

XML

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

www.w3c.org

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential.
(RECOMMENDATION)

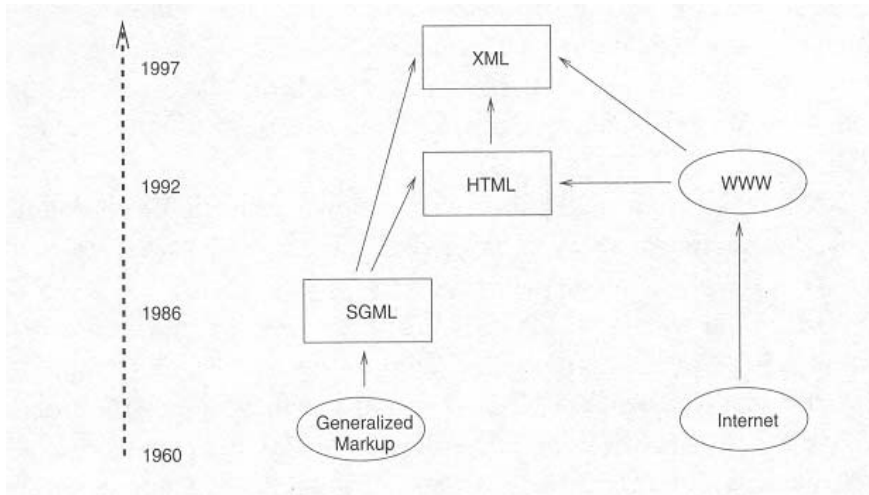
SGML

SGML ist die Abkürzung für **Structured Generalized Markup Language**, eine **Seitenbeschreibungs-Sprache**, die die **"Mutter"** der heute verwendeten Hypertext-Sprachen wie HTML oder XML.

Sie ist Bestandteil der ISO-Norm und wurde lange Zeit für den Austausch von Daten zwischen Firmen benutzt.

XML

Zeigt die Wurzeln von XML und ordnet diese zeitlich ein



Elemente

Die »Grundbausteine« von XML-Dokumenten sind Elemente. Sie bestehen jeweils aus einem Start-Tag und einem Ende-Tag, zwischen denen der Inhalt des Elements steht.

Tags sind Bezeichner, die den Inhalt des Elements beschreiben. Das Start-Tag wird in spitzen Klammern geschrieben (`<tag>`), beim Ende-Tag erscheint nach der öffnenden spitzen Klammer zusätzlich ein Schrägstrich `</tag>`. Die Bezeichner des Start- und des Ende-Tags müssen übereinstimmen.

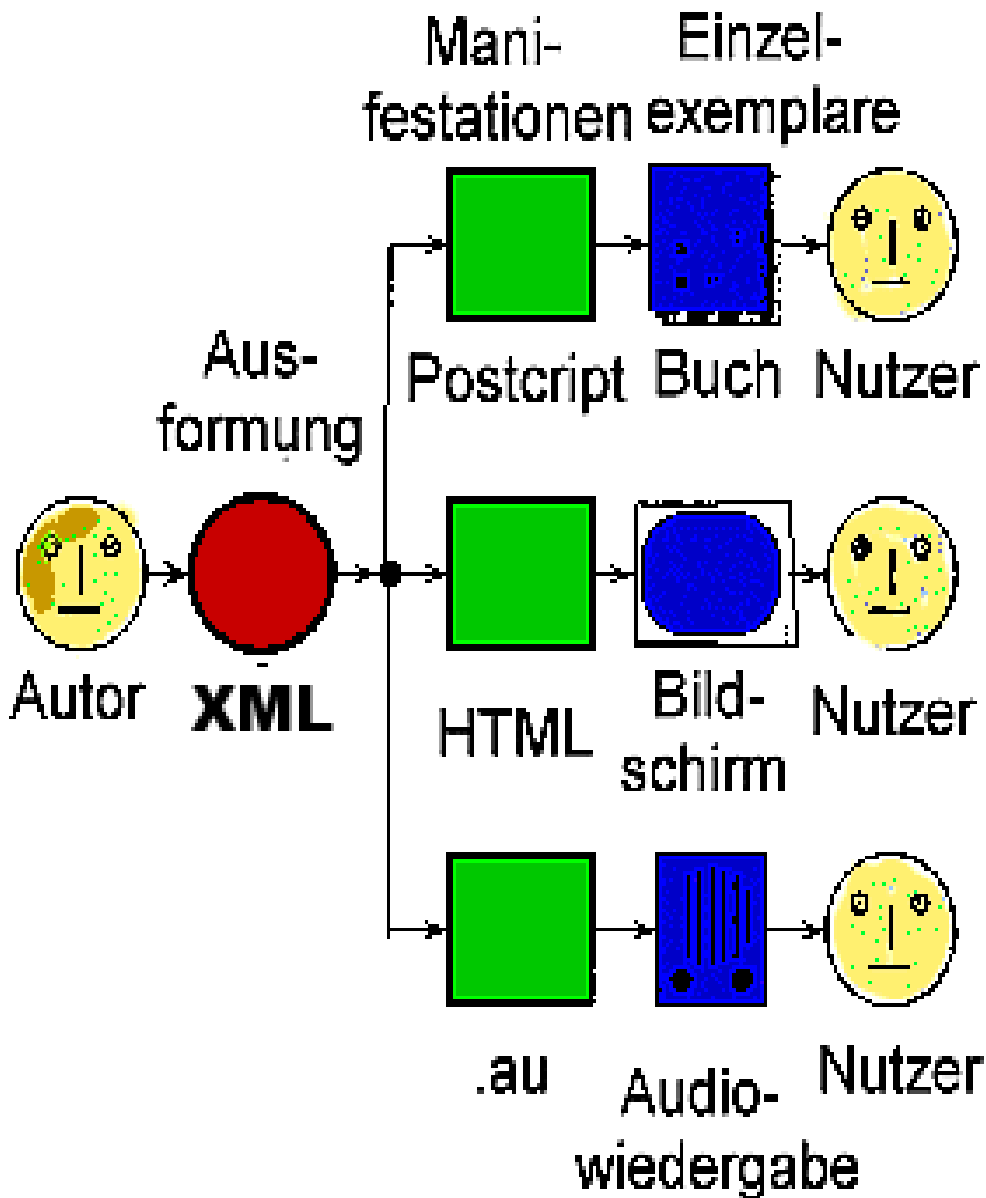
Alternativ können auch leere Elemente deklariert sein. Hier erscheint nur ein Tag, das von einer spitzen öffnenden Klammer und Schrägstrich mit spitzer schließender Klammer (`<emptytagj/>`) eingeschlossen ist.

Die Tags liefern Informationen über die Bedeutung der konkreten Werte, stellen also in der Datenbankterminologie Strukturinformationen dar. Im Inhalt eines Elements dürfen weitere Elemente, Zeichenketten, Processing Instructions und Kommentare auftreten. Diese werden; im Verlauf dieses Abschnitts beschrieben. Durch das Auftreten weiterer:

Elemente wird eine hierarchische Schachtelung der Dokumente möglich.

XML-Idee

XML-gestützte Publikation



XML-Vorteile

XML versucht Informationstypen (z.B. Text, Ton, Grafik, Video, usw.) relativ unabhängig zu speichern!

- Struktur (Kapitel, Absatz etc)
- Inhalt (Bedeutung: z.B. Name des Autors)
- Format (Schriftarten, Schriftgröße etc)

Vorteile:

- **Das Format kann unabhängig von den anderen Teilen verändert werden.**
- **Bei WWW-Anwendungen kann die Formatierung durch den lokalen Browser durchgeführt werden.**
- **Strukturdefinition und Formatsdefinition kann für verschiedene Inhalte wieder verwendet werden.**
- **Bei der Weitergabe von Daten gibt man nicht nur die Inhalte weiter sondern gibt auch gleich an, wie die Struktur der Daten aussieht. Der Empfänger kann nach dieser strukturellen Information seine Verarbeitung ausrichten.**

XML-Einordnung

Auszeichnungssprache:	... wie etwa HTML auch; mit ihr kann die Struktur, das Format und der Inhalt eines Dokumentes beschrieben werden.
Metasprache:	Mit ihrer Hilfe lassen sich eigene Sprachen zur Beschreibung bestimmter Dokumente oder Dokumenttypen definieren. Beispielsweise lässt sich HTML mit XML definieren. Mittlerweile gibt es einige mit XML definierte spezialisierte Auszeichnungssprachen. Etwa „MathML“ zur Beschreibung komplexer mathematischer Formeln oder „SMIL“ zur Integration und Synchronisation von Multimedia-Inhalten.

