Access & XML

Mark Dörbandt - Skriptum zum AEK-Vortrag - 2005-10-01 & 2005-10-08

Einordnung

warum X*?

XML ist eine **universelle Austauschsprache**. Verschiedenste Anwendungen können mit XML und den damit verbundenen Technologien strukturiert Daten austauschen. Sicher, das geht auch mit einfachen Textdateien. Doch die im folgenden genannten Aspekte der inzwischen weit fortgeschrittenen – aber immer noch heftig im Gange befindlichen – Entwicklung von XML & Co. bringen viele Vorteile, so daß man ernsthaft erwägen sollte, nun endlich in das Thema einzusteigen, so man nicht sowieso schon lange dabei ist!

XML, XSD, XPath, XSLT und viele weitere sind Standards. Das W3C hat diverse sog. "Empfehlungen" (recommendations) verabschiedet, die de facto aber gut **dokumentierte**, **offene Standards** sind [W3C]. Die Verabschiedung des XML-Standards erfolgte bereits 1997, also vor knapp acht Jahren.

XML ist eine Sprache, genauer gesagt eine Auszeichnungssprache (markup language). Als **Metasprache** ist grundsätzlich nur die Syntax sog. wohlgeformter XML-Dokumente definiert. XML wird in diversen Branchen genutzt und zwar z.B. für die Konfiguration von Anwendungen, das Speichern von Dokumenten oder das Übertragen von Daten, z.B. über das Internet.

Ein wesentlicher Vorteil ist das Vorliegen umfangreicher **Tools** und **Bibliotheken** sowie eingebauter **Funktionen** in vielen Anwendungen, die XML unterstützen. Damit hat man als Entwickler – meist sogar kostenlos – mächtige Werkzeuge an der Hand, um schnell und effizient zum Ziel zu gelangen.

Besonders gelobt wird die **Lesbarkeit** von XML: XML ist Text! Dabei ist die Lesbarkeit durch den Menschen sicherlich praktisch, die Lesbarkeit durch die Maschine aber wesentlich wichtiger. Damit wird eine automatisierte Weiterverarbeitung der Dokumente ermöglicht und gleichzeitig eine **Mehrfachverwendung**, z. B. eines einzigen XML-Dokumentes für die Anzeige in einer HTML-Seite, den Ausdruck als PDF-Datei und die Weitergabe an eine andere Applikation.

XML-Dokumente enthalten neben den Daten auch Informationen zur **Struktur** – das ist die sogenannte Auszeichnung. Die weit verbreitete These, XML sei **selbstbeschreibend**, ist aber nur bedingt gültig, denn die Bedeutung der Daten erschließt sich erst in der Anwendungslogik bzw. in der Interpretation durch den Menschen.

Dennoch ist ein großer Vorteil der X-Familie darin zu sehen, daß man die Inhalte der Dokumente über Schemas (XSD) beschreiben kann. Mit einem Schema kann man die für einen bestimmten Anwendungsbereich erlaubten oder vorgeschriebenen Inhalte eines XML-Dokumentes sehr genau definieren. Für viele Anwendungsbereiche gibt es bereits **vorgefertigte Schemata**, die bspw. branchenweit vereinbart wurden. Neben der **Wohlgeformtheit** von XML-Dokumenten anhand korrekter Syntax kann man mit solchen Schemata dann auch die **Gültigkeit** für einen bestimmten Anwendungsbereich prüfen.

Zum Thema Sicherheit wird oft gelobt, daß XML – als Text-Datei oder -Strom – grundsätzlich **Firewall**-tauglich sei. Abgesehen davon, daß Administratoren über content filtering hier auch leicht einen Riegel vorschieben können, ist das sicher richtig. Solange nur Daten im XML enthalten sind, kann man es wohl auch als reinen Vorteil werten.

Die einfache Lesbarkeit wird z. T. auch als Sicherheitsrisiko gewertet. Dazu ist natürlich auf einfache und gängige Techniken wie **Verschlüsselung** (https: oder auch VPN) zu verweisen. Auch die Argumentation, XML-Dateien würden wegen ihrer Größe zu viel Transportvolumen nutzen, ist mit einem Hinweis auf die gute **Komprimierbarkeit** von XML-Dateien abgetan.

XML enthält zwar Struktur und Daten, in der Regel aber bewußt keine Informationen zu Layout und Darstellung derselben. Diese **Trennung von Information und Design** hat sehr große Vorteile, denn eine Aufbereitung derselben Daten für unterschiedliche Anzeigezwecke wird sehr einfach. Auch muß bei Änderungen der Daten nicht jedesmal das Layout angepaßt werden, wenn eine generische Darstellung verwendet werden kann.

