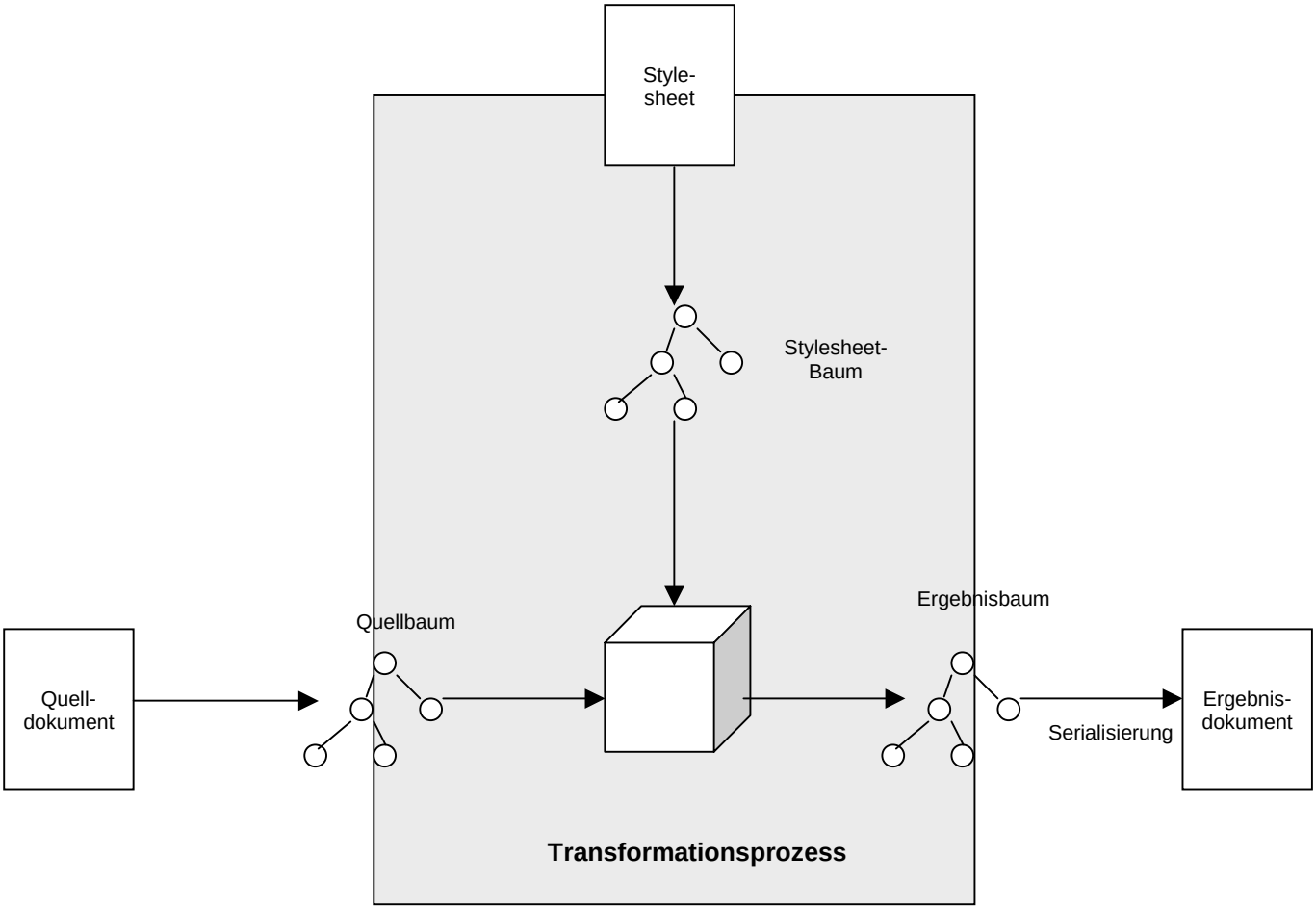


Abbildung XSLT-1: Bäume im XSLT-Transformationsprozess

Der Vorteil, dass Input und Output einer XSLT-Transformation jedes mal ein Baum ist, liegt in der Möglichkeit, den Output direkt als Input eines weiteren sich anschließenden Transformationsschritts nutzen zu können.

XSLT-Prozessoren

Name	Lizenz	Entwicklung	implementiert in
Xalan	open source	Apache XML Project	Java, C++
Saxon	open source	Michael Kay	Java
XT	open source	James Clark	Java
MSXML	frei	Microsoft	DLL

Verknüpfung XML-Dokument und Stylesheet

```
<?xml-stylesheet href="URI" type="text/xml"?>
```

Ein Stylesheet binden Sie über eine Verarbeitungsanweisung in ein XML-Dokument ein. Im `href`-Attribut geben Sie über einen URI den Pfad zur Datei an, die das assoziierte Stylesheet enthält. Der `type` des Stylesheet sollte laut Spezifikation eigentlich `text/xml` sein. Für den IE müssen Sie allerdings `text/xsl` angeben, da sonst das Stylesheet nicht assoziiert wird.

Beispiel XSLT-2: Verknüpfung Stylesheet und XML-Dokument

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

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

<wurzelelement>
.......
</wurzelelement>
```

Mittels der gleichen Verarbeitungsanweisung werden auch CSS mit XML-Dokumenten verknüpft!

Die Dateiendung für XSLT-Stylesheets lautet `.xsl`.

Struktur eines XSLT-Stylesheets

Ein Stylesheet beginnt mit einer XML-Deklaration, da ein XSLT-Stylesheet ein XML-Dokument ist.

Folgende XSLT-Elemente haben wir in obigem Beispiel kennen gelernt:

- `<xsl:stylesheet>`
- `<xsl:template>`
- `<xsl:value-of>`

Das Wurzelelement: `<xsl:stylesheet>`

Das Attribut `version` ist ein Pflichtattribut. Hier geben Sie die XSLT-Version an, die vom Stylesheet genutzt wird (aktuell ist das die Version 1.0).

Ins Wurzelelement gehört auch die XSLT-Namensraumangabe. Der XSLT-Namensraum verwendet das Präfix `xsl` und hat den URI `http://www.w3.org/1999/XSL/Transform`.

Statt `<xsl:stylesheet>` kann alternativ `<xsl:transform>` stehen.

Template Rules: `<xsl:template>`

Templates sind das zentrale Element eines jeden Stylesheet. Mit dem Element `<xsl:template>` definieren Sie eine Transformationsregel für einen bestimmten Ausschnitt des Quelldokuments. Im Fach-XSLT heißt eine solche Regel **Template Rule**. Der "Body" der Template Rule wird als **Template** bezeichnet.

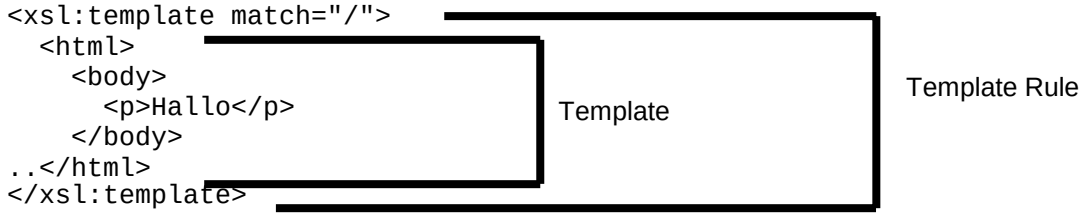


Abbildung XSLT-2: Template und Template Rule

Eine Template Rule besagt: "Schreibe den Inhalt der Template Rule (also das Template) in den Ergebnisbaum und führe alle Anweisungen mit `xsl`-Namensraum aus." Die Elemente, die nicht zum `xsl`-Namensraum gehören und in den Ergebnisbaum geschrieben werden, heißen **Literale Ergebniselemente (*Literal Result Elements*)**

Über das Attribut `match` des `<xsl:template>`-Elements, selektieren Sie den oder die Knoten des XML-Dokuments, auf den die Template Rule angewendet werden soll. In unserem Beispiel also das Element `eintrag`.

Der Attributwert des `match`-Attributs ist ein so genannter **XPath-Ausdruck**, in diesem speziellen Fall auch **Muster (pattern)** genannt. Mittels eines solchen Ausdrucks navigieren Sie durch den Quellbaum. Zum ersten Verständnis reicht es, wenn Sie sich einen solchen Ausdruck als eine Pfadangabe durch den Baum vorstellen, etwa wie in einem Unix-Dateibaum. Der Schrägstrich deutet dann jeweils eine Verzweigung in eine tiefere Ebene an. In unserem Beispiel (`/literaturliste/eintrag`) gilt die Regel also für das Element `<eintrag>`, welches ein Kind des Elements `<literaturliste>` ist. Der führende Schrägstrich steht für den Wurzelknoten.

Text aus dem Quelldokument: `<xsl:value-of>`

Mittels des `<xsl:value-of>`-Konstrukts setzen Sie Text aus dem Quelldokument in das Ergebnisdokument ein. Im `select`-Attribut wird der Knoten angegeben, dessen Textinhalt in das Ergebnisdokument übernommen werden soll.

Das Ausgabeformat kontrollieren (<xsl:output>)

Das <xsl:output>-Element ist ein sogenanntes **Top-Level-Element**, d.h. es darf nur als Kind von <xsl:stylesheet> auftauchen, also nur auf der "obersten Ebene" des Stylesheets. Es dient der Kontrolle des Ausgabeformats des Stylesheets. Es betrifft somit die Stufe des XSLT-Verarbeitungsprozesses, bei der der Ergebnisbaum in eine serielle Datei geschrieben wird.

Beachten Sie:
Ein XSLT-Prozessor ist laut Spezifikation nicht gezwungen, den Ergebnisbaum in eine Ausgabedatei zu serialisieren. Er kann den Baum auch z.B. über die DOM-API zur Verfügung stellen. Diese Prozessoren können das <xsl:output>-Element ignorieren.

Das wichtigste Attribut des <xsl:output>-Elements ist das method-Attribut, welches festlegt, ob das Ausgabeformat HTML, XML oder einfach Text sein soll.

Beispiel XSLT-3: method-Attribut des <xsl:output>-Elements