HTML vs. XML

```
<ul>
  <li>Curie</li>
  <li>Sokrates</li>
</ul>
<ul>
  <li>Mathematik</li>
  <li>Bioethik</li>
</ul>
```

```
<Universitaet>
  <Professoren>
    <ProfessorIn>Curie</ProfessorIn>
    <ProfessorIn>Sokrates</ProfessorIn>
  </Professoren>
  <Vorlesungen>
    <Vorlesung>Mathematik</Vorlesung>
    <Vorlesung>Bioethik</Vorlesung>
  </Vorlesungen>
</Universitaet>
```

Teile des Standards 1/2

XML:	Die „Extensible Markup Language“ ist eine Sprache zur logischen und semantischen Auszeichnung, die vom W3C standardisiert worden ist. Momentan (Stand 2003M01) ist die „Version 1.0 Second Edition“ gültig.
DTD:	Unter der „Document Type Definition“ ist ein Definition zu verstehen, die Elemente, Attribute, Entities, Reihenfolge, Anzahl, Werte und Datentypen vereinbart, die in einem XML-Dokument enthalten sein dürfen bzw. müssen. Die DTD ist in einer eigenen Syntax verfasst.
XML Schema:	Eine weitere, tiefer gehende und modernere Möglichkeit der Definition von Dokumenttypen bietet XML Schema; dabei liegt die Definition selbst als XML-Dokument vor – eine neue Syntax muss nicht erlernt werden. Momentan (Stand 2003M01) ist die „Version 1.0“ gültig, an der „Version 1.1“ wird gerade gearbeitet.
XSL:	Die „Extensible Stylesheet Language“ ist das, was CSS (Cascading Style Sheet) für HTML ist: sie um die Ausgabe von XML. Sie besteht aus zwei Teilen: <ul data-bbox="289 1354 1328 1572" style="list-style-type: none">• XSLT, eine Sprache zur Konvertierung von XML Dokumenten (z.B. in HTML)• XSL-FO, eine Sprache zur Formatierung von XML-Dokumenten (z.B. zur Ausgabe auf einem Drucker ohne HTML-Umweg)

Teile des Standards 2/2

XSLT:	Mit Hilfe von „XSL Transformations“ wandelt man ein XML-Dokument beispielsweise in ein anderes XML-Format oder in HTML-fähige Konstrukte zur Präsentation auf einen Browser um (oder in PDF, ...). Im Gegensatz zu CSS enthält XSL jedoch typische Konstrukte einer Programmiersprache, nämlich Schleifen, Bedingungen und Variable.
XSL-FO:	XSL Formatting Objects ist eine Sprache, die vor allem der professionellen Druckausgabe (als vor allem der Umwandlung in Postscript) dient: XSL im engeren Sinne. Die Spezifikationen von XSL-FO sind sehr umfangreich und gehören in die Hände von gelernten Schriftsetzern.
XPath:	XPath ist eine deklarative Sprache zum Durchsuchen der Inhalte eines XML-Dokuments. XPath ermöglicht die Navigation entlang einer hierarchischen Struktur – dem sog. „XML Information Set“.
XLink:	Die „XML Linking Language“ ermöglicht das Einfügen von Links in XML-Dokumente: damit sind einerseits sowohl die HTML-artigen, einfachen unidirektionalen Links, als auch weiterführende Konzepte (bidirektionale Links, Zugriff auf einzelne Objekte, Links zu mehreren Quellen) möglich.

XML-Schnittstellen

Es gibt zwei standardisierte Schnittstellen für den Zugriff auf XML-Dokumente:

SAX:	XML-Prozessoren auf der Basis von SAX (Simple API for XML) realisieren ein ereignisgesteuertes Vorgehen zum Parsen von XML-Dokumenten. Bei diesen Prozessoren wird ein XML-Dokument sequenziell abgearbeitet. Bereits während der Abarbeitung wird die Applikation über das Auftreten der einzelnen Bestandteile (z.B. Start- und Ende-Tags von Elementen) durch das Aufrufen spezieller Methoden informiert. Auf diese Weise wird eine einfache Variante des Zugriffs auf die einzelnen Bestandteile des Dokuments realisiert.
DOM:	Das „ Document Object Model “ ist das standardisierte API für die Erstellung und Manipulation eines „XML Information Sets“, der a) aus einem XML-Dokument erstellt werden kann (parsing) und aus dem b) wieder ein XML-Dokument erstellt werden kann (serialisieren).

Beispiel für XML-Prozessoren:

Xerces	Im Rahmen des Apache-Projekts wurden DOM- und SAX Prozessoren entwickelt. Es handelt sich hierbei um validierende Parser für Java und C++. Implementiert sind dabei DOM, Level1 und 2 sowie SAX 2.0. Der Prozessor unterstützt ab der Version 1.4.1 XML Schema und kann beim Parsen von Dokumenten die Gültigkeit gegenüber einem XML-Schema testen. (xml.apache.org)
MSXML Parser	Ein weiteres, frei verfügbares Produkt ist der Microsoft XML Parser (MSXML). Er enthält einen DOM- und einen SAX-Prozessor. Diese Prozessoren können die Konformität von XML-Dokumenten bezüglich eines zugehörigen XML-Schema überprüfen. (msdn.microsoft.com/downloads/)

XML- Einsatzgebiete

- **WWW 1:** Um den Server von der Formatierung zu entlasten und darüber hinaus Browser-individuelle Formatierungen zu erlauben, können Inhalte von Web-Seiten (XML) z.B. von einem Client-seitigem JavaScript als HTML dargestellt werden – für XML-unkompatible Browser (z.B. IE < Version 5.5, NE 4, ...) übrigens die einzige Möglichkeit, XML-Dokumente darzustellen..
- **WWW 2:** Versteht der Browser allerdings schon XML, so kann er mit Hilfe von XSLT die Umwandlung in HTML selbst (ohne zusätzlicher Scripts) durchführen – und diesen Code dann wie gewohnt darstellen.
- **WWW 3:** Der gesamte Inhalt der Site liegt in XML vor und wird mittels Java Server Pages via XSLT nach HTML umgewandelt; derzeit eine Hauptanwendung von XML und XSLT.
- **Neue Sprachen:** Durch die strenge Syntax können weitere (somit XML-basierte) Sprachen entwickelt werden. Dazu gehört WML, XHTML, SVG und MathML.
- **Dateiformat:** Da XML genau spezifiziert ist, eignet sich dieses präzise Dateiformat sehr gut zum plattformunabhängigen Datenaustausch.
- **Sonstiges:** Konfigurationsdateien, normales Dateiformat zum Speichern von Daten (MS Excel XP, MS Access XP, StarOffice 6.0)

Abteilungsdatenbank

```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<!DOCTYPE abteilungen SYSTEM "abteilungen.dtd">
<abteilungen>
  <abteilung>
    <abtNr>10</abtNr>
    <abtName>Einkauf</abtName>
    <abtOrt>Linz</abtOrt>
  </abteilung>
  <abteilung>
    <abtNr>20</abtNr>
    <abtName>Verkauf</abtName>
    <abtOrt>Wels</abtOrt>
  </abteilung>
</abteilungen>
```

XML-Namespaces

Prinzipiell ist es erlaubt, in einem XML-Dokument Tags mehrerer

Dokumenttypen zu verwenden – also mehrere DTD oder XML Schema anzugeben. Mit Namensräumen soll das Problem gleich benannter Tags in verschiedenen Definitionen

gelöst werden.

```
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

DTD - Einführung

Was ist das DTD:

In einer Document Type Definition werden die Grundregeln für die Struktur eines Dokuments festgelegt.

```
<!-- DTD-Definition Adressen -->
<!ELEMENT adresse (vorname, name, strasse, plz, ort)>
<!ELEMENT      vorname      #PCDATA>
<!ELEMENT      name         #PCDATA>
<!ELEMENT      strasse      #PCDATA>
<!ELEMENT      plz          #PCDATA>
<!ELEMENT      ort          #PCDATA>
```

- Mithilfe der erstellten DTD ließe sich das unten stehende Dokument mit externem Hinweis auf adressen.dtd erzeugen:

```
<?xml version="1.0"?>
<!DOCTYPE adressen SYSTEM "adressen.dtd">
<ADRESSE>
  <VORNAME>      Peter
  <NAME>         Meyer
  <STRASSE>      Waldweg 5
  <PLZ>         36756
  <ORT>         Detmold
</ADRESSE>
```

DTD

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<!DOCTYPE customers[
  <!ELEMENT customers (customer*)>
    <!ELEMENT customer (name, address+)>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT address (street, city, state?,
        postal)>
        <!ATTLIST address country NMTOKEN
          #REQUIRED>
        <!ELEMENT street (#PCDATA)>
        <!ELEMENT city (#PCDATA)>
        <!ELEMENT state (#PCDATA)>
        <!ELEMENT postal (#PCDATA)>
    ]>
<customers>
<customer>
  <name>Baeckerei Huber</name>
  <address country=" AT ">
    <street>Landstrasse 63</street>
    <city>Linz</city>
    <postal>4020</postal>
  </address>
</customer>
<customer>
  ...
</customers>
```

Elemente im DTD

<?xml	
<!DOCTYPE	
[
<!ELEMENT	
Name	
address+	
State?	
<!ATTLIST	
NMTOKEN	
#REQUIRED	
#PCDATA	
]>	

DTD extern

car.dtd:

```
<!ELEMENT cars (car*)>
  <!ELEMENT car (#PCDATA)>
    <!ATTLIST car
      type (coupe|limo|cabrio|kombi) "coupe"
      id CDATA #REQUIRED
      color CDATA #IMPLIED
    >
```

(coupe limosine ...	
"coupe"	
#IMPLIED	
<!DOCTYPE	
Ford	
VW	
Nissan	

car.xml:

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<!DOCTYPE cars SYSTEM "cars.dtd">
<cars>
  <car type="cabrio" id="L-0815Y"
    color="red">Ford</car>
  <car id="L-4711Z" color="blue">VW</car>
  <car type="kombi" id="L-1232Y">Nissan</car>
</cars>
```

XML-Schema

An und für sich findet man mit der DTD sein auslangen; jedoch führten folgende Schwächen zu der Entwicklung des XML Schema:

- Keine XML-konforme Syntax (unangenehm für Entwickler, aber vor allem keine DOM-Unterstützung (API) zur Manipulation von Programmen aus!)
- Nicht erweiterbar
- Schlechte Unterstützung von XML Namespaces
- Schwache Unterstützung von Datentypen (numerisch?)
- Geringe Möglichkeiten zur exakten Definition der Inhalte von Tags
- Keine Erweiterbarkeit

**Ergänzung zu den
Namensräumen:**

**[XML-Tipp Nr 5
Namensraum.htm](#)**

XML-Schema I

Index.xsd:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Buch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Titel"
          type="xsd:string"/>
        <xsd:element name="Inhalt"
          type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

<?xml version ...	
<xsd:schema ...	
... xmlns:xsd="...	
<xsd:element ...	
<xsd:complexType>	
<xsd:sequence>	
<xsd:element ...	

XML-Schema I

Index.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Buch
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="http://members
  .aon.at/tumfart/index.xsd">
  <Titel>An Introduction to Database
  Systems</Titel>
  <Inhalt>Datenmodellierung</Inhalt>
</Buch>
```

<?xml ...	
<Buch ...	
... xmlns:xsi="..."	
... xsi:noName sp...	