Darüber hinaus bietet XSLT als Mitglied der "X-Familie" eine sehr mächtige Sprache für **Transformationen** von XML-Dateien wieder nach XML, nach HTML, nach Text oder sogar hin zu Binär-Dateien.

XML und Datenbanken

Wenn XML so viele Vorteile bietet, wofür braucht man dann noch Datenbanken? Zunächst muß betont werden, daß mit XML in erster Linie Dokumente oder auszutauschende Datenmengen in Dokumentform beschrieben werden.

Ein wesentlicher Unterschied zwischen typischerweise in XML-Dateien gespeicherten Informationen und solchen, die in (relationalen) Datenbanken abgelegt werden, liegt in ihrer **Strukturiertheit**.

XML-Dokumente können sehr gut und flexibel **hierarchische** Strukturen abbilden, das relationale Modell dagegen ist prinzipiell "flach". Natürlich lassen sich auch in relationalen Datenbanken Baumstrukturen abbilden, aber häufig muß schon bei geringen Abweichungen das Datenmodell angepaßt werden.

Gerade solche Abweichungen sind aber in **semistrukturierten** Dokumenten eine häufige Erscheinung. Damit eignet sich XML besonders gut für deren Abbildung.

Als Anmerkung: in diesem Sinne sind klassische Textdokumente völlig unstrukturiert. Eine wesentliche Idee **semantischer Netze** ist z. B. gerade die Auszeichnung von Detailstrukturen in solchen Volltexten, um Dokumente mit Meta-Informationen anzureichern und damit besser recherchierbar zu machen.

Die Spezifikation und Einhaltung von **Datentypen** für die zu verarbeitenden Daten ist übrigens kein Unterscheidungskriterium mehr: mit XML Schema lassen sich Datentypen und deren Gültigkeitsregeln sehr detailliert beschreiben.

Im Gegensatz zu XML eignen sich relationale Datenbanken sehr gut dazu, **atomare Massendaten** mit vielen **Referenzen** untereinander abzuspeichern.

Das Vorhandensein einer **Datenbank-Engine**, die optimiert ist für den schnellen Zugriff auf viele Daten, begründet wesentliche Stärken der Datenbank gegenüber XML: Stichworte wie Indizierung, Protokollierung, Transaktionen, Replikation oder Trigger seien genannt.

Für die Verarbeitung von XML-Dateien liegen zwar auch Tools und Bibliotheken vor, diese sind in der Regel jedoch nicht für große Datenmengen optimiert und bieten viele der o. g. Funktionen einer Datenbank-Engine gerade nicht.

Genannt seien hier die weit verbreiteten DOM und SAX, die beide eine XML-Datei einlesen und eine Art "**In-Memory-Datenbank**" nutzen, um Operationen darauf auszuführen – mit einem komplexeren oder einfacheren Objektmodell.

Die **Änderungshäufigkeit** der betroffenen Daten ist neben der Menge derselben ein weiteres Kriterium zur Unterscheidung. Während sich in relationalen Datenbanken sehr viele Werte sehr häufig ändern können, sind XML-Dateien wegen des Dokumenten-Charakters eher als statisch anzusehen.

Resümee: naheliegend ist also, eine **Kombination** beider Technologien XML und relationale Datenbanken anzuwenden, um die Vorteile aus beiden Welten effizient nutzen zu können.

Moderne relationale Datenbanksysteme bieten die Möglichkeit, **XML** auch **in** den **Datenbanken** selbst zu nutzen. So bietet der neue SQL Server 2005 z.B. einen eigenen Datentyp "XML" an und entsprechende Suchfunktionen.

In Access wird man für die Speicherung von XML meist Memo-Felder nutzen.

"SQL in – XML out": auch die Möglichkeiten, XML als Ergebnismenge aus Datenbanken zu erhalten oder als Input in Datenbanken einzulesen, werden immer besser. So bietet z.B. der SQL Server schon länger die Klausel "FOR XML" und auch in Access werden die Import- und Export-Funktionen von Version zu Version besser.

XML in Access

Access 97

Die eigentlich ja sehr gute Hilfe zu Access 97 findet bei der Suche nach dem Stichwort "XML" leider genau gar nichts - "damals" war XML noch nicht "en vogue".

Import und Export bieten keine Option für XML. Allerdings ist es natürlich möglich, bei Vorliegen der entsprechenden Bibliotheken per VBA XML-Dateien programmgesteuert einzulesen oder zu erzeugen.