```
<xsl:output method="xml"/>
```

HTML als Ausgabeformat

Innerhalb eines XSLT-Stylesheets müssen Sie auf Wohlgeformtheit achten. D.h. leere HTML-Elemente müssen ein Abschluss-Tag erhalten, obwohl dies laut HTML nicht notwendig ist und in einigen Browsern auch Probleme bereiten kann. Wollen Sie diese Abschluss-Tags in ihrer Ausgabedatei entfernen, also "reines" HTML erzeugen, so müssen Sie das <xsl:output>-Element verwenden und dessen Attribut method mit dem Wert html versehen. Dann werden gleichzeitig auch die deutschen Umlaute durch HTML-Zeichenreferenzen ersetzt (z.B. ä durch ä)

Fügen wir eine durchgezogene Linie unterhalb der Überschrift in die Literaturliste ein, so können wir mit dem <xsl:output>-Element sicherstellen, dass die Ausgabe des <hr/>-Elements als <hr> erfolgt. Außerdem wird der Umlaut ü im Namen Müller durch eine HTML-Entity-Referenz (ü) ersetzt.

Beispiel XSLT-4: <xsl:output>

Stylesheet

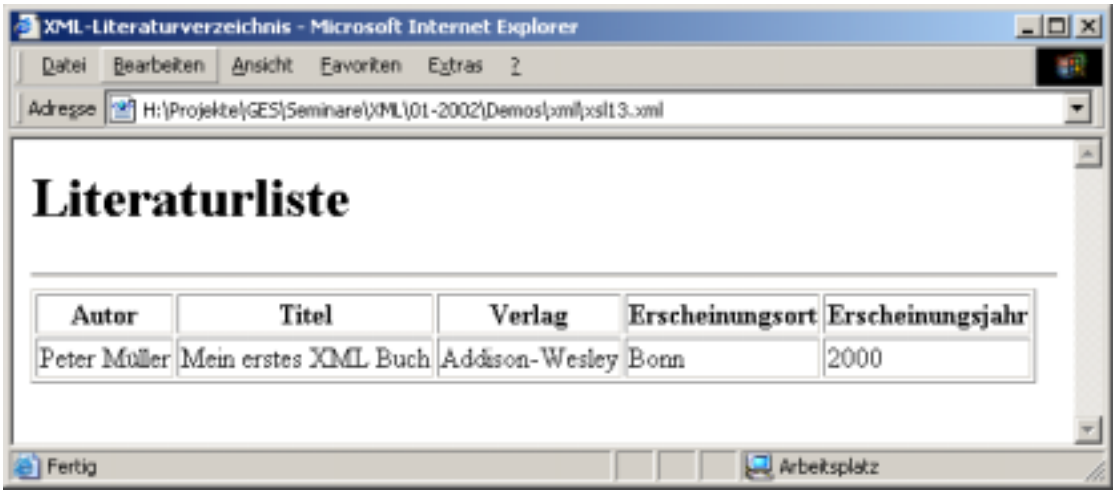
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
     version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="html"/>

  <xsl:template match="/literaturliste/eintrag">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <hr/>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <tr>
            <td>
              <xsl:value-of select="autor"/>
            </td>
            <td>
              <xsl:value-of select="titel"/>
            </td>
            <td>
              <xsl:value-of select="verlag"/>
            </td>
            <td>
              <xsl:value-of select="erscheinungsort"/>
            </td>
            <td>
              <xsl:value-of select="erscheinungsjahr"/>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
```

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

Ergebnisdokument

```
<html>
  <head>
    <title>XML-Literaturverzeichnis</title>
  </head>
  <body>
    <h1>Literaturliste</h1>
    <hr>
    <table border="1">
      <tr>
        <th>Autor</th>
        <th>Titel</th>
        <th>Verlag</th>
        <th>Erscheinungsort</th>
        <th>Erscheinungsjahr</th>
      </tr>
      <tr>
        <td>Peter M&uuml;ller</td>
        <td>Mein erstes XML Buch</td>
        <td>Addison-Wesley</td>
        <td>Bonn</td>
        <td>2000</td>
      </tr>
    </table>
  </body>
</html>
```

Ansicht im Browser



Praxistipp:
Möchten Sie die Ausgabe des HTML-Codes durch Einrückungen leserlicher machen, nutzen Sie ein weiteres Attribut des `<xsl:output>`-Elements, das Attribut `indent`, und setzen dessen Wert auf `yes`:

Beispiel XSLT-5: Das Attribut `indent` des `<xsl:output>`-Elements

```
<xsl:output method="html" indent="yes"/>
```

Geben Sie kein explizites Ausgabeformat für Ihr Ergebnisdokument an, so sollte der XSLT-Prozessor standardmäßig XML als Ausgabeformat annehmen, es sei denn, das erste in den Ergebnisbaum geschriebene Element ist ein `<html>`-Element. Nicht alle XSLT-Prozessoren befolgen jedoch diese Regel. Sie sollten daher im Zweifelsfall die `<xsl:output>`-Anweisung verwenden.

X(HT)ML als Ausgabeformat

Befolgt ein XSLT-Prozessor die Regel, HTML als Ausgabeformat anzunehmen, wenn das erste in den Ergebnisbaum geschriebene Element ein `<html>`-Element ist, dann müssen Sie eine gewünschte XHTML-Ausgabe über die Angabe des Wertes `xml` für das `method`-Attribut des `<xsl:output>`-Elements erzwingen.

Beispiel XSLT-6: XHTML-Ausgabe erzwingen

```
<xsl:output method="xml" encoding="iso-8859-1"/>
```

Zusätzlich sollten Sie wegen der deutschen Umlaute ein `encoding`-Attribut mit dem Wert `iso-8859-1` angeben.

Text als Ausgabeformat

Ein letzter möglicher Wert des `method`-Attributs des `<xsl:output>`-Elements ist `text`. Damit veranlassen Sie den XSLT-Prozessor, alle Textknoten des Ergebnisbaums auszugeben. Alle anderen Knoten werden ignoriert.

Angewandt auf unsere Literaturliste ergäbe dies folgende Ausgabe:

Beispiel XSL-7: Textausgabe der Literaturliste

```
XML-
LiteraturverzeichnisLiteraturlisteAutorTitelVerlagEr-
scheinungsortErscheinungsjahrPeter MüllerMein erstes XML
BuchAddison-WesleyBonn2000
```

Die `text`-Methode werden Sie v.a. dann verwenden, wenn Sie XML in ein Textformat wie z.B. CSV wandeln müssen.

Template Rules (<xsl:apply-templates>)

Anwenden von mehreren Template Rules

Formulieren wir unser Stylesheet zur Darstellung der Literaturliste etwas um, dann wird deutlicher, dass XSLT eine regelbasierte Sprache ist.

Wir schreiben für jedes Kindelement des <eintrag>-Elements eine eigene Template Rule, welche Sie über das Element <xsl:apply-templates> in das HTML-Gerüst hineinpacken:

Beispiel XSLT-8: <xsl:apply-templates>

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor">
```

```
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

  <xsl:template match="titel">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

  <xsl:template match="verlag">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

  <xsl:template match="erscheinungsort">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

  <xsl:template match="erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ein XSLT-Prozessor beginnt seine Arbeit immer beim Wurzelknoten des Quelldokuments. Findet er eine Template Rule für diesen Knoten, beginnt er mit der Abarbeitung. Findet er keine, traversiert er den Baum weiter. In unserem Beispiel findet er eine Template Rule für das Element `<literaturliste>`. Bei der Abarbeitung trifft er auf die Anweisung `<xsl:apply-templates>`, was ihn dazu veranlasst, seinen Weg umzulenken und *alle* Kindelemente des Elements `<literaturliste>` zu bearbeiten, d.h. im Stylesheet-Baum zu suchen, ob es Template Rules für diese Elemente gibt - falls, ja muss er diese abarbeiten. Man spricht im Fachjargon vom **Instanziieren** einer Template Rule. In unserem Beispiel findet der XSLT-Prozessor eine Template Rule für das `<eintrag>`-Element. Diese schreibt eine Tabellenzeile für den Literatureintrag und weist den XSLT-Prozessor erneut über einen `<xsl:apply-templates>`-Befehl an, die Kinder des gegenwärtigen Knotens abzuarbeiten. Auf diese Weise werden dann die einzelnen Template Rules für Autor, Titel usw. des Literatureintrags instanziiert.

Der Punkt (`.`) als Attributwert der `select`-Attribute der `<xsl:value-of>`-Elemente ist der XPath-Ausdruck für den gegenwärtigen Knoten. Der **gegenwärtige Knoten (*current node*)** ist derjenige, der gerade vom XSLT-Prozessor bearbeitet wird.

Mittels `<xsl:apply-templates>` können Sie den Prozessor somit veranlassen, durch den Quellbaum hinabzusteigen und dabei vorhandene Regeln für die passierten Elemente abzuarbeiten. Sie müssen dem Prozessor dabei nicht - wie in einer klassischen Programmiersprache - sagen, *wie* er das zu tun hat und in welcher Reihenfolge. Sie sagen ihm lediglich über ihre Template Rule, *was* er zu tun hat. Die Anordnung der Template Rules im Stylesheet ist daher auch irrelevant für das Ergebnis.

Zusammenfassen verschiedener Template Rules mit gleichem Template

Die Templates der Template Rules für die Kinder des `<eintrag>`-Elements sind alle gleich. Über den **XPath-Vereinigungsoperator (*Union Operator*)** können Sie zusammengefasst werden:

Beispiel XSLT-9: Vereinigungsoperator

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
```

```
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Eine Template Rule für mehrere Elemente

Mit dem Konstrukt `<xsl:apply-templates>` weisen Sie den XSLT-Prozessor an, *alle* Kinder des gegenwärtigen Knotens abzuarbeiten. Daher können Sie mit demselben Stylesheet, auch eine XML-Datei mit mehreren Literatureinträgen transformieren.

Beispiel XSLT-10: Transformation einer Literaturliste mit mehreren Einträgen

---

Quelldokument

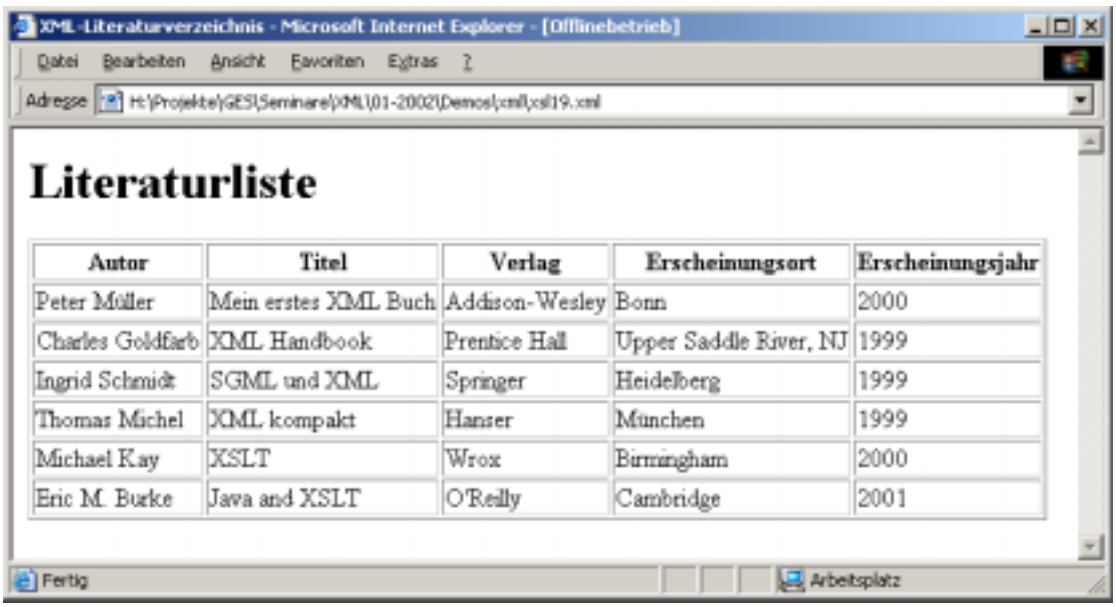
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../xsl/xsl19.xsl"
                 type="text/xsl"?>
<literaturliste>
  <eintrag>
    <autor>Peter Müller</autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Charles Goldfarb</autor>
    <titel>XML Handbook</titel>
    <verlag>Prentice Hall</verlag>
    <erscheinungsort>Upper Saddle Ri-
ver</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
```