XML-Schema II 1/3

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Purchase order schema for Example.com.
      Copyright 2000 Example.com.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="purchaseOrder
                type="PurchaseOrderType"/>

  <xsd:element name="comment"
type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo"
type="USAddress"/>
      <xsd:element name="billTo"
type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate"
                    type="xsd:date"/>
  </xsd:complexType>
```

XML-Schema II 2/3

```
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN,,
    fixed="US"/>
</xsd:complexType>
```

```
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0,,
      maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
```

XML-Schema II 3/3

```
<xsd:element name="USPrice" type="xsd:decimal"/>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="shipDate" type="xsd:date,,
    minOccurs="0"/>
</xsd:sequence>
<xsd:attribute name="partNum" type="SKU,,
  use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

XML-Schema II - Erklärung

<?xml version ...	
<xsd:schema ...	
<xsd:annotation>	
<xsd:element ... purchaseOrder ...	
<xsd:element ... comment ...	
<xsd:complexType ... PurchaseOrderType ...	
... ref="comment" ...	
... minOccurs ...	
<xsd:attribute ...	
... type="xsd:NMTOKEN" ...	
... fixed="US" ...	
<xsd:simpleType>	
<xsd:restriction ...	
... base="..."	
<xsd:maxExclusive value=...>	
<xsd:attribute ... partNum ...	
<xsd:pattern ...	

XML-Schema Datentypen

String

Name	Description
ENTITIES	
ENTITY	
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
IDREFS	
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NCName	
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
NMTOKENS	
normalizedString	A string that does not contain line feeds, carriage returns, or tabs
QName	
string	A string
token	A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

Date / Time

Name	Description
date	Defines a date value
dateTime	Defines a date and time value
duration	Defines a time interval
gDay	Defines a part of a date - the day (DD)
gMonth	Defines a part of a date - the month (MM)
gMonthDay	Defines a part of a date - the month and day (MM-DD)
gYear	Defines a part of a date - the year (CCYY)
gYearMonth	Defines a part of a date - the year and month (CCYY-MM)
time	Defines a time value

Numeric

Name	Description
Byte	A signed 8-bit integer
Decimal	A decimal value
Int	A signed 32-bit integer
Integer	An integer value
Long	A signed 64-bit integer
negativeInteger	An integer containing only negative values (.., -2, -1.)
nonNegativeInteger	An integer containing only non-negative values (0, 1, 2, ..)
nonPositiveInteger	An integer containing only non-positive values (.., -2, -1, 0)
positiveInteger	An integer containing only positive values (1, 2, ..)
Short	A signed 16-bit integer
unsignedLong	An unsigned 64-bit integer
unsignedInt	An unsigned 32-bit integer
unsignedShort	An unsigned 16-bit integer
unsignedByte	An unsigned 8-bit integer

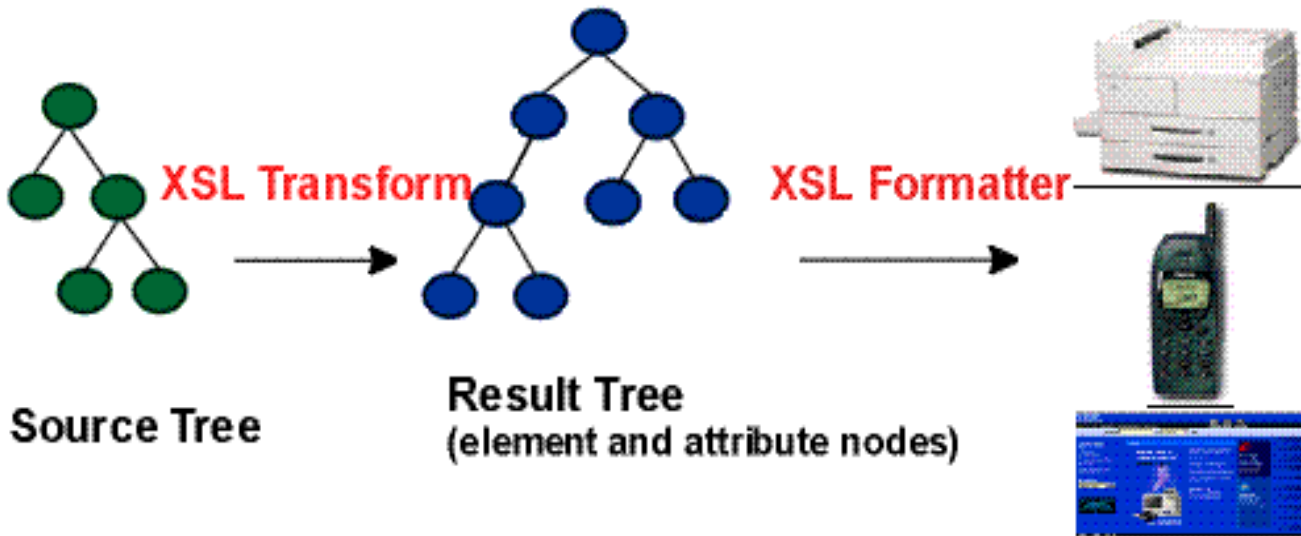
Restrictions

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) are handled

Qualitätsattribute

well-formed:	<p>Ein „wohlgeformtes“ Dokument muss den generellen XML-Regeln entsprechen - also:</p> <ul style="list-style-type: none">• Das gesamte Dokument muss in ein einzelnes Wurzelement eingeschlossen sein. Im Beispiel „<abteilungen>“.• Alle Tags müssen durch einen Ende-Tag geschlossen werden.• Alle Tags müssen richtig geschachtelt sein (nicht: "<a>")• Attributswerte müssen unter Hochkommata stehen.
Valid:	<p>Ein „gültiges“ Dokument muss darüber hinaus seiner Dokumentstyp-Definition entsprechen:</p> <ul style="list-style-type: none">• Das Dokument ist wohlgeformt.• Es gibt eine DTD oder ein XSD.• Die Werte von Elementen und Attributen müssen dem angegebenen Datentyp entsprechen.• Die Daten müssen entsprechend der DTD bzw. XSD strukturiert sein.
Canonical:	<p>Jüngst ist noch ein weiteres Qualitätsattribut aufgekommen - kanonische XML (kanonisch = anerkannt, autorisiert, vollständig, vollständig):</p> <ul style="list-style-type: none">• Das Dokument ist in UTF-8 codiert.• XML-Deklarationen und DTD / XSD sind ausgelagert.• Leere Elemente werden zu Beginn- und Ende-Tags konvertiert.• Überflüssige Namespace-Deklarationen werden entfernt.• Überflüssige Leerzeichen werden entfernt.• Default-Attribute werden jedem Element hinzugefügt.• ...

XSL



Result XML tree is the result of XSLT processing.

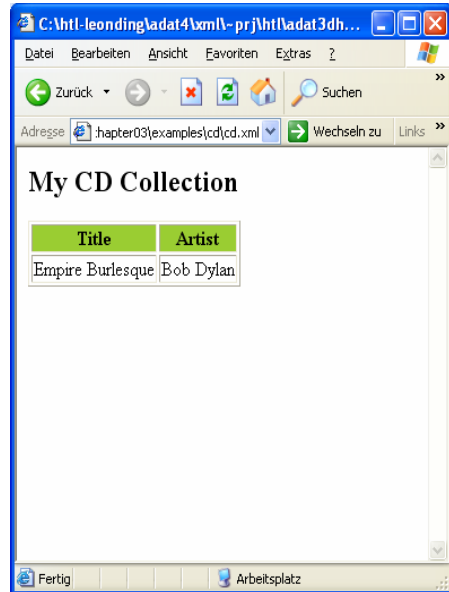
XSLT dient zur Transformation eines „XML Source Tree“ in einen „XML Result Tree“ ... also der Umwandlung eines XML-Dokuments eines bestimmten Dokumenttyps in ein Dokument eines anderen Typs. Ein Spezialfall ist die Umwandlung in den Dokumenttyp XHTML – das ist die Re-Definition von HTML mit Hilfe von XML Schema: diese Dokumente können von einem XML-fähigen Browser dann angezeigt werden.