Für den **Import** lohnt es, die MSXML-Bibliotheken (sprich: DOM) zu benutzen. Das einfache Codebeispiel 1 zum Befüllen eines TreeView findet sich unten.

```
Dim XMLDoc As MSXML.DOMDocument

Private Sub Command1_Click()
  Set XMLDoc = New MSXML.DOMDocument
  XMLDoc.async = False
  XMLDoc.Load Me!Text1
  If XMLDoc.parseError.errorCode = 0 Then
    MsgBox "Succeeded"
    If XMLDoc.readyState = 4 Then
      Me!TreeView1.Nodes.Clear
      AddNode XMLDoc.documentElement
    End If
  Else
    MsgBox XMLDoc.parseError.reason & vbCrLf & XMLDoc.parseError.Line & vbCrLf & _
           XMLDoc.parseError.srcText
  End If
End Sub

Private Sub AddNode(ByRef XML_Node As IXMLDOMNode, Optional ByRef TreeNode As Node)
  Dim xNode As Node
  Dim xNodeList As IXMLDOMNodeList
  Dim i As Long
  If TreeNode Is Nothing Then
    Set xNode = Me!TreeView1.Nodes.Add
  Else
    Set xNode = Me!TreeView1.Nodes.Add(TreeNode, tvwChild)
  End If
  xNode.Expanded = True
  xNode.Text = XML_Node.nodeName
  If xNode.Text = "#text" Then
    xNode.Text = XML_Node.nodeTypedValue
  Else
    xNode.Text = "<" + xNode.Text + ">"
  End If
  Set xNodeList = XML_Node.childNodes
  For i = 0 To xNodeList.length - 1
    AddNode xNodeList.Item(i), xNode
  Next
End Sub
```

Codebeispiel 1: Befüllen eines TreeView (nach KB-Artikel 244954)

```
Public Function SaveToXML(strFile As String) As Boolean
  Dim db As DAO.Database
  Dim rs As DAO.Recordset
  Dim FNr As Long, i As Long
  Set db = CurrentDb
  Set rs = db.OpenRecordset("Bestellung", dbOpenSnapshot, dbForwardOnly)
  If rs.EOF Then Exit Function
  FNr = FreeFile
  Open strFile For Output As #FNr
  Print #FNr, "<?xml version=""1.0"" encoding=""windows-1252""?>"
  Print #FNr, "<files>"
  Do Until rs.EOF
    Print #FNr, vbTab & "<file type=""Bestellung"" id=""" & rs![id] & """>"
    Print #FNr, vbTab & vbTab & "<title>" & rs![Title] & "</title>"
    For i = 2 To rs.fields.count - 1
      If rs(i).Name = "Dokumente" Then
        Print #FNr, vbTab & vbTab & "<document register=""Dokumente"">" & _
                                    Nz(rs(i).Value) & "</document>"
      Else
        Print #FNr, vbTab & vbTab & "<field name=""" & rs(i).Name & """>" & _
                                    Nz(rs(i).Value) & "</field>"
      End If
    Next i

    Print #FNr, vbTab & "</file>"
    rs.MoveNext
  Loop
  Print #FNr, "</files>"
  Close #FNr
  rs.Close: Set rs = Nothing
  Set db = Nothing
  SaveToXML = True
End Function
```

Codebeispiel 2: Erzeugen einer XML-(Text-)Datei (Abwandlung eines Beispiels von Jörg Ackermann)

Dagegen wird man beim **Export** von Daten nach XML häufig einfach Textdateien erzeugen und die benötigten Tags im Text ausschreiben. Das Codebeispiel 2 oben zeigt, wie es geht.

Die MSXML-Bibliotheken sind auch sehr nützlich, wenn man XML-Dateien transformieren möchte (z.B. mit der DOM-Methode `.transformNodeToObject`).

DAO als Objektbibliothek bietet keine XML-Funktionalitäten an.

Access XP

Access XP war die erste Version mit nennenswerter XML-Unterstützung. Aus jetziger Sicht waren die Funktionen jedoch nur "halbgar". Sprich: einiges, was man sich wünschen würde, gab es noch nicht in ausgereifter Form.

Es ist möglich, XML-Dateien in Access XP zu importieren und XML-Dateien aus Access XP heraus zu exportieren. Allerdings ist das Schema der Dateien bei Import und Export starr.

Beim **Export** kann man die Daten, das Schema und auch eine Präsentation der Daten (im HTML-Format) erzeugen. Die erzeugte XML-Datei hat das Format

```
<dataroot>
  <Tabelle>
    <Feld1>Wert</Feld1>
    <Feld2>Wert</Feld2>
    <Feld3>Wert</Feld3>
  </Tabelle>
</dataroot>
```

und ist damit elementzentriert aufgebaut. Leider kann man hier keine Variation der Struktur hin zu einem gewünschten Zielformat angeben.