```
  </eintrag>
  <eintrag>
    <autor>Ingrid Schmidt</autor>
    <titel>SGML und XML</titel>
    <verlag>Springer</verlag>
    <erscheinungsort>Heidelberg</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Thomas Michel</autor>
    <titel>XML kompakt</titel>
    <verlag>Hanser</verlag>
    <erscheinungsort>München</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Michael Kay</autor>
    <titel>XSLT</titel>
    <verlag>Wrox</verlag>
    <erscheinungsort>Birmingham</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Eric M. Burke</autor>
    <titel>Java and XSLT</titel>
    <verlag>O'Reilly</verlag>
    <erscheinungsort>Cambridge</erscheinungsort>
    <erscheinungsjahr>2001</erscheinungsjahr>
  </eintrag>
</literaturliste>
```

Stylesheet

wie oben

Ansicht im Browser



Gezieltes Ansteuern einer Template Rule

Mittels des `<xsl:apply-templates>`-Elements weisen Sie den XSLT-Prozessor an, auch die untergeordneten Elemente des gegenwärtigen Elements zu bearbeiten. Sie können jedoch eine Template Rule auch direkt ansteuern. Das `<xsl:apply-templates>`-Element kann nämlich - wie das `<xsl:value-of>`-Element - ein `select`-Attribut besitzen. Der Wert dieses Attributs ist ein XPath-Ausdruck, der dem XSLT-Prozessor angibt, für welches Element des Quellbaums er eine Template Rule suchen und abarbeiten soll. Dadurch gewinnen Sie, anders als bei der Verwendung von `<xsl:apply-templates>` ohne `select`-Attribut, mehr Kontrolle über den Transformationsprozess.

Die Literaturliste könnten wir auch mit folgendem Stylesheet transformieren, das die jeweiligen Template Rules direkt ansteuert:

Beispiel XSLT-11: `<xsl:apply-templates>` mit `select`-Attribut

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

  <xsl:template match="/literaturliste">
```

```
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates select="eintrag"/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates select="autor"/>
      <xsl:apply-templates select="titel"/>
      <xsl:apply-templates select="verlag"/>
      <xsl:apply-templates select="erscheinungsort"/>
      <xsl:apply-templates select="erscheinungsjahr"/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                       erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Sie können auf diese Weise auch Elemente filtern, indem Sie für das auszufiltternde Element keine Template Rule ansteuern.

Eingebaute Template Rules

So genannte **eingebaute Template Rules (*built-in Template Rules*)** kommen ins Spiel, wenn der XSLT-Prozessor für einen Knoten im Quelldokument,

für den er laut Stylesheet eine Template Rule instanziieren soll, keine Template Rule findet. Diese eingebauten Regeln definieren ein Standardverhalten des XSLT-Prozessors für verschiedene Arten von Knoten, falls keine passende Template Rule gefunden wird. Solche einheitlichen impliziten Regeln haben den Vorteil, dass Sie im Fall einer fehlenden Template Rule nicht selbst das Verhalten für Elemente oder Inhalte des Quelldokuments definieren müssen.

Die folgende Tabelle listet die eingebauten Template Rules für die Kotentypen in XSLT auf:

Knotentyp	**eingebaute Template Rule**
Wurzelknoten	Aufruf von `<xsl:apply-templates>`, um die Kinder des Wurzelknotens zu bearbeiten
Element	Aufruf von `<xsl:apply-templates>`, um die Kinder des gegenwärtigen Knotens zu bearbeiten
Attribut	Kopieren des Attributwertes in den Ergebnisbaum (als Text)
Text	Kopieren des Textes in den Ergebnisbaum
Kommentar	Tue nichts
Verarbeitungsanweisung	Tue nichts
Namensraum	Tue nichts

Findet der XSLT-Prozessor also keine Regel für den Wurzelknoten, dann wird die `<xsl:apply-templates>`-Anweisung aufgerufen, d.h., es werden alle Kindknoten des Wurzelknotens selektiert und passende Template Rules instanziiert. Dies ist der Grund, weshalb unser Literaturlisten-Beispiel funktioniert, obwohl wir keine Template Rule für den Wurzelknoten definiert haben. Da der Prozessor keine Template Rule für den Wurzelknoten findet, greift die eingebaute Template Rule und es wird die vorhandene Template Rule für das einzige Kindelement des Wurzelknotens, das `<literaturliste>`-Element, instanziiert.

Modifizieren wir erneut unser Literaturlisten-Stylesheet ein wenig, um uns die eingebauten Regeln für Elemente und Texte zu vergegenwärtigen:

Beispiel XSLT-12: Eingebaute Template Rules

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

  <xsl:template match="/">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag | erschei
ungsort | erscheinungsjahr">
    <td>
      <xsl:apply-templates/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Das Stylesheet besitzt nun eine explizite Template Rule für den Wurzelknoten:

Beispiel XSLT-13: Template Rule für Wurzelknoten

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

Diese Regel schreibt wieder das Grundgerüst der HTML-Datei in den Ergebnisbaum und setzt den XSLT-Prozessor über die <xsl:apply-templates>-Anweisung auf die Kindelemente des Wurzelknotens an, also auf das <literaturliste>-Element. Für dieses Element gibt es keine Template Rule in unserem Stylesheet. Daher greift die eingebaute Regel für Elementknoten und der XSLT-Prozessor ruft <xsl:apply-templates> auf, um die Kinder des <literaturliste>-Elements zu selektieren und passende Template Rules zu instanziieren. Für das <eintrag>-Element findet er eine Template Rule, die ihn schließlich (wieder über die <xsl:apply-templates>-Anweisung) zur Template Rule für die Elemente <autor>, <titel> und <verlag>, <erscheinungsort> und <erscheinungsjahr> führt. Hier sehen Sie die nächste Veränderung: Statt der Anweisung <xsl:value-of> finden Sie auch hier, auf der untersten Ebene, die Anweisung <xsl:apply-templates>. In diesem Fall sind die Kinder des gegenwärtigen Knotens Textknoten, für die es keine Template Rule im Stylesheet gibt. Daher greift die eingebaute Template Rule für Textknoten, die lautet: "Kopiere den Textknoten in den Ergebnisbaum". In diesem Fall ist das gleichbedeutend mit einer <xsl:value-of>-Anweisung für den gegenwärtigen Knoten. Wir können somit unser Literaturlisten-Stylesheet auf der Basis eines einzigen Anweisungstyps und den eingebauten Template Rules aufbauen.

Schleifen (<xsl:for-each>)

Sie wissen bereits, wie Sie Elementfolgen in einem Quelldokument über den Einsatz von <xsl:apply-templates> abarbeiten können. Eine andere Möglichkeit ist der Einsatz einer Schleife. In XSLT gibt es ein Schleifenkonstrukt, das durch das Element <xsl:for-each> repräsentiert wird. Formulieren wir unser Literaturlisten-Beispiel so um, dass wir nun eine Schleifenkonstruktion verwenden, um die einzelnen Kinder der <eintrag>-Elemente abzuarbeiten, statt <xsl:apply-templates> aufzurufen und den XSLT-Prozessor anzuweisen, alle Kinder des <eintrag>-Elements abzuarbeiten:

Beispiel XSLT-14: <xsl:for-each>

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:for-each select="eintrag">
            <tr>
              <td>
                <xsl:value-of select="autor"/>
              </td>
              <td>
                <xsl:value-of select="titel"/>
              </td>
              <td>
                <xsl:value-of select="verlag"/>
              </td>
              <td>
                <xsl:value-of select="erscheinungsort"/>
```

```
              </td>
              <td>
                <xsl:value-of
                     select="erscheinungsjahr"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Das <xsl:for-each>-Element besitzt ein select-Attribut, das die gleiche Funktion hat wie das select-Attribut des <xsl:apply-templates>-Elements: eine Knotenmenge auszuwählen, in unserem Beispiel also die Knotenmenge, die aus allen Elementknoten des <eintrag>-Elements besteht. Für jeden dieser Knoten führt der XSLT-Prozessor dann alles aus, was innerhalb des <xsl:for-each>-Elements steht. Für das obige Beispiel legt er daher für jedes <eintrag>-Element eine Tabellenzeile an und schreibt die Inhalte der Elemente <autor>, <titel>, <verlag>, <erscheinungsort> und <erscheinungsjahr> in jeweils eine Tabellenzelle.

Unterschied <xsl:apply-templates> und <xsl:for-each>

Sowohl mit <xsl:apply-templates> als auch mit <xsl:for-each> können Sie den Transformationsprozess steuern. Mit <xsl:apply-templates> weisen Sie den XSLT-Prozessor an, nach Abarbeitungsregeln (Template Rules) für bestimmte Knoten des Quelldokuments zu suchen und diese dann anzuwenden. Dieser Ansatz ist regelbasiert und für jemanden, der von klassischen Programmiersprachen her zu XSLT kommt, zunächst gewöhnungsbedürftig. Mit <xsl:for-each> weisen Sie den XSLT-Prozessor explizit an, welche Knoten als nächste wie zu bearbeiten sind. Sie "ziehen" die ausgewählten Knoten sozusagen in das Template hinein. Deswegen spricht man bei diesem Designmuster auch von **Pull Processing** (engl. *pull* = dt. *ziehen*). Betrachten Sie noch einmal das obige Beispiel, so sehen Sie, dass wir nur eine Template Rule verwendet haben, in die wir mittels <xsl:for-each> alle <eintrag>-Elemente hineinziehen und abarbeiten. Im Gegensatz dazu "drücken" Sie über <xsl:apply-templates> bildlich gesprochen die ausgewählten Knoten hinaus aus dem Template, wo sie von

den jeweils zuständigen Template Rules aufgefangen und abgearbeitet werden. Dieses Designmuster nennt man auch **Push Processing** (engl. *push* = dt. *drücken*).

Wann <xsl:apply-templates>, wann <xsl:for-each>?

Ob Sie <xsl:apply-templates> oder <xsl:for-each> verwenden, um den Transformationsprozess zu steuern, ist aim Grunde eine Geschmacksfrage. Es kann für die Verarbeitung von sich ständig erweiternden XML-Dokumenten von Vorteil sein, den regelbasierten Ansatz über <xsl:apply-templates> zu nutzen. Falls dann Elemente im XML-Dokument hinzugefügt werden, können Sie einfach eine Template Rule für das neue Element schreiben und müssen nicht die Logik der Template Rules für die übergeordneten Elemente ändern. Falls Sie jedoch das Pull Processing über <xsl:for-each> übersichtlicher finden, weil sie beim Lesen des Stylesheets nicht zwischen verschiedenen Template Rules hin- und herspringen müssen, ist dies vielleicht der geeignetere Ansatz für Sie.

Faustregel

- Pull Processing, wenn die Daten vorhersehbar sind
- Push-Processing, wenn die Daten nicht vorhersehbar sind, aber die Struktur bekannt ist (generische Lösung).

Vereinfachte Stylesheets

Sie können den Aufbau eines Stylesheets vereinfachen, indem Sie ein literales Ergebniselement als Wurzelelement des Stylesheets verwenden.

Beispiel XSLT-15: Vereinfachtes Stylesheet

```
<html
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xsl:version="1.0">
  <head>
    <title>XML-Literaturverzeichnis</title>
  </head>
  <body>
    <h1>Literaturliste</h1>
    <table border="1">
      <tr>
        <th>Autor</th>
        <th>Titel</th>
        <th>Verlag</th>
        <th>Erscheinungsort</th>
        <th>Erscheinungsjahr</th>
      </tr>
      <xsl:for-each select="/literaturliste/eintrag">
        <tr>
          <td>
            <xsl:value-of select="autor"/>
          </td>
          <td>
            <xsl:value-of select="titel"/>
          </td>
          <td>
            <xsl:value-of select="verlag"/>
          </td>
          <td>
            <xsl:value-of select="erscheinungsort"/>
          </td>
          <td>
            <xsl:value-of select="erscheinungsjahr"/>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
```

Das Stylesheet ähnelt nun mehr einer HTML-Datei als einem Stylesheet, wie Sie es bisher kennen gelernt haben. Wurzelelement ist das <html>-Element. Die XSLT-Namensraum-Deklaration erfolgt in diesem Wurzelelement wie auch die Angabe des Versionsattributs.

An den entsprechenden Stellen werden über <xsl:value-of> Werte aus dem Quelldokument eingefügt. Diese vereinfachte Syntax bietet der XSLT-Standard an, um HTML-Autoren, die nur geringe XSLT-Kenntnisse besitzen, die Arbeit zu erleichtern. Sie arbeiten auf der gewohnten Struktur einer HTML-Datei und können HTML-Editoren zum Erstellen eines Stylesheets verwenden.

Das obige vereinfachte Stylesheet ist äquivalent zu folgendem Stylesheet:

Beispiel XSLT-16: Äquivalentes Stylesheet zum vorigen, vereinfachten Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:for-each select="eintrag">
            <tr>
              <td>
                <xsl:value-of select="autor"/>
              </td>
              <td>
                <xsl:value-of select="titel"/>
              </td>
              <td>
                <xsl:value-of select="verlag"/>
              </td>
              <td>
                <xsl:value-of select="erscheinungsort"/>
              </td>
```

```
              <td>
                <xsl:value-of
                    select="erscheinungsjahr"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>

  </xsl:template>

</xsl:stylesheet>
```

Bei der vereinfachten Syntax geht der XSLT-Prozessor davon aus, dass alles, was in dem Wurzelelement auftaucht, ein Template für den Wurzelknoten ist.