XSL-Beispiel (cd.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cd1.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
  .....
</catalog>
```

XSL-Beispiel (cd1.xsl)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xslt:stylesheet version="1.0"
  xmlns:xslt="http://www.w3.org/1999/XSL/Transform">
<xslt:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <tr>
        <td><xslt:value-of select="catalog/cd/title"/></td>
        <td><xslt:value-of select="catalog/cd/artist"/></td>
      </tr>
    </table>
  </body>
</html>
</xslt:template>
</xslt:stylesheet>
```



XSL-Beispiel (cd2.xsl)

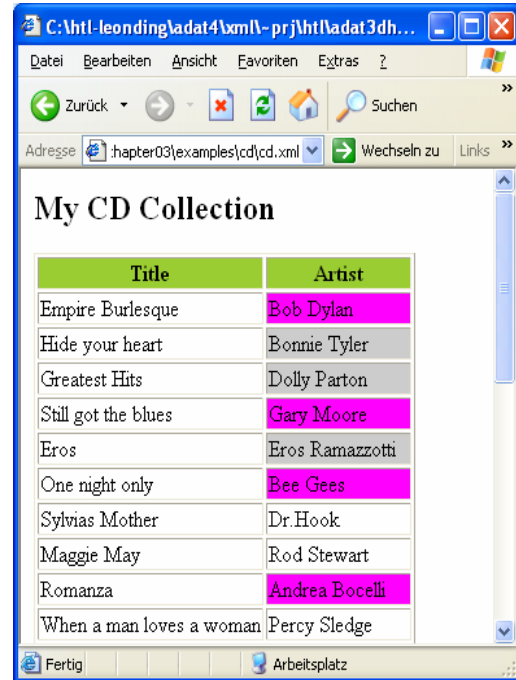
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xslt:stylesheet version="1.0"
  xmlns:xslt="http://www.w3.org/1999/XSL/Transform"
  >
<xslt:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xslt:for-each select="catalog/cd">
        <xslt:sort select="artist"/>
      <tr>
        <td><xslt:value-of select="title"/></td>
        <td><xslt:value-of select="artist"/></td>
      </tr>
    </xslt:for-each>
  </table>
  </body>
  </html>
</xslt:template>

</xslt:stylesheet>
```



XSL-Beispiel (cd3.xsl)

```
<?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>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <xsl:choose>
            <xsl:when test="price>'10'">
              <td bgcolor="#ff00ff">
                <xsl:value-of select="artist"/></td>
            </xsl:when>
            <xsl:when test="price>'9' and price<='10'">
              <td bgcolor="#cccccc">
                <xsl:value-of select="artist"/></td>
            </xsl:when>
            <xsl:otherwise>
              <td><xsl:value-of select="artist"/></td>
            </xsl:otherwise>
          </xsl:choose>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```



XSL-Beispiel (cd4.xsl)

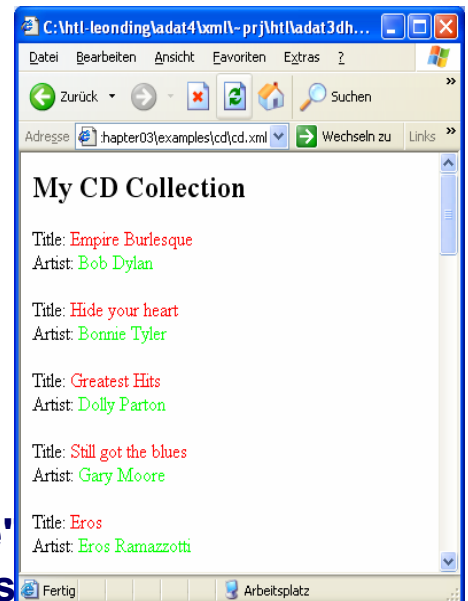
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xslt:stylesheet version="1.0"
xmlns:xslt="http://www.w3.org/1999/XSL/Transform">
```

```
<xslt:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<xslt:apply-templates/>
</body>
</html>
</xslt:template>
```

```
<xslt:template match="cd">
<p>
<xslt:apply-templates select="title"
<xslt:apply-templates select="artist"
</p>
</xslt:template>
```

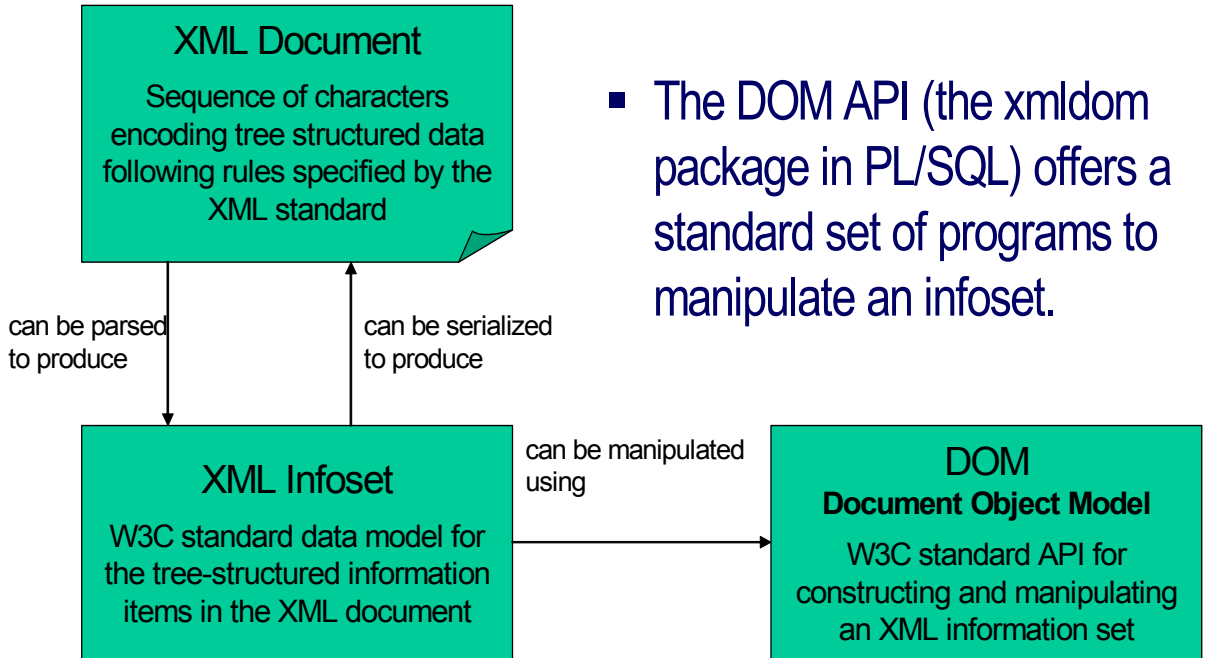
```
<xslt:template match="title">
Title: <span style="color:#ff0000">
<xslt:value-of select="."/></span>
<br />
</xslt:template>
```

```
<xslt:template match="artist">
Artist: <span style="color:#00ff00">
<xslt:value-of select="."/></span>
<br />
</xslt:template>
</xslt:stylesheet>
```



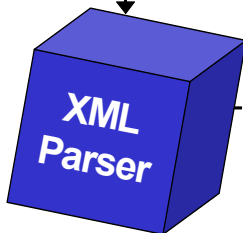
Dokument <> DOM

- The DOM API (the xmldom package in PL/SQL) offers a standard set of programs to manipulate an infoset.

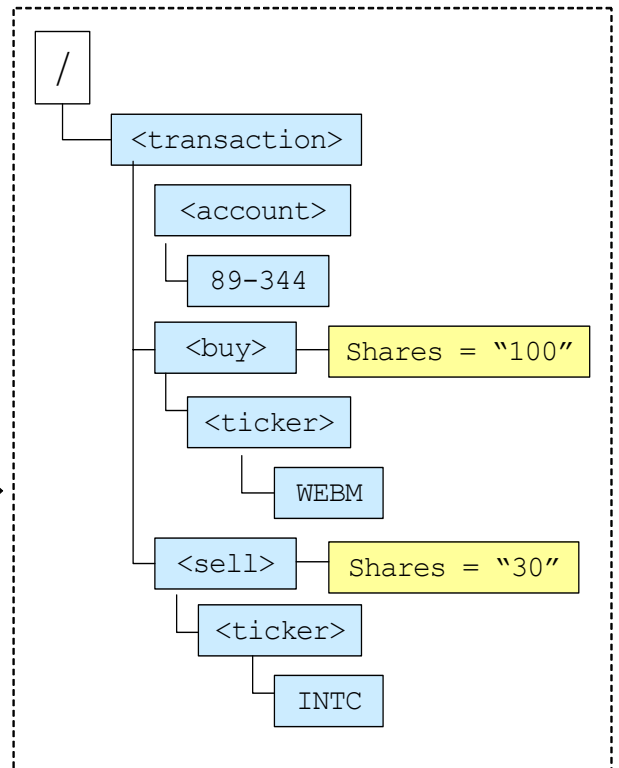


Text Document