Geeignet ist diese Variante aber lediglich für einfache Stylesheets, die keine Top-Level-Elemente verwenden und nur aus einer einzigen Template Rule bestehen.

Bedingungen (<xsl:if> und <xsl:choose>)

Es gibt zwei Arten von Bedingungen:

1. <xsl:if> für den bedingten Einschluss eines Templates
2. <xsl:choose> für eine Fallunterscheidung zwischen mehreren Alternativen

<xsl:if> entspricht dem aus allen Programmiersprachen bekannten if. Allerdings gibt es in XSLT nicht die Möglichkeit mit else eine Alternative anzugeben. <xsl:choose> entspricht den case-Konstrukten der Programmiersprachen, muss aber in XSLT bereits verwendet werden, wenn man nur eine Alternative angeben möchte.

<xsl:if>

Das <xsl:if>-Konstrukt besitzt ein Attribut test. Der Wert des Attributs ist ein XPath-Ausdruck, der einen booleschen Wert zurückliefert (true oder false). Ist der Wert true, wird das Template unterhalb des <xsl:if>-Konstrukts instanziiert.

Über <xsl:if> könnten Sie z.B. nur die Titel des Literaturverzeichnis ausgeben lassen, die 1999 erschienen sind.

Beispiel XSLT-17: <xsl:if>

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
```

```
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:for-each select="eintrag">
            <xsl:if test="erscheinungsjahr='1999'">
              <tr>
                <td>
                  <xsl:value-of select="autor"/>
                </td>
                <td>
                  <xsl:value-of select="titel"/>
                </td>
                <td>
                  <xsl:value-of select="verlag"/>
                </td>
                <td>
                  <xsl:value-of
                       select="erscheinungsort"/>
                </td>
                <td>
                  <xsl:value-of
                       select="erscheinungsjahr"/>
                </td>
              </tr>
            </xsl:if>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Ansicht im Browser

Beachten Sie:

Genau genommen wird in unserem Beispiel der Zeichenkettenwert des Elementknotens `<erscheinungsjahr>` mit der Zeichenkette `1999` verglichen. Was ist aber der Zeichenkettenwert eines Elementknotens? Nun, laut XSLT die Zusammensetzung aller Zeichenkettenwerte der Kindknoten. In diesem Fall gibt es nur einen Kindknoten, nämlich einen Textknoten, der den Inhalt des `<erscheinungsjahr>`-Elements repräsentiert. Einleuchtenderweise ist der Zeichenkettenwert eines Textknotens der Text im Quelldokument, den er repräsentiert, in unserem Fall also entweder `1999`, `2000` usw. Sie können den Textknoten auch explizit auswählen, indem Sie schreiben: `<xsl:if test="erscheinungsjahr/text()='1999'">`.

Liefert ein XPath-Ausdruck, der als Wert des `test`-Attributs auftaucht, keinen booleschen Wert zurück, so wird der zurückgelieferte Wert implizit konvertiert. Wählen Sie mit dem XPath-Ausdruck im `test`-Attribut des `<xsl:if>`-Elements eine Knotenmenge aus, dann ist das Ergebnis eben genau diese Knotenmenge und kein boolescher Wert. D.h., es muss eine Konvertierung stattfinden. Die Regel dabei lautet: Ist die Knotenmenge leer, so liefert die Konvertierung `false`, enthält die Knotenmenge mindestens ein Element, so ist der Wert `true`. Damit können Sie hervorragend Templates in Abhängigkeit von der Existenz anderer Elemente instanziieren. Zum Beispiel könnten Sie innerhalb der Schleife über die `<eintrag>`-Elemente der Literaturliste folgendermaßen prüfen, ob auch jeder Eintrag ein `<erscheinungsjahr>`-Element besitzt:

Beispiel XSLT-18: Prüfung, ob Element im Quelldokument vorhanden ist

```
<xsl:if test="erscheinungsjahr">
  <!-- Template -->
</xsl:if>
```

<xsl:choose>

Die Struktur des <xsl:choose>-Elements sieht folgendermaßen aus:

```
<xsl:choose>
  <!-- Inhalt: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>

<xsl:when test="boolescher-ausdruck">
  <!-- Inhalt = Template -->
</xsl:when>
<xsl:otherwise>
  <!-- Inhalt = Template -->
</xsl:otherwise>
```

Das <xsl:choose>-Konstrukt beinhaltet eine oder mehrere <xsl:when>-Bedingungen und einen optionalen Standardzweig <xsl:otherwise>. Die <xsl:when>-Elemente haben ein test-Attribut, welches dieselbe Funktion wie bei <xsl:if> erfüllt. Gibt es mehrere <xsl:when>-Elemente innerhalb eines <xsl:choose>-Konstrukts, werden die einzelnen Alternativen nacheinander getestet. Bei der ersten Bedingung, die true ergibt, - und nur bei dieser - wird das eingeschlossene Template instanziiert. Sollten weitere Bedingungen true ergeben, werden diese nicht mehr berücksichtigt.

Sollten keine der Bedingungen erfüllt sein, so wird das Template des <xsl:otherwise>-Konstrukts instanziiert - falls ein solches vorhanden ist.

Über <xsl:choose> könnten Sie z.B. alle Titel, deren Erscheinungsjahr 1999 ist grün, alle Titel, deren Erscheinungsjahr 2000 ist, blau und alle anderen Titel rot färben:

Beispiel XSLT-19: <xsl:choose>

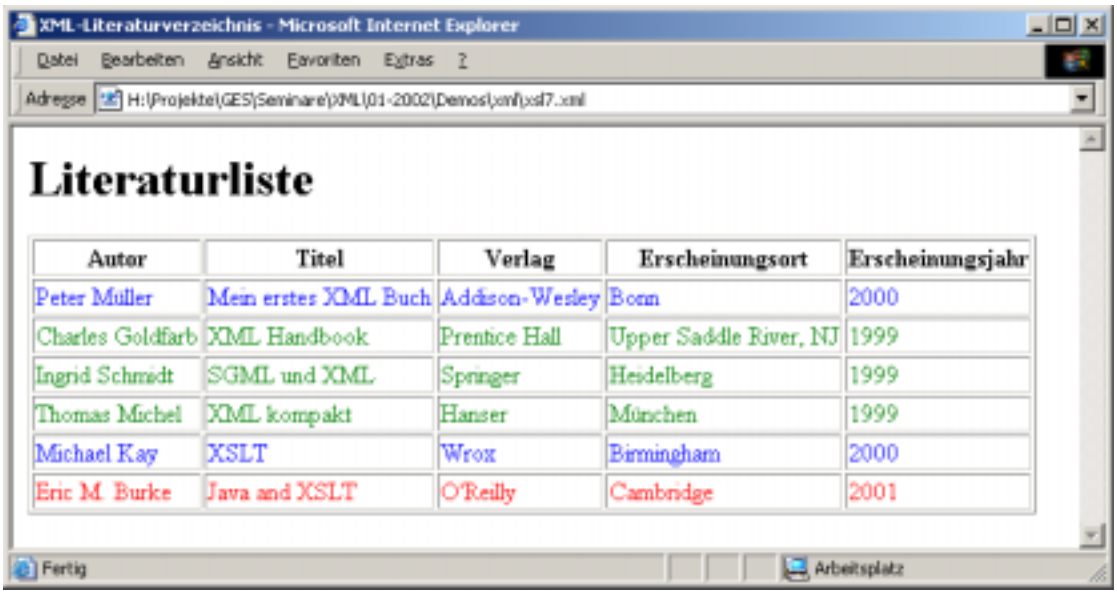
Stylesheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
     version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:for-each select="eintrag">
            <xsl:choose>
              <xsl:when test="erscheinungsjahr='1999'">
                <tr>
                  <td>
                    <font color="green">
                      <xsl:value-of select="autor"/>
                    </font>
                  </td>
                  <td>
                    <font color="green">
                      <xsl:value-of select="titel"/>
                    </font>
                  </td>
                  <td>
                    <font color="green">
                      <xsl:value-of select="verlag"/>
                    </font>
                  </td>
                  <td>
                    <font color="green">
                      <xsl:value-of
                          select="erscheinungsort"/>
                    </font>
                  </td>
                  <td>
```

```
        <font color="green">
          <xsl:value-of
              select="erscheinungsjahr"/>
        </font>
      </td>
    </tr>
</xsl:when>
<xsl:when test="erscheinungsjahr='2000'">
  <tr>
    <td>
      <font color="blue">
        <xsl:value-of select="autor"/>
      </font>
    </td>
    <td>
      <font color="blue">
        <xsl:value-of select="titel"/>
      </font>
    </td>
    <td>
      <font color="blue">
        <xsl:value-of select="verlag"/>
      </font>
    </td>
    <td>
      <font color="blue">
        <xsl:value-of
            select="erscheinungsort"/>
      </font>
    </td>
    <td>
      <font color="blue">
        <xsl:value-of
            select="erscheinungsjahr"/>
      </font>
    </td>
  </tr>
</xsl:when>
<xsl:otherwise>
  <tr>
    <td>
      <font color="red">
        <xsl:value-of select="autor"/>
      </font>
    </td>
    <td>
      <font color="red">
        <xsl:value-of select="titel"/>
      </font>
    </td>
```

```
                    <td>
                      <font color="red">
                        <xsl:value-of select="verlag"/>
                      </font>
                    </td>
                    <td>
                      <font color="red">
                        <xsl:value-of
                            select="erscheinungsort"/>
                      </font>
                    </td>
                    <td>
                      <font color="red">
                        <xsl:value-of
                            select="erscheinungsjahr"/>
                      </font>
                    </td>
                  </tr>
                </xsl:otherwise>
              </xsl:choose>
            </xsl:for-each>
          </table>
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

Ansicht im Browser



Dieses Vorgehen entspricht im Wesentlichen den klassischen `case`-Konstrukten aus höheren Programmiersprachen. Das `if-else`-Konstrukt kennt XSLT nicht. Es kann aber über `<xsl:choose>` simuliert werden. Dann wird genau ein `<xsl:when>`-Zweig benötigt und ein `<xsl:otherwise>`-Zweig.

Beachten Sie:

Ergibt der `test`-Ausdruck für kein `<xsl:when>`-Element `true`, so wird das Template des `<xsl:otherwise>`-Elements instanziiert. Ist kein `<xsl:otherwise>`-Element vorhanden, so passiert nichts.

Praxistipp:

Denselben Effekt wie mit `<xsl:if>` erzielen Sie, indem Sie eine `<xsl:choose>`-Anweisung verwenden, die nur *ein* `<xsl:when>`-Element enthält und kein `<xsl:otherwise>`-Element. Diese Alternative sollten Sie dann verwenden, wenn Sie nicht sicher sind, ob Sie später eine weitere Alternative benötigen.

Einbinden von Multimedia-Elementen

Laut XML-Spezifikation sollten Multimedia-Elemente über externe, nicht-analysierte Entities in XML eingebunden werden. Dazu benötigen Sie jedoch immer eine DTD. Außerdem verstehen die Browser dieses Verfahren noch nicht.