```
<?xml version = "1.0"?>  
<transaction><account>89-344</ac  
count><buy shares = "100"><ticker>  
WEBM</ticker></buy><sell shares=  
"30"><ticker>INTC</ticker></sell  
></transaction>
```



"Information Set"



DOM

W3C DOM Specifications and Timeline

Specification	Latest Draft	Proposed	Recommendation
DOM Level 1	20. Jul 98	18. Aug 98	01. Oct 1998
DOM Level 1 (SE)	29. Sep 00		
DOM Level 2 Core	10. May 00	27. Sep 00	13. Nov 2000
DOM Level 2 HTML	10. Dec 01	08. Nov 02	09. Jan 2003
DOM Level 2 Views	10. May 00	27. Sep 00	13. Nov 2000
DOM Level 2 Style	10. May 00	27. Sep 00	13. Nov 2000
DOM Level 2 Events	10. May 00	27. Sep 00	13. Nov 2000
DOM Level 2 Traversal-Range	10. May 00	27. Sep 00	13. Nov 2000
DOM Level 3 Requirements	26. Feb 04		
DOM Level 3 Core	07. Nov 03	05. Feb 04	07. Apr 2004
DOM Level 3 Events	07. Nov 03		
DOM Level 3 Load and Save	19. Jun 03	05. Feb 04	07. Apr 2004
DOM Level 3 Validation	05. Feb 03	15. Dec 03	27. Jan 2004
DOM Level 3 XPath	26. Feb 04		
DOM Level 3 Views	26. Feb 04		

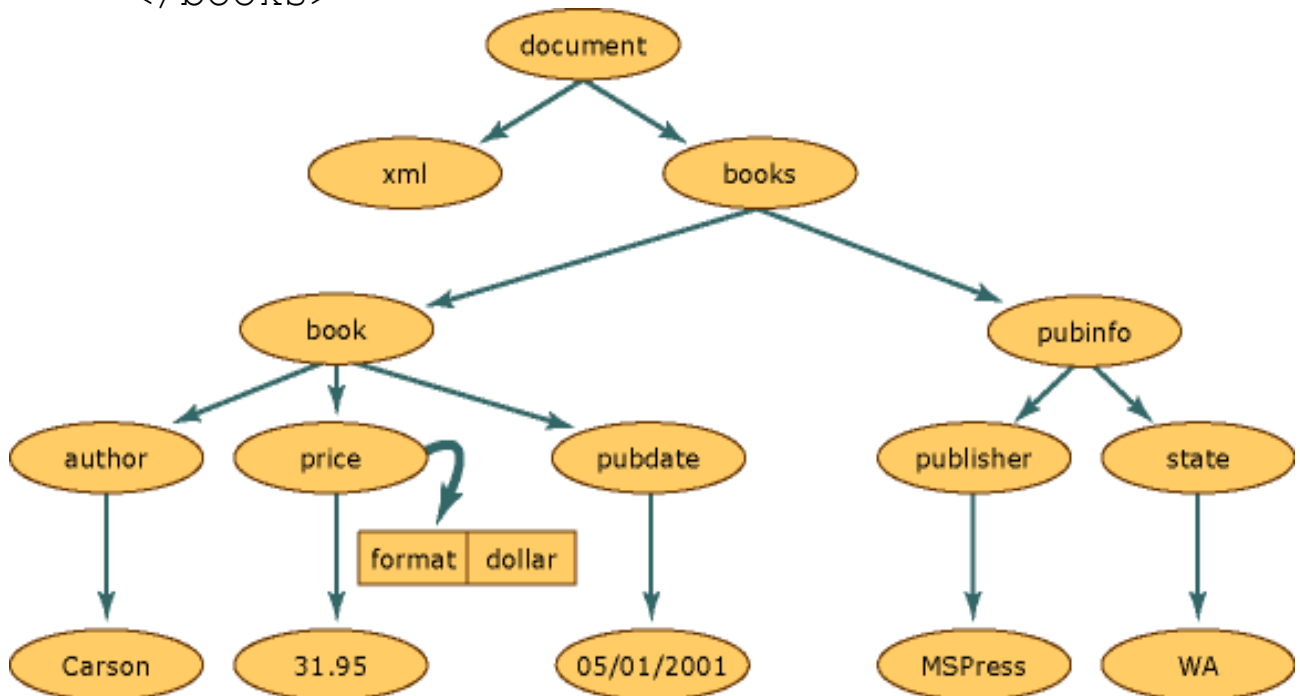
Proposed... vorgeschlagen
Recommendation... Empfehlung

DOM Levels Überblick

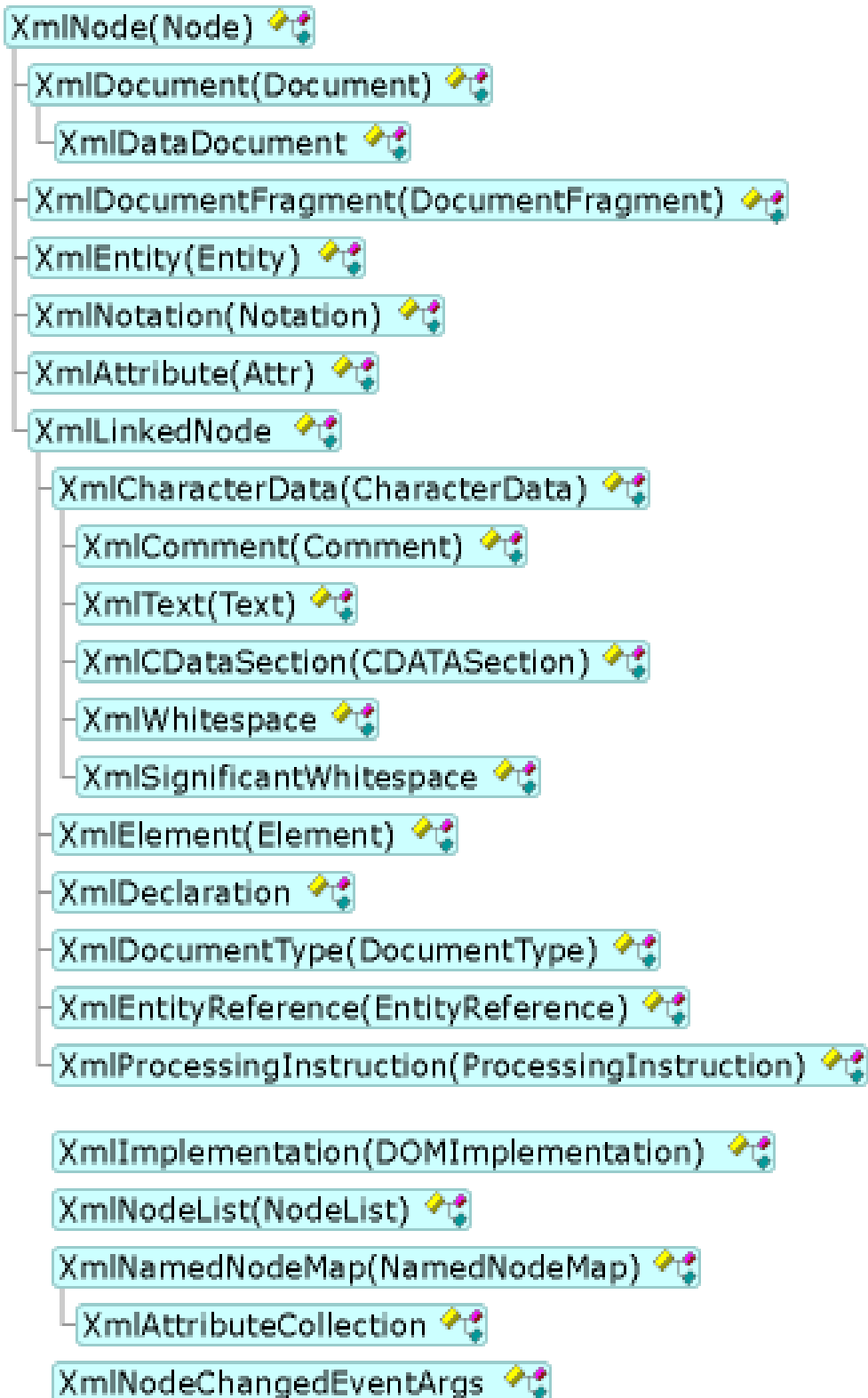
- Level 1: core, die HTML und XML document models