Mit XSLT steht ihnen ein anderer, intuitiverer Weg offen. Folgendes Listing bindet ein Bild von Tim Berners-Lee, dem "Erfinder" des WWW, zu Beginn des Literaturverzeichnisses ein.

Beispiel XSLT-20: Einbinden von Bildern

Erweitern wir unsere XML-Literaturliste um Angaben zu dem Bild.

Quelldokument

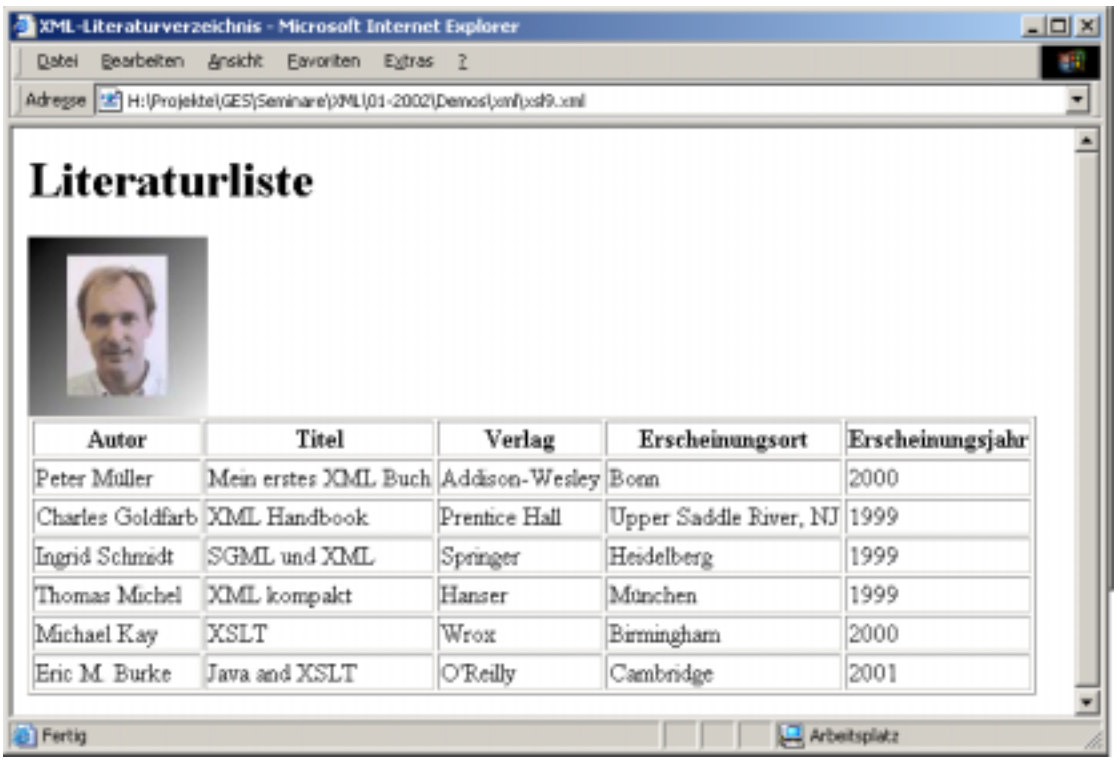
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../xsl/xsl7.xsl" ty-
pe="text/xsl"?>
<literaturliste>
  <bild quelle="..img/tim.jpg"/>
  <eintrag>
    <autor>Peter Müller</autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>

  <!-- Weitere Einträge -->

  </literaturliste>
```

Möchten Sie diese Angaben nutzen, um das Bild in Ihrem HTML-Ergebnisdokument anzeigen zu lassen, so müssen Sie ein `<img>`-Element erzeugen, dessen `src`-Attribut als Wert den Wert des `<quelle>`-Attributs des `<bild>`-Elements besitzt. Voraussetzung ist natürlich, dass ein Bild mit dem entsprechenden Namen (hier `tim.jpg`) im angegebenen Verzeichnis liegt.

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <img src="{bild/@quelle}" border="0"/>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                  erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Erläuterungen

Zunächst könnte es scheinen, dass Sie bereits ein Mittel kennen, um dem `src`-Attribut eines `<img>`-Elements einen Wert zuzuweisen, nämlich das `<xsl:value-of>`-Element, mit dessen Hilfe Sie Elementinhalte in den Ergebnisbaum kopieren können. Somit müssten Sie das Stylesheet einfach an der Stelle, an der Sie das Bild einfügen möchten, um die Zeile

```
<img src="<xsl:value-of select='bild/@quelle'/>/>
```

erweitern.

Versuchen Sie es auf diese Weise, werden Sie feststellen, dass der XML-Parser mit einer Fehlermeldung abbricht. Und zwar wird er sich beschweren, dass Sie Markup-Symbole wie < und > innerhalb eines Attributwertes verwenden, was nicht zulässig ist. Sie können diese Markup-Symbole zwar über vordefinierte Entity-Referenzen maskieren, indem Sie < als `&lt;` und > als `&gt;` schreiben. Das führt allerdings zu recht unleserlichen Ausdrücken und ist auch recht umständlich in der Handhabung. Die Lösung in XSLT sieht anders und wesentlich einfacher aus.

XSLT bietet Ihnen die so genannten **Attribute Value Templates**, um Attribute zu generieren, deren Wert sich zur Laufzeit erst berechnet, statt einen festen

Wert anzunehmen. Um ein Attribut in dieser Weise zu parametrisieren, müssen Sie lediglich geschweifte Klammern um die Wertangabe setzen:

```
<img src="{bild/@quelle}"/>
```

Durch die geschweiften Klammern wird dem XSLT-Prozessor angezeigt, dass er den Attributwert nicht eins zu eins in den Ergebnisbaum schreiben soll, sondern den XPath-Ausdruck auswerten und das Ergebnis, d.h. in unserem Fall den Wert des `quelle`-Attributs des `<bild>`-Elements, einsetzen soll.

Auf dieselbe Weise sind Sie nun in der Lage, Sounddateien, Videodateien und andere Multimediaobjekte in HTML einzubinden.

Einbinden von Skripten

Die XSLT-Empfehlung in der aktuellen Version sieht kein eigenes Element zum Einbinden von Skripten vor. Ein solches war mal in ersten Entwürfen geplant und ebenso im Arbeitsentwurf für die Version 1.1 von XSLT.

Sie können Skripte jedoch über das HTML-Tag `<script>` in ihre HTML-Ergebnisdokumente integrieren. Folgendes Beispiel gibt unter der Literaturliste mittels JavaScript das aktuelle Datum aus:

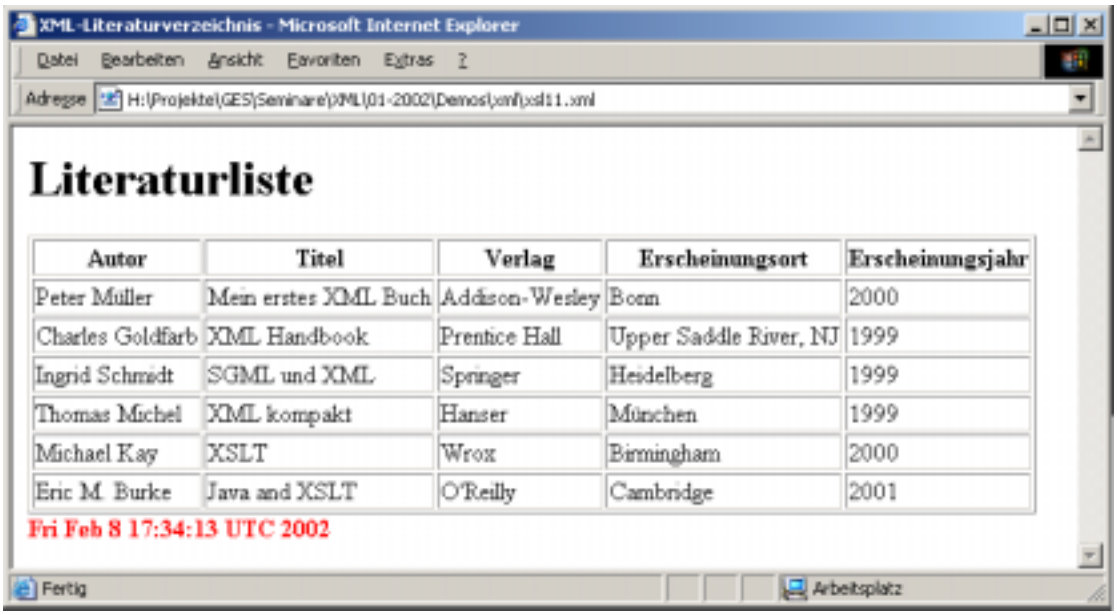
Beispiel XSLT-21: Einbinden von Skripten

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
        <script language="JavaScript">
          <![CDATA[
               var datum = new Date();
               datum = datum.toString();
               window.document.write(datum);
          ]]>
        </script>
      </body>
```

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

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Praxistipp:

Der Skript-Code innerhalb des `<script>`-Tags sollte immer in einem CDA-TA-Abschnitt stehen, damit er nicht als XML behandelt wird.

Einbinden von Cascading Style Sheets

Cascading Style-Sheets (CSS) werden nach dem gleichen Prinzip eingebunden wie Skripte. Sie benutzen lediglich das `<style>`-Tag, um dort ihre CSS-Klassen zu definieren. So ließen sich etwa die Farbgebungen ohne das vom W3C als *deprecated (deprecate = missbilligen)* gekennzeichnete HTML-Tag `<font>` durchführen oder auch wie im folgenden Beispiel Arial oder Helvetica als Standardschriftart vorgeben.

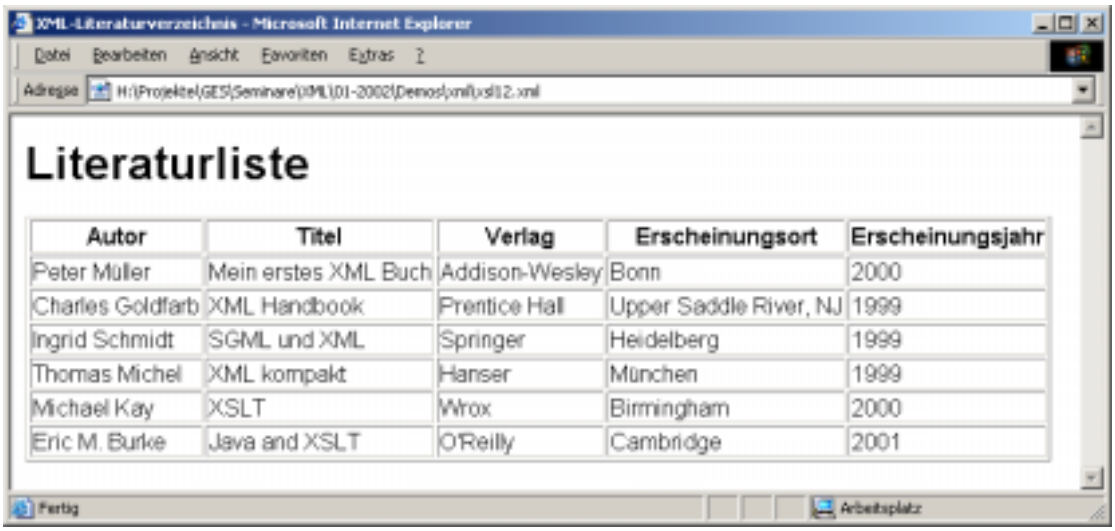
Beispiel XSLT-22: Einbinden von CSS

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
      <style>
        <![CDATA[
             body { font-family:Arial,Helvetica; }
        ]]>
      </style>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
```

```
  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Sortierung (<xsl:sort>)

Mit XSLT können Sie ihre Quelldokumente auch sortiert ausgeben lassen. Dafür gibt es das Konstrukt <xsl:sort>.

Beachten Sie:
Das <xsl:sort>-Element darf nur als Kindelement von <xsl:apply-templates> oder <xsl:for-each> verwendet werden. Nur dort macht es ja auch Sinn.

Sortierkriterium festlegen

Über das select-Attribut des <xsl:sort>-Elements legen Sie fest, nach welchem Element des Quelldokuments Sie sortieren möchten.

Sortieren wir unsere Literaturliste nach dem Erscheinungsjahr der Titel:

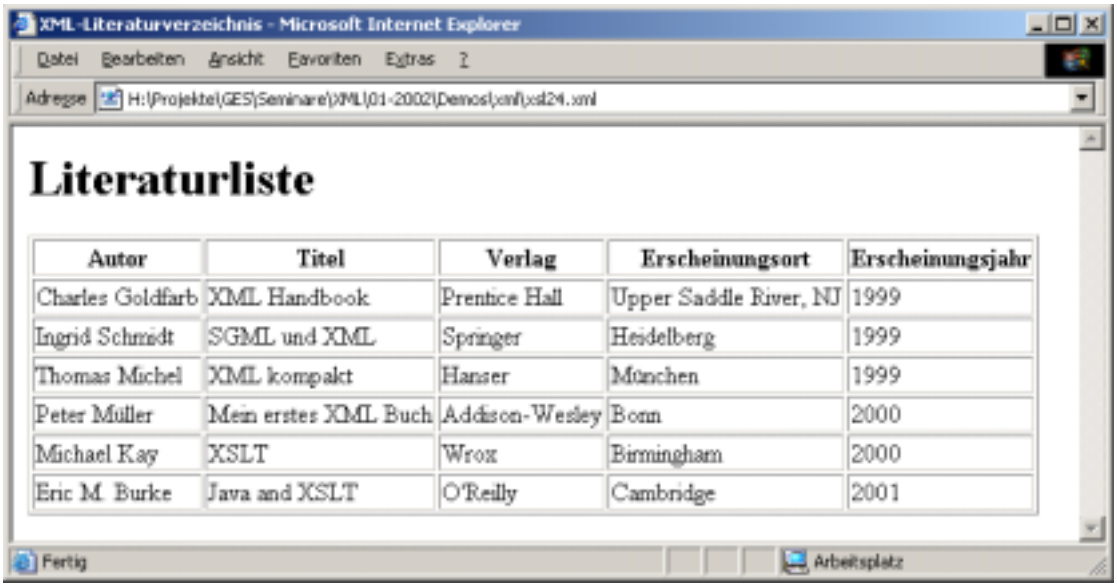
Beispiel XSLT-23: Sortieren mit <xsl:sort>

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
```

```
          </tr>
          <xsl:apply-templates>
            <xsl:sort select="erscheinungsjahr"/>
          </xsl:apply-templates>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Sortierrichtung festlegen

Die Sortierrichtung legen Sie über das Attribut `order` fest, das folgende Werte annehmen kann:

Wert	**Bedeutung**
`ascending`	aufsteigend (Standardwert!)
`descending`	absteigend

Möchten Sie die Literaturliste also *absteigend* nach Erscheinungsjahr sortieren, müssen Sie den fett markierten Teil des obigen Beispiels wie folgt ändern:

Beispiel XSLT-24: Sortierrichtung festlegen

```
<xsl:apply-templates>
  <xsl:sort select="erscheinungsjahr"
            order="descending"/>
</xsl:apply-templates>
```

Sortieren nach Datentyp

Ein weiteres wichtiges (optionales) Attribut des `<xsl:sort>`-Elements ist das Attribut `data-type`. Es legt fest, ob alphabetisch (Attributwert `text`) oder numerisch (Attributwert `number`) sortiert werden soll. Die Standardeinstellung ist `text`.

Den Unterschied zeigt das folgende Beispiel:

Beispiel XSLT-25: Sortieren nach Datentyp

Gegeben sei folgende unsortierte Mitarbeiterliste:

```
<mitarbeiterliste>
  <mitarbeiter id="7"/>
  <mitarbeiter id="1"/>
  <mitarbeiter id="13"/>
  <mitarbeiter id="12"/>
</mitarbeiterliste>
```

Sortieren Sie diese Liste nach der Mitarbeiter-ID (Attribut `id`) mit `<xsl:sort select="@id" data-type="text">` erhalten Sie folgendes (vermutlich unerwünschtes) Ergebnis:

```
<mitarbeiterliste>
  <mitarbeiter id="1"/>
  <mitarbeiter id="12"/>
  <mitarbeiter id="13"/>
  <mitarbeiter id="7"/>
</mitarbeiterliste>
```

Mit `<xsl:sort select="@id" data-type="number">` erhalten Sie die korrekte numerische Sortierung:

```
<mitarbeiterliste>
  <mitarbeiter id="1"/>
  <mitarbeiter id="7"/>
  <mitarbeiter id="12"/>
  <mitarbeiter id="13"/>
</mitarbeiterliste>
```

Sortieren nach mehreren Kriterien

Sie können auch mehrere `<xsl:sort>`-Elemente hintereinander setzen und auf diese Weise mehrere Suchkriterien verwenden.

Die Reihenfolge der `<xsl:sort>`-Elemente spielt dabei eine wichtige Rolle. Das erste `<xsl:sort>`-Element stellt das primäre Suchkriterium dar, das zweite das sekundäre Kriterium usw.

Wollen wir also unsere Literaturliste so sortieren, dass Bücher desselben Erscheinungsjahrs wiederum alphabetisch nach dem Verlag sortiert sind, so müssen wir unser Stylesheet an der fettgedruckten Stelle erneut abwandeln:

Beispiel XSLT-26: Sortieren nach mehreren Kriterien

```
<xsl:apply-templates>
  <xsl:sort select="erscheinungsjahr">
  <xsl:sort select="verlag"/>
</xsl:apply-templates>
```

Zählen und Nummerieren

Mittels einiger Funktionen, die XPath und XSLT zur Verfügung stellen, können Sie Elemente mit Stylesheets durchzählen und nummerieren.

count()

Die count()-Funktion erhält als Parameter eine Knotenmenge und gibt als Wert die Anzahl der Knoten der Knotenmenge zurück. Mittels dieser XPath-Funktion können wir z.B. vor unserer Tabelle ausgeben, wie viele Einträge unsere Literaturliste enthält.

Beispiel XSLT-27: Die count()-Funktion

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <p>
          Anzahl der Einträge: <xsl:value-of
                          select="count(eintrag)"/>
        </p>
        <table border="1">
          <tr>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
```

```
  <xsl:template match="eintrag">
    <tr>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

position()

Die XPath-Funktion `position()` ermittelt die Position des Kontextknotens. Die Funktion liefert eine Zahl zurück, deren Wert besagt, der wievielte Knoten (gezählt in Dokumentordnung) der gegenwärtige Knoten in der ausgewählten Knotenmenge ist. Sie erwartet keinerlei Argumente.

So lassen sich z.B. die Einträge unseres Literaturverzeichnisses mittels der `position()`-Funktion mit einer laufenden Nummer versehen:

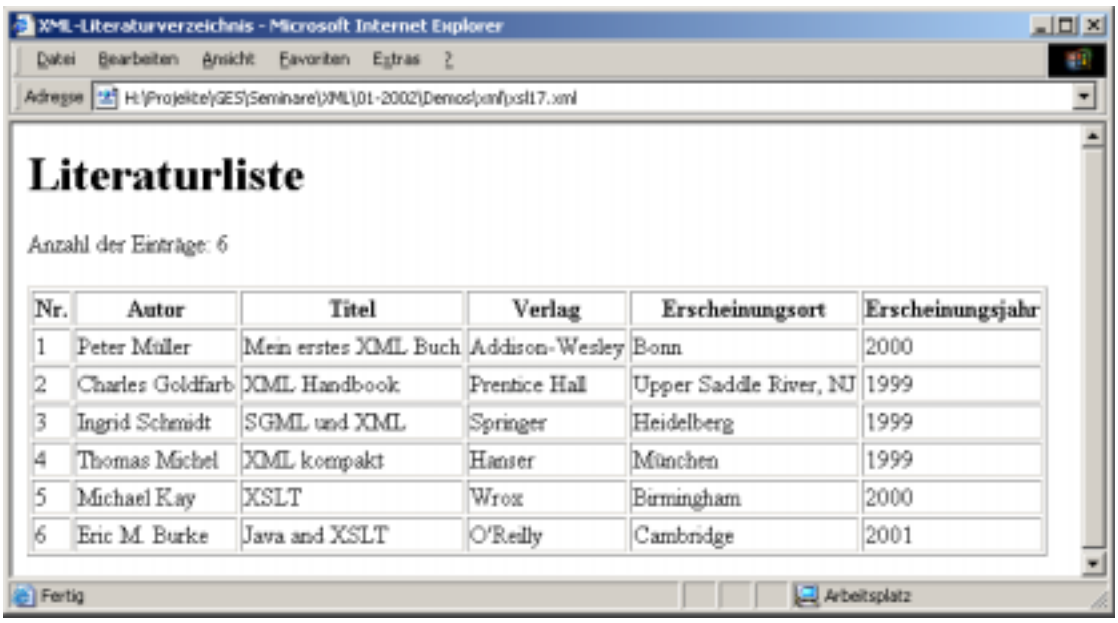
Beispiel XSLT-28: Nummerierung über die `position()`-Funktion

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <p>
          Anzahl der Einträge: <xsl:value-of
```

```
                              select="count(eintrag)"/>
        </p>
        <table border="1">
          <tr>
            <th>Nr.</th>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <td>
        <xsl:value-of select="position()"/>
      </td>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                       erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



<xsl:number>

XSLT stellt ein komplexes Konstrukt für die Nummerierung zur Verfügung: das `<xsl:number>`-Element.

Die einfachste Form der Nummerierung über `<xsl:number>` besteht in der Verwendung dieses Elements ohne Attribute. In unserem Beispiel hat dies den gleichen Effekt wie der Gebrauch der `position()`-Funktion.

Beispiel XSLT-29: einfachster Gebrauch von <xsl:number>

Statt

```
<xsl:value-of select="position()/>
```

könnten wir ebenso schreiben:

```
<xsl:number/>
```

Beachten Sie:
Sie sollten für diese einfache Form der Nummerierung die `position()`-Funktion verwenden, da sie wesentlich schneller ist.

Das `<xsl:number>`-Element kann Attribute besitzen, über die Sie die Art der Nummerierung steuern können. Die Möglichkeiten, die sich Ihnen damit bieten, sind vielfältig, gehören aber zu den komplexesten Dingen bei XSLT. Die Vorstellung aller Nummerierungsmöglichkeiten würde den Rahmen dieser Einführung sprengen. Einige einfache, jedoch sehr effektive Anwendungsmöglichkeiten möchte ich Ihnen aber nicht vorenthalten.

Beachten Sie:
Das `<xsl:number>`-Element ist immer leer und kann nur in Templates genutzt werden.

Nummernformat festlegen

Über das Attribut `format` können Sie die Formatierung der Nummern beeinflussen, die ausgegeben werden sollen. Die Tabelle listet die möglichen Angaben auf.