- Level 2: inkludiert style sheet object model, Manipulation von styles, Traversieren des Dokumentes, Unterstützung für XML Namespaces
- Level 3: wird sich unter anderem beschäftigen mit Laden/Speichern von Dokumenten, Formatierung von Dokumenten
- Weitere Levels: mögliche Themen sind Sicherheit/User Interaction

Dokument <> DOM

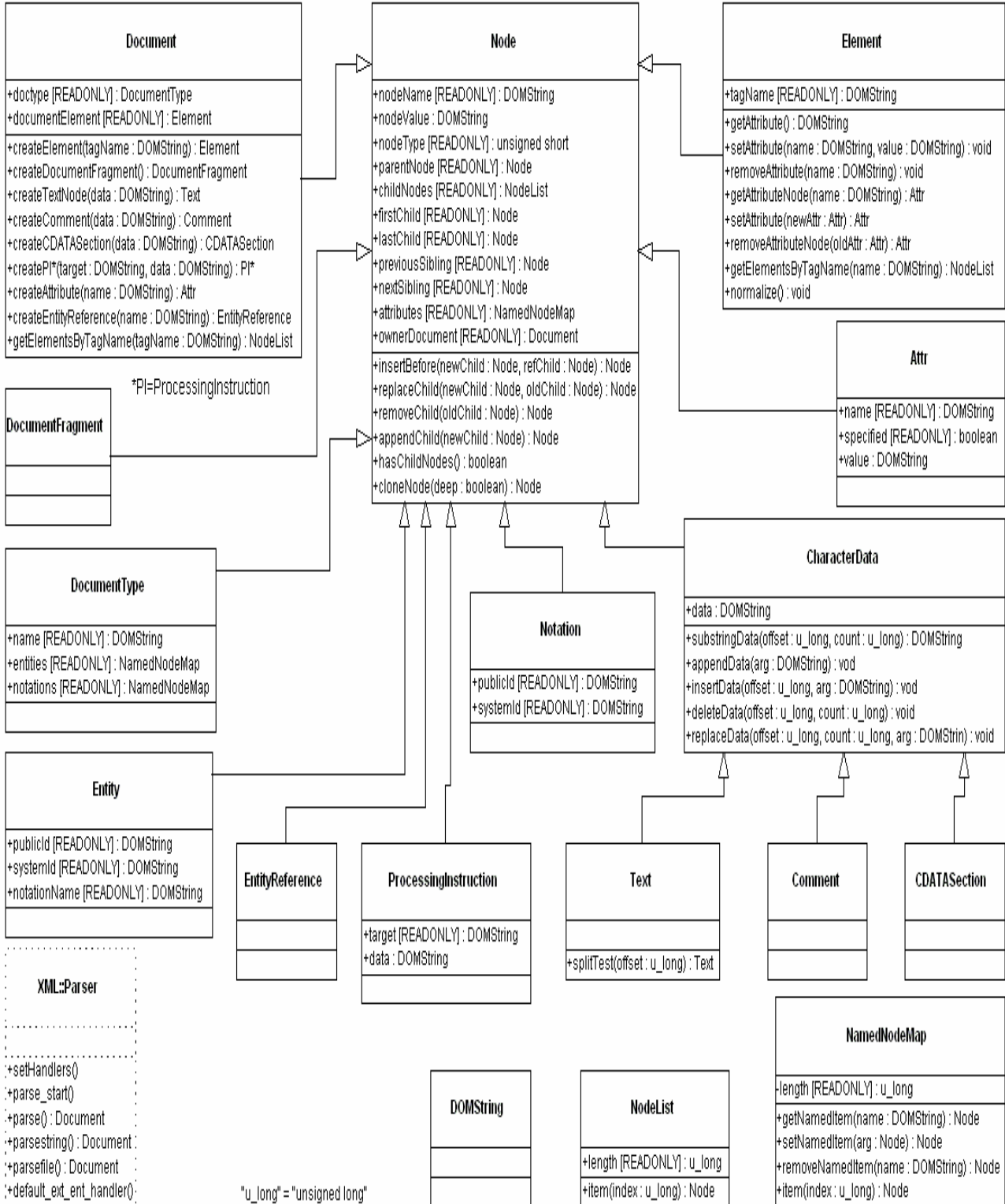
```
<?xml version="1.0"?>
<books>
  <book>
    <author>Carson</author>
    <price format="dollar">31.95</price>
    <pubdate>05/01/2001</pubdate>
  </book>
  <pubinfo>
    <publisher>MSPress</publisher>
    <state>WA</state>
  </pubinfo>
</books>
```



DOM Class Hierarchy



DOM



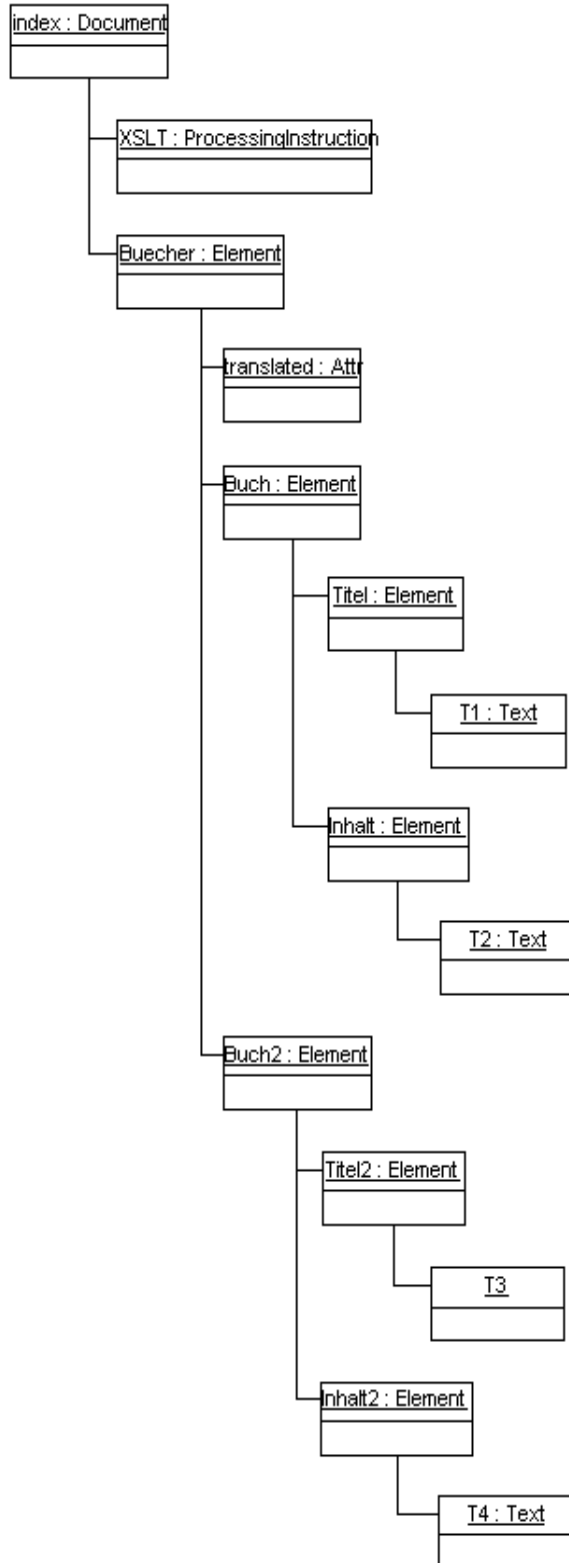
Beispiel – index.xml

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

<?xml-stylesheet type="text/xsl" href="index_List.xsl"
  title="Liste" alternate="yes"?>
<?xml-stylesheet type="text/xsl" href="index_Table2.xsl"
  title="Tabelle" alternate="no"?>

<Buecher xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:noNamespaceSchemaLocation="index.xsd">
  <Buch>
    <Titel>An Introduction to Database Systems</Titel>
    <Inhalt>Datenmodellierung</Inhalt>
  </Buch>
  <Buch>
    <Titel>XML: Das Einsteigerseminar</Titel>
    <Inhalt>XML, XSLT, DTD, etc...</Inhalt>
  </Buch>
</Buecher>
```