Mögliche Werte des `format`-Attributs

`format`-Attributwert	**Ausgabefolge**
`1`	`1,2,3,4 ...` (Standardeinstellung)
`01`	`01,02,03 ... 10,11,12 ...`
`a`	`a,b,c ... x,y,z,aa,ab,ac ...`
`A`	`A,B,C ... X,Y,Z,AA,AB,AC ...`
`i`	`i,ii,iii,iv ... x,xi,xii ...`
`I`	`I, II, III,IV ... X,XI,XII ...`

Um unsere Literaturliste also römisch zu nummerieren, müssen wir an entsprechender Stelle Folgendes notieren:

Beispiel XSLT-30: römische Nummerierung für Literaturliste

```
<xsl:number format="I"/>
```

An die Zahlen können Sie zudem ein Zeichen anhängen, z.B. einen Punkt, um eine nummerierte Liste zu erstellen (1., 2., 3. ...):

Beispiel XSLT-31: Zeichen an Nummernformat anfügen

```
<xsl:number format="1."/>
```

Attribute und Elemente erzeugen

<xsl:attribute>

Attribute Value Templates sind die einfachste und schnellste Möglichkeit, um Attribute mit Werten im Ergebnisbaum zu erzeugen. XSLT bietet Ihnen zusätzlich eine andere, etwas wortreichere Variante: die Verwendung des <xsl:attribute>-Elements.

Beispiel XSLT-32: <xsl:attribute> versus Attribute Value Templates

---

Das src-Attribut für ein <img>-Element könnten Sie sowohl über Attribute Value Templates

```
<img src="{bild/@quelle}"/>
```

als auch über das <xsl:attribute>-Element erzeugen

```
<img>
  <xsl:attribute name="src">
    <xsl:value-of select="bild/@quelle"/>
  </xsl:attribute>
</img>
```

Über das Pflichtattribut name des <xsl:attribute>-Elements legen Sie den Namen des zu erzeugenden Attributs fest. Der Inhalt des <xsl:attribute>-Elements legt den Wert für das erzeugte Attribut fest. In unserem Fall wird über <xsl:value-of> der Wert des Attributs quelle ausgewählt.

Der Einsatz von <xsl:attribute> ist allerdings nur dann sinnvoll, wenn Sie Ihr Ziel nicht auch über Attribute Value Templates erreichen können. Dies ist v.a. dann der Fall, wenn Sie es von einer Bedingung abhängig machen möchten, ob das Attribut erzeugt wird oder nicht.

Beispiel XSLT-33: <xsl:attribute>

Erweitern wir die XML-Literaturliste um die Information, ob ein Buch auch gelesen wurde oder nicht, indem wir ein leeres Element <gelesen/> zu denjenigen <eintrag>-Elementen hinzufügen, die wir gelesen haben:

Quelldokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../xsl/xsl28.xsl"
                 type="text/xsl"?>
<literaturliste>
  <eintrag>
    <autor>Peter Müller</autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
    <gelesen/>
  </eintrag>
  <eintrag>
    <autor>Charles Goldfarb</autor>
    <titel>XML Handbook</titel>
    <verlag>Prentice Hall</verlag>
    <erscheinungsort>Upper Saddle Ri-
ver</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Ingrid Schmidt</autor>
    <titel>SGML und XML</titel>
    <verlag>Springer</verlag>
    <erscheinungsort>Heidelberg</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
    <gelesen/>
  </eintrag>
  <eintrag>
    <autor>Thomas Michel</autor>
    <titel>XML kompakt</titel>
    <verlag>Hanser</verlag>
    <erscheinungsort>München</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Michael Kay</autor>
    <titel>XSLT</titel>
    <verlag>Wrox</verlag>
    <erscheinungsort>Birmingham</erscheinungsort>
```

```
    <erscheinungsjahr>2000</erscheinungsjahr>
    <gelesen/>
  </eintrag>
  <eintrag>
    <autor>Eric M. Burke</autor>
    <titel>Java and XSLT</titel>
    <verlag>O'Reilly</verlag>
    <erscheinungsort>Cambridge</erscheinungsort>
    <erscheinungsjahr>2001</erscheinungsjahr>
  </eintrag>
</literaturliste>
```

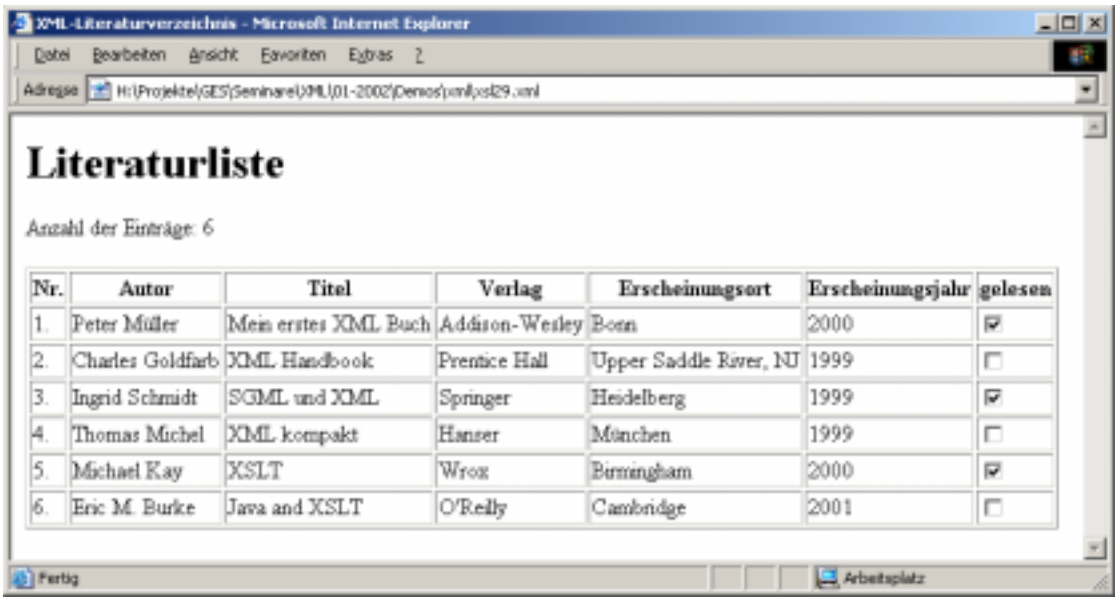
Es soll nun eine zusätzliche Spalte mit der Überschrift "gelesen" an die Tabelle angehängt werden. Inhalt der einzelnen Tabellenzellen ist jeweils eine Checkbox, die angehackt ist, wenn das Buch gelesen wurde.

Stylesheet

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

  <xsl:template match="/literaturliste">
    <html>
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <p>
          Anzahl der Einträge:
          <xsl:value-of select="count(eintrag)"/>
        </p>
        <table border="1">
          <tr>
            <th>Nr.</th>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>
            <th>gelesen</th>
          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
```

```
  <xsl:template match="eintrag">
    <tr>
      <td>
        <xsl:number format="1."/>
      </td>
      <xsl:apply-templates/>
      <td>
        <input type="checkbox">
          <xsl:if test="gelesen">
            <xsl:attribute
name="checked">checked</xsl:attribute>
          </xsl:if>
        </input>
      </td>
    </tr>
  </xsl:template>

  <xsl:template match="autor | titel | verlag |
                 erscheinungsort | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Erläuterung

Über `<xsl:if>` testen Sie, ob das gegenwärtig vom XSLT-Prozessor bearbeitete `<eintrag>`-Element ein Kindelement `<gelesen>` hat. Wenn ja, erzeugen Sie für das literale Ergebniselement `<input>` zusätzlich ein Attribut namens `checked` mit dem Wert `checked`, was dazu führt, dass die Checkbox angehackt wird. Diese Bedingung könnten Sie nicht in einem Attribute Value Template abfragen, da Sie ja keine Elemente mit Attributen verschachteln dürfen.

Praxistipp:

Wenn Sie `<xsl:attribute>` verwenden, sollten Sie darauf achten, dass das gesamte Element mit Inhalt eine Zeile bildet, da in Attributwerten keine Zeilenumbrüche erlaubt sind.

`<xsl:element>`

Sie können mit XSLT nicht nur Attribute, sondern auch Elemente erzeugen. Zu diesem Zweck stellt XSLT das Element `<xsl:element>` zur Verfügung.

Beispiel XSLT-34: literale Ergebniselemente versus `<xsl:element>`

Die Verwendung dieses Elements innerhalb eines Templates hat denselben Effekt wie die direkte Notation eines literalen Ergebniselements:

```
<xsl:template match="/">
  <xsl:element name="br"/>
</xsl:template>
```

ist also äquvalent zu:

```
<xsl:template match="/">
  <br/>
</xsl:template>
```

Das Konstrukt <xsl:element> benötigen Sie immer dann, wenn Sie den Namen des zu erzeugenden Elements zur Laufzeit des Stylesheets erst festlegen wollen.

Der klassische Anwendungsfall für <xsl:element> ist die Umwandlung einer attributlastigen Struktur in eine elementlastige Struktur:

Beispiel XSL-35: <xsl:element>

Unsere Literaturliste in attributlastiger Form soll in die bereits bekannte elementlastige Form umgewandelt werden.

Quelldokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<literaturliste>
  <eintrag autor="Peter Müller"
           titel="Mein erstes XML-Buch"
           verlag="Addison-Wesley"
           erscheinungsort="Bonn"
           erscheinungsjahr="2000"/>
  <eintrag autor="Charles Goldfarb"
           titel="XML Handbook"
           verlag="Prentice Hall"
           erscheinungsort="Upper Saddle River"
           erscheinungsjahr="1999"/>
  <eintrag autor="Ingrid Schmidt"
           titel="SGML und XML"
           verlag="Springer"
           erscheinungsort="Heidelberg"
           erscheinungsjahr="1999"/>
  <eintrag autor="Thomas Michel"
           titel="XML kompakt"
           verlag="Hanser"
           erscheinungsort="München"
           erscheinungsjahr="1999"/>
  <eintrag autor="Michael Kay"
           titel="XSLT"
           verlag="Wrox"
           erscheinungsort="Birmingham"
           erscheinungsjahr="2000"/>
  <eintrag autor="Eric M. Burke"
           titel="Java and XSLT"
           verlag="O'Reilly"
           erscheinungsort="Cambridge"
           erscheinungsjahr="2001"/>
</literaturliste>
```

Stylesheet

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

  <xsl:output indent="yes" encoding="iso-8859-1"/>

  <xsl:template match="/">
    <literaturliste>
      <xsl:apply-templates/>
    </literaturliste>
  </xsl:template>

  <xsl:template match="eintrag">
    <eintrag>
      <xsl:for-each select="@*">
        <xsl:element name="{name()}">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
    </eintrag>
  </xsl:template>

</xsl:stylesheet>
```

Ergebnisdokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../xsl/xsl28.xsl"
                 type="text/xsl"?>
<literaturliste>
  <eintrag>
    <autor>Peter Müller</autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Charles Goldfarb</autor>
    <titel>XML Handbook</titel>
    <verlag>Prentice Hall</verlag>
    <erscheinungsort>Upper Saddle Ri-
ver</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Ingrid Schmidt</autor>
```

```
    <titel>SGML und XML</titel>
    <verlag>Springer</verlag>
    <erscheinungsort>Heidelberg</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Thomas Michel</autor>
    <titel>XML kompakt</titel>
    <verlag>Hanser</verlag>
    <erscheinungsort>München</erscheinungsort>
    <erscheinungsjahr>1999</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Michael Kay</autor>
    <titel>XSLT</titel>
    <verlag>Wrox</verlag>
    <erscheinungsort>Birmingham</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>
  <eintrag>
    <autor>Eric M. Burke</autor>
    <titel>Java and XSLT</titel>
    <verlag>O'Reilly</verlag>
    <erscheinungsort>Cambridge</erscheinungsort>
    <erscheinungsjahr>2001</erscheinungsjahr>
  </eintrag>
</literaturliste>
```

Erläuterung

Das `<xsl:output>`-Element dient dazu, die richtige Kodierung für die Ausgabe zu setzen und Einrückungen zu erzwingen, damit die Ausgabe lesbarer wird. Es folgt ein Template für den Wurzelknoten, welches das Wurzelelement `<literaturliste>` erzeugt und über `<xsl:apply-templates>` die Abarbeitung der `<eintrag>`-Elemente anstößt. Die Template Rule für die `<eintrag>`-Elemente erzeugt über ein literales Ergebniselement ein `<eintrag>`-Element und lässt innerhalb dieses Elements eine Schleife über alle Attribute des aktuellen `<eintrag>`-Elements laufen. Dabei wird ein Element für jedes dieser Attribute mit dem Namen des jeweiligen Attributs erzeugt. Als Inhalt der erzeugten Elemente wird der Attributwert des entsprechenden Attributs über `<xsl:value-of>` in den Ergebnisbaum geschrieben. Zur Erzeugung dieser Attribute müssen Sie `<xsl:element>` verwenden, da die Namen ja nicht im voraus feststehen, sondern zur Laufzeit des Stylesheets ermittelt werden. Die Festlegung der Namen erfolgt über eine so genannte *XPath-Funktion* im `name`-Attribut des `<xsl:element>`-Konstrukts. **Die Funktion `name()`** liefert den Namen des aktuellen Knotens als Rückgabewert.

Praxistipp: <xsl:processing-instruction>

Im vorigen Beispiel handelt es sich um eine XML-zu-XML-Transformation. Das Ergebnis dieser Transformation ist eine unformatierte XML-Datei. Sie können jedoch zusätzlich eine Referenz auf ein Stylesheet erzeugen, mit dem dann das Ergebnisdokument formatiert werden soll. Zur Erzeugung einer solchen Verarbeitungsanweisung benötigen Sie das Element <xsl:processing-instruction>, das eine Verarbeitungsanweiung im Ergebnisbaum erzeugt.

Beispiel XSLT-36: <xsl:processing-instruction>

```
<xsl:processing-instruction
    name="xml-stylesheet">href="../xsl/xsl29.xsl"
type="text/xsl"</xsl:processing-instruction>
```

Beachten Sie:

Fügen Sie keinen Leerraum in den Inhalt des <xsl:processing-instruction>-Elements ein!

Mit Hilfe dieses XSLT-Elements können Sie somit eine erste Transformation zur Filterung, Sortierung usw. auf dem Server durchführen und die Transformation zur Formatierung auf dem Client erledigen lassen. Dazu können Sie natürlich auch CSS nutzen.

Links in XML

Exkurs: XLink

Mit XLink steht ein Standard des W3C zur Verfügung, der es ermöglicht, Links in XML-Dokumente zu integrieren. Die Spezifikation unterscheidet zwischen **einfachen** und **erweiterten Links**. Die einfachen Links entsprechen den von HTML bekannten Hyperlinks.

Erweiterte Links bieten über diese in HTML gebotenen Möglichkeiten hinaus vielfache Linking-Möglichkeiten an. Hier wollen wir uns zunächst auf die einfachen Links beschränken.

Beispiel XLINK-1: HTML-Link

```
<a href="http://www.thomas-mann.de">Thomas Mann</a>
```

Beispiel XLINK-2: XML-Link

```
<autor xlink:type="simple"
       xlink:href="http://www.thomas-mann.de">
  Thomas Mann
</autor>
```

In XML machen Sie ein Element zu einem (einfachen) Link, indem Sie ihm das Attribut `xlink:type` mit dem Wert `simple` und das Attribut `xlink:href` mit der Angabe des Verweisziels (URL) als Wert hinzufügen. Die Angabe des Namensraumkürzel `xlink` ist dabei notwendig.

Die Namensraumangabe muss natürlich einem übergeordneten Element hinzugefügt werden.

Folgendes Listing zeigt unser Literaturverzeichnis, ergänzt um Links zu den Homepages der Autoren. Verweisanker ist das `<autor>`-Element:

Beispiel XLINK-3: Literaturliste mit XLink-Verweisen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<literaturliste>
  <eintrag>
    <autor xmlns:xlink="http://www.w3.org/1999/xlink"
           xlink:type="simple"
           xlink:href="http://www.peter.de">
```

```
      Peter Müller
    </autor>
    <titel>Mein erstes XML Buch</titel>
    <verlag>Addison-Wesley</verlag>
    <erscheinungsort>Bonn</erscheinungsort>
    <erscheinungsjahr>2000</erscheinungsjahr>
  </eintrag>

  <!-- Weitere Einträge -->

</literaturliste>
```

Zur Zeit unterstützten lediglich Netscape 6 und 7 und Mozilla 1.0 einige Features von XLink, wobei Netscape lediglich in der Version 6.0 den Link wie in untenstehender Abbildung durch Unterstreichung auszeichnet.

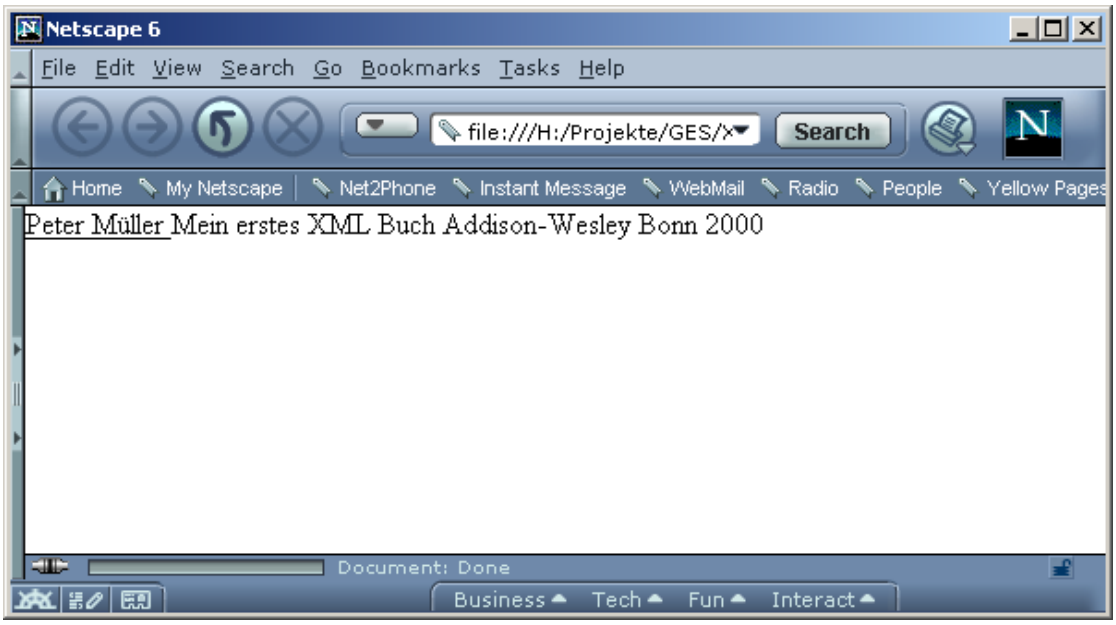


Abbildung XLINK-1: Darstellung von XLink-Verweisen im NS 6.0

Praxistipp:
Wissen Sie bereits im voraus, welche Elemente Sie (einfach) verlinken wollen, dann können Sie eine interne Teilmenge nutzen, um dort für dieses Element das Attribut `xlink:type` mit dem fixen Wert `simple` zu belegen.

Beispiel XLINK-4: Fixer Wert für zu verlinkende Elemente

```
<!DOCTYPE literaturliste [
    <!ATTLIST autor
  xmlns:xlink CDATA #FIXED http://www.w3.org/1999/xlink"
  xlink:type CDATA #FIXED "simple"
    >
]>
```

Erzeugen von Links mit XSLT

Um Links browserübergreifend darstellen zu können, müssen Sie z.Zt. die XML-Links mit XSLT in HTML-Links transformieren. Das Prinzip ähnelt dem Einbinden von Bildern. Folgendes Stylesheet bindet die Links aus obigem XML-Dokument in die Ausgabe mit ein:

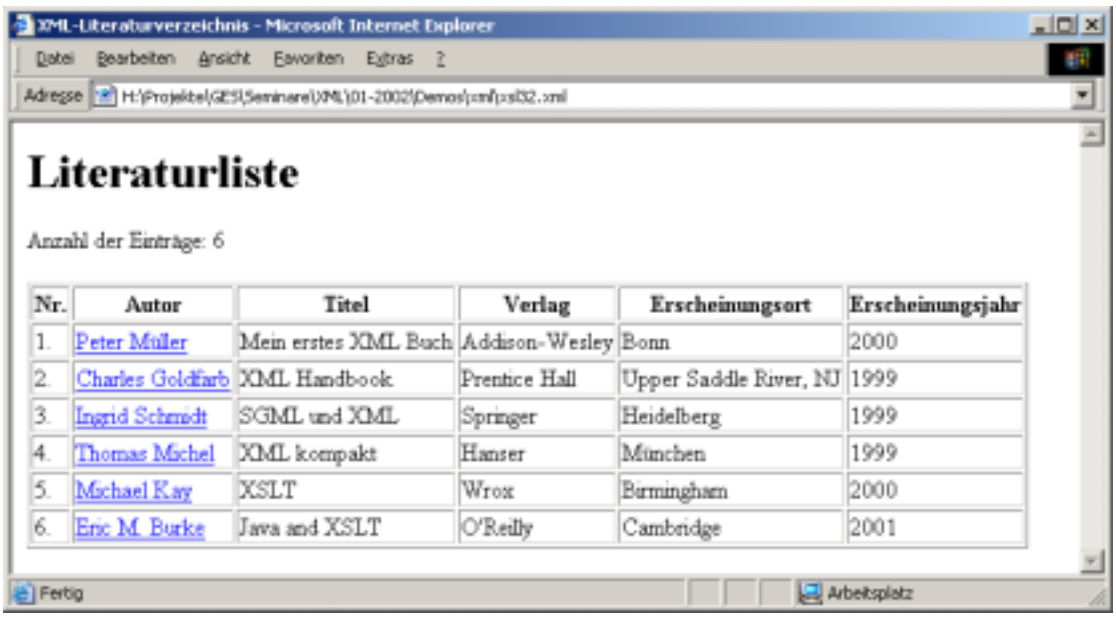
Beispiel XLINK-5: Links erzeugen mit XSLT

Stylesheet

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

  <xsl:template match="/literaturliste">
      <head>
        <title>XML-Literaturverzeichnis</title>
      </head>
      <body>
        <h1>Literaturliste</h1>
        <p>
          Anzahl der Einträge: <xsl:value-of se
ect="count(eintrag)"/>
        </p>
        <table border="1">
```

```
          <tr>
            <th>Nr.</th>
            <th>Autor</th>
            <th>Titel</th>
            <th>Verlag</th>
            <th>Erscheinungsort</th>
            <th>Erscheinungsjahr</th>


          </tr>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="eintrag">
    <tr>
      <td>
        <xsl:number format="1."/>
      </td>
      <xsl:apply-templates/>
    </tr>
  </xsl:template>

  <xsl:template match="autor">
    <td>
      <a href="{@xlink:href}">
        <xsl:value-of select="."/>
      </a>
    </td>
  </xsl:template>

  <xsl:template match="titel | verlag | erscheinungsort
                       | erscheinungsjahr">
    <td>
      <xsl:value-of select="."/>
    </td>
  </xsl:template>

</xsl:stylesheet>
```

Ansicht im Browser



Nr.	Autor	Titel	Verlag	Erscheinungsort	Erscheinungsjahr
1.	Peter Müller	Mein erstes XML Buch	Addison-Wesley	Bonn	2000
2.	Charles Goldfarb	XML Handbook	Prentice Hall	Upper Saddle River, NJ	1999
3.	Ingrid Schmidt	SGML und XML	Springer	Heidelberg	1999
4.	Thomas Michel	XML kompakt	Hanser	München	1999
5.	Michael Kay	XSLT	Wrox	Birmingham	2000
6.	Eric M. Burke	Java and XSLT	O'Reilly	Cambridge	2001