Infoset von Index



DOM / VBScript (1)

```
<html>
<body>
<script type="text/vbscript">

set DocOld = createObject("Microsoft.XMLDOM")
DocOld.async = "false"
DocOld.load("index.xml")

set DocNew = createObject("Microsoft.XMLDOM")
DocNew.async = "false"

set NewPI = DocNew.createProcessingInstruction("xml-
    stylesheet", "type=""text/xsl"" href=""index_List.xsl"")
DocNew.appendChild(NewPI)

set NewNode =
    DocNew.createElement(translate(DocOld.documentElement.
        nodeName))
set Father = DocNew.appendChild(NewNode)

for each P in DocOld.documentElement.childNodes
    set NewNode =
        DocNew.createElement(translate(P.nodeName))
    call NewNode.setAttribute("translated", "yes")
    set Father2 = Father.appendChild(NewNode)
```

DOM / VBScript (2)

```
for each Q in P.childNodes
  set NewNode =
    DocNew.createElement(translate(Q.nodeName))
  set Father3 = Father2.appendChild(NewNode)

  for each L in Q.childNodes
    set NewNode = DocNew.createTextNode(L.nodeValue)
    Father3.appendChild(NewNode)
  next
next
next

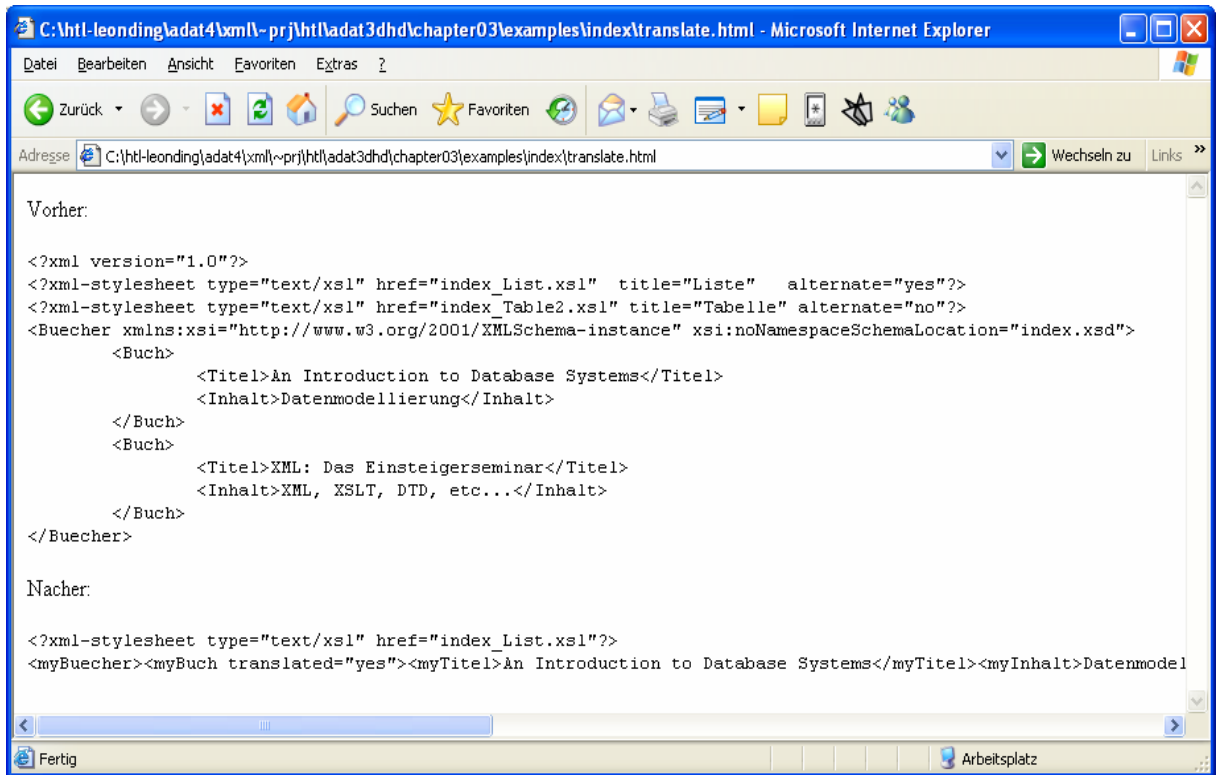
document.write("Vorher:")
document.write("<xmp>" & DocOld.xml & "</xmp>")

document.write("Nacher:")
document.write("<xmp>" & DocNew.xml & "</xmp>")

function Translate(TextOld)
  Translate = "my" & TextOld
end function

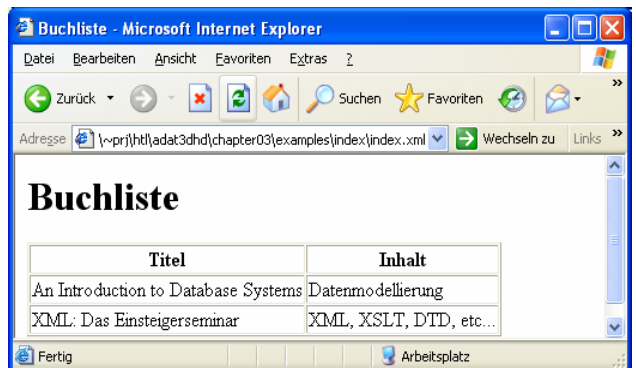
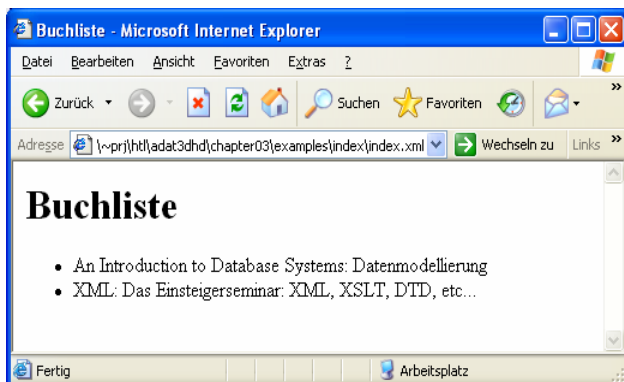
</script>
</body>
```


DOM/VBScript Ergebnis



Hinweis:

<http://www.w3schools.com/dom/default.asp>

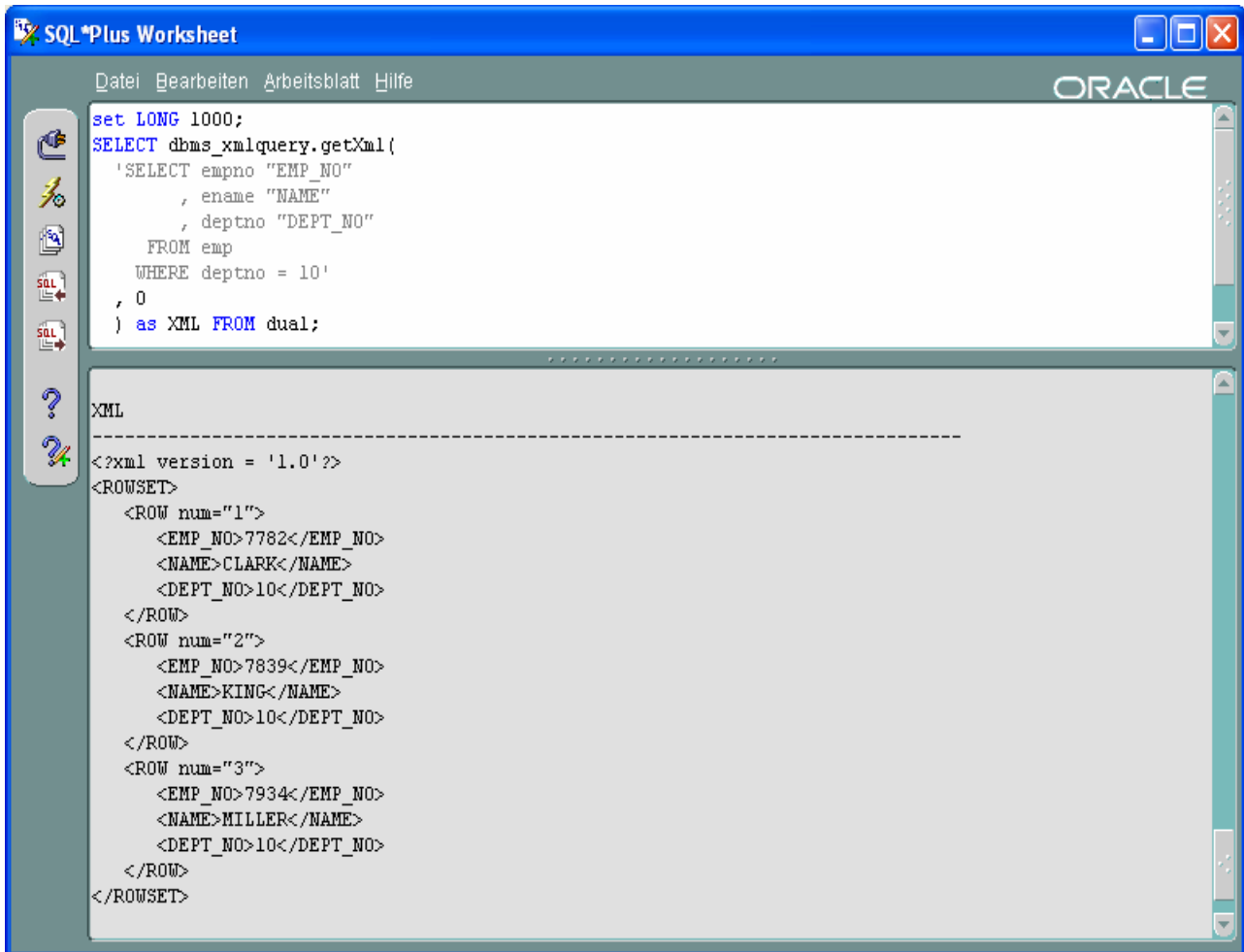


Index_List.xsl

Index_Table2.xsl

Oracle XDK

Oracle Für das Produkt Oracle wird ab Version 8i das XML Developer's Kit (Oracle XDK) bereitgestellt. Es enthält Grundbausteine zum Lesen, Manipulieren, Transformieren und Anzeigen von XML-Dokumenten. Im XML Developer's Kit gibt es ein XML SQL Utility (XSU), das die Ausgabe von Datenbankinhalten mit XML-Syntax unterstützt. Da- mit besteht die Möglichkeit, vollständige Datenbankinhalte relationaler Datenbanken auf XML-Dokumente abzubilden. Die Attribute der Datenbank werden als Elemente eines XML-Dokuments dargestellt. Einfache Typen (skalare Werte) werden auf Elemente, die als #PCDATA definiert sind, abgebildet, strukturierte Typen und ihre Attribute werden auf Elemente mit Subelementen abgebildet. Aus Kollektionstypen ent- stehen Listen von Elementen. Hier werden also die Objektrelationalen Merkmale adäquat im resultierenden XML-Dokument abgebildet.



The screenshot shows the SQL*Plus Worksheet interface. The top menu bar includes 'Datei', 'Bearbeiten', 'Arbeitsblatt', and 'Hilfe'. The 'ORACLE' logo is visible in the top right corner. The main window is divided into two sections. The upper section contains the following SQL code:

```
set LONG 1000;
SELECT dbms_xmlquery.getXml(
  'SELECT empno "EMP_NO"
    , ename "NAME"
    , deptno "DEPT_NO"
  FROM emp
  WHERE deptno = 10'
, 0
) as XML FROM dual;
```

The lower section, titled 'XML', displays the output of the query as an XML document:

```
XML
-----
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <EMP_NO>7782</EMP_NO>
    <NAME>CLARK</NAME>
    <DEPT_NO>10</DEPT_NO>
  </ROW>
  <ROW num="2">
    <EMP_NO>7839</EMP_NO>
    <NAME>KING</NAME>
    <DEPT_NO>10</DEPT_NO>
  </ROW>
  <ROW num="3">
    <EMP_NO>7934</EMP_NO>
    <NAME>MILLER</NAME>
    <DEPT_NO>10</DEPT_NO>
  </ROW>
</ROWSET>
```

Create XML docs using XSU

```
CONNECT scott/tiger@sid
--
SET SERVEROUTPUT ON
--
-- Simple procedure to support printing out a CLOB to screen
CREATE OR REPLACE PROCEDURE printClobOut(result IN OUT NOCOPY CLOB) IS
  xmlstr VARCHAR2(32767);
  line VARCHAR2(2000);
BEGIN
  xmlstr := dbms_lob.substr(result,32767);
  LOOP
    EXIT WHEN xmlstr IS NULL;
    line := substr(xmlstr,1,instr(xmlstr,chr(10))-1);
    dbms_output.put_line(line);
    xmlstr := substr(xmlstr,instr(xmlstr,chr(10))+1);
  END LOOP;
END;
/
SHOW ERRORS;
--
-- Simple example with binding variable
DECLARE
  queryCtx dbms_xmlquery.ctxType;
  result CLOB;
BEGIN
  -- set up the query context
  queryCtx := dbms_xmlquery.newContext(
    'SELECT empno "EMP_NO"
      , ename "NAME"
      , deptno "DEPT_NO"
    FROM emp
    WHERE deptno = :DEPTNO'
  );
  -- set row tag name and row set tag name
  dbms_xmlquery.setRowTag(
    queryCtx
    , 'EMP'
  );
  dbms_xmlquery.setRowSetTag(
    queryCtx
    , 'EMPSET'
  );
  -- bind variables
  dbms_xmlquery.setBindValue(
    queryCtx
    , 'DEPTNO'
    , 10
  );
  -- run query and print out result
  result := dbms_xmlquery.getXml(queryCtx);
  printClobOut(result);
  -- free resources
  dbms_xmlquery.closeContext(queryCtx);
END;
/
```

XML-Speicherung mit XMLType(1)

Ab der Version Oracle9i steht ein neuer Server-Datentyp XML Type bereit, der die Speicherung von XML-Dokumenten oder -fragmenten realisiert und Anfragen auf den so gespeicherten Daten ermöglicht. Das Erzeugen einer Instanz vom Typ XMLType erfolgt mit `sys.XMLType.createXML(xml_as_string)`.

Intern basiert die Speicherung auf CLOBs. Bei Anfragen an die gespeicherten XML-Dokumente können XPath-Funktionalitäten verwendet werden.

```
drop table hotels;
CREATE Table hotels(
name VARCHAR2(40) ,
ort VARCHAR2(35),
beschreibung SYS.XMLTYPE,
ausstattung SYS.XMLTYPE,
preise SYS.XMLTYPE,
telefonnr VARCHAR(30)) ;

delete from hotels;
INSERT INTO hotels (name, ort, beschreibung, ausstattung, preise, telefonnr)
VALUES ('Hotel Neptun', 'Warnemuende',
sys.XMlType.createXML('<beschreibung>In Warnemuende ankommen und sich
zu Hause fuehlen</beschreibung>'),
sys.XMLType.createXML('
<ausstattung>
<schwimmbad>(1)Original-Thalassozentrum. Wir haben fuer Sie frisches
Ostseewasser in das Hotel geholt!</schwimmbad>
<schwimmbad>(2)Original-Kneippzentrum. Wir haben fuer Sie frisches
Ostseewasser in das Hotel geholt!</schwimmbad>
</ausstattung>'),
sys.XMLType.createXML('<Preis><DZ waehrung="Euro">186</DZ> <EZ
waehrung="Euro">135</EZ></Preis>') ,
'0381-54370');
```

XML-Abfragen mit XPath

```
SELECT h.name,  
h.ausstattung.extract('ausstattung/schwimmbad[2]/text()').getClobVa  
l() "Swimmingpool",  
h.preise.extract('/Preis/DZ/text() ').getNumberVal() "Doppelzimmer"  
FROM hotels h where ort='Warnemuende';
```

NAME	Swimmingpool
	Doppelzimmer

Hotel Neptun (2)Original-Kneippzentrum. Wir haben
fuer Sie frisches Ostseewasser in das Hotel 186

1 Zeile wurde ausgewählt.

XML Processing using XERCEs

Using XML to transport information between different systems is nowadays a state of the art technology. Searching the WEB you'll find a huge set of examples how to validate and parse an XML document. But when it comes to the point where you have to create a XML document and send it as stream (i.g. character-stream, string, byte-array, etc.) to another system, it becomes difficult to find a proper example. Furthermore, different implementations of the DOM specification using different class names to do the same thing.

Xerces (named after the Xerces Blue butterfly) provides world-class XML parsing and generation. Fully-validating parsers are available for both Java and C++, implementing the W3C XML and DOM (Level 1 and 2) standards, as well as the de facto SAX (version 2) standard. The parsers are highly modular and configurable. Initial support for XML Schema (draft W3C standard) is also provided.

Aktuelle Version (Stand 11/04): Xerces2 Java Parser 2.6.2

Here comes an example which creates a XML Document using DOM and transforms it to a String. We used the XERCEs Java Parser V1_4_3 from apache.org.

Running the example produces the following output

```
<?xml version="1.0" encoding="UTF-8"?>
  <ProbeMsg>
    <TimeStamp>2001-11-30T09:08:07Z</TimeStamp>
    <Probeld>1A6F</Probeld>
    <ProbeValue ScaleUnit="mm">1245</ProbeValue>
  </ProbeMsg>
```

XERCES – Example 1/3

```
import java.io.*;
import java.util.*;
import java.text.*;
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.apache.xerces.jaxp.DocumentBuilderFactoryImpl;
import org.apache.xerces.jaxp.DocumentBuilderImpl;
import org.apache.xml.serialize.XMLSerializer;
import org.apache.xml.serialize.OutputFormat;

public class ProbeMsg {

    // XML tag's
    private static final String TAG_PROBE_MSG    = "ProbeMsg";
    private static final String TAG_TIMESTAMP    = "TimeStamp";
    private static final String TAG_PROPE_ID     = "Probeld";
    private static final String TAG_PROPE_VALUE = "ProbeValue";

    // XML Settings
    private static final String XML_VERSION      = "1.0";
    private static final String XML_ENCODING     = "UTF-8";

    // Format definitions
    private static final String DATE_TIME_FORMAT =
        "yyyy-MM-dd'T'HH:mm:ss'Z'";

    // Variables
    private Date    msgTimeStamp = null;
    private String  probeld      = "";
    private Integer probeValue   = null;
    private Document xmlDoc      = null;
    private String  xmlStr       = null;

    // Constructor
    public ProbeMsg(Date pTimeStamp
        ,String pProbeld
        ,int pProbeValue ) {

        this.msgTimeStamp = pTimeStamp;
        this.probeld      = pProbeld;
        this.probeValue   = new Integer(pProbeValue);

        // Generate the XML Document using DOM
        this.generateXMLDocument();

        // Generate a XML String
        this.generateXMLString();
    }

    // Retrieve probe message as XML string
    public String getXMLString() {
        return xmlStr;
    }
}
```

XERCES – Example 2/3

```
// Generate a DOM XML document
private void generateXMLDocument()
{
    Element main;
    Element root;
    Element item;
    DateFormat timeStampFormat =
        new SimpleDateFormat( DATE_TIME_FORMAT );

    try {

        //Create a XML Document
        DocumentBuilderFactory dbFactory =
            DocumentBuilderFactoryImpl.newInstance();
        DocumentBuilder docBuilder = dbFactory.newDocumentBuilder();
        xmlDoc = docBuilder.newDocument();
    } catch(Exception e) {
        System.out.println("Error " + e);
    }

    // Create the root element
    root = xmlDoc.createElement(TAG_PROBE_MSG);

    // Add TimeStamp Element and its value
    item = xmlDoc.createElement(TAG_TIMESTAMP);
    item.appendChild(xmlDoc.createTextNode(
        timeStampFormat.format(msgTimeStamp)));
    root.appendChild(item);

    // Add Probeld Element and its value
    item = xmlDoc.createElement(TAG_PROPE_ID);
    item.appendChild(xmlDoc.createTextNode(probeld));
    root.appendChild(item);

    // Add ProbeValue Element and its value
    item = xmlDoc.createElement(TAG_PROPE_VALUE);
    item.appendChild(xmlDoc.createTextNode(probeValue.toString() ));
    item.setAttribute("ScaleUnit", "mm");
    root.appendChild(item);

    // Add to the root Element
    xmlDoc.appendChild(root);
}
```


XERCES – Example 3/3

```
// Generate String out of the XML document object
private void generateXMLString() {

    StringWriter strWriter = null;
    XMLSerializer probeMsgSerializer = null;
    OutputFormat outFormat = null;

    try {
        probeMsgSerializer = new XMLSerializer();
        strWriter = new StringWriter();
        outFormat = new OutputFormat();

        // Setup format settings
        outFormat.setEncoding(XML_ENCODING);
        outFormat.setVersion(XML_VERSION);
        outFormat.setIndenting(true);
        outFormat.setIndent(4);

        // Define a Writer
        probeMsgSerializer.setOutputCharStream(strWriter);

        // Apply the format settings
        probeMsgSerializer.setOutputFormat(outFormat);

        // Serialize XML Document
        probeMsgSerializer.serialize(xmlDoc);
        this.xmlStr = strWriter.toString();
        strWriter.close();

    } catch (IOException ioEx) {
        System.out.println("Error " + ioEx);
    }
}

public static void main (String argv[]) {

    ProbeMsg pMsg = new ProbeMsg(new Date() // Timestamp
        , "1A6F" // Probe ID
        , 1245); // Probe Value
    System.out.println(pMsg.getXMLString() );
}
}
```

www.

w3schools.

com