Die erzeugte Schema-Datei beschreibt die Datentypen der exportierten Tabelle oder Abfrage

```
<xsd:element name="Einzelpreis"
  minOccurs="0"
  od:jetType="currency"
  od:sqlSType="money"
  type="xsd:double"/>
```

und referenziert dabei u. a. auf

```
urn:schemas-microsoft-com:officedata
```

womit neben den XSD-Datentypen auch die spezifischen Datentypen für Access und SQL Server angegeben werden, was sehr praktisch sein kann.

Die beim Export erzeugte XSL-Datei und die daraus generierte HTML-Datei sind als nur begrenzt nützlich zu bewerten. Das Stylesheet benutzt im wesentlichen die Direktive <xsl:script> und damit ein JavaScript zum Erzeugen der HTML-Datei.

Beim **Import** wird ebenfalls eine elementzentrierte XML-Datei erwartet, andere Strukturelemente, insbesondere sämtliche Attribute werden ignoriert (siehe KB-Artikel 285329).

Problematisch sowohl beim Export als auch beim Import kann sein, daß die eher statische Struktur der Access-XML-Dateien nicht dem gewünschten Quell- oder Zielformat genügt. Hier helfen Transformationen, die man allerdings mit VBA (und DOM) ausprogrammieren muß. Es sind zwar meist nur wenige Zeilen Code erforderlich, aber ohne VBA geht das in Access XP leider nicht.

Natürlich kann man in Access XP auch die schon für Access 97 gezeigten VBA-Funktionen nutzen, um XML-Dateien in MSXML-DOM-Strukturen einzulesen, zu verarbeiten oder um XML-Dateien zu erzeugen.

Mit der **ADO**-Bibliothek (ab Access 2000, per Verweis auch in Access 97 nutzbar) gibt es die Möglichkeit, Recordsets direkt als XML persistent zu speichern und wieder zu laden:

```
rs.Save "Export.xml", _
  adPersistXML
rs.Open "Export.xml", _
  "Provider=MSPersist;"
```

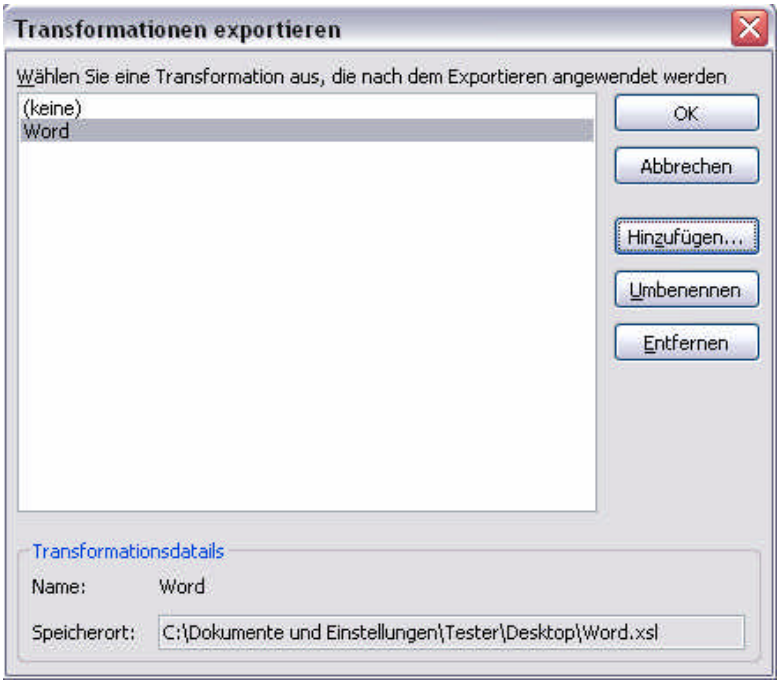
Die so erzeugten XML-Dateien können natürlich auch für andere Zwecke, etwa Web-Publishing weiter verwendet werden.

Generell kann man sagen, daß zur Zeit der Entwicklung von Access XP der Grundgedanke bei der Verarbeitung von XML-Dateien noch die "Nutzung für das **Internet**" war. Dieser Gedanke greift jedoch zu kurz. Inzwischen hat auch Microsoft – nicht zuletzt getrieben von der eigenen .net-Entwicklung – XML als universelles Format entdeckt, das überall und in allen Anwendungen auftaucht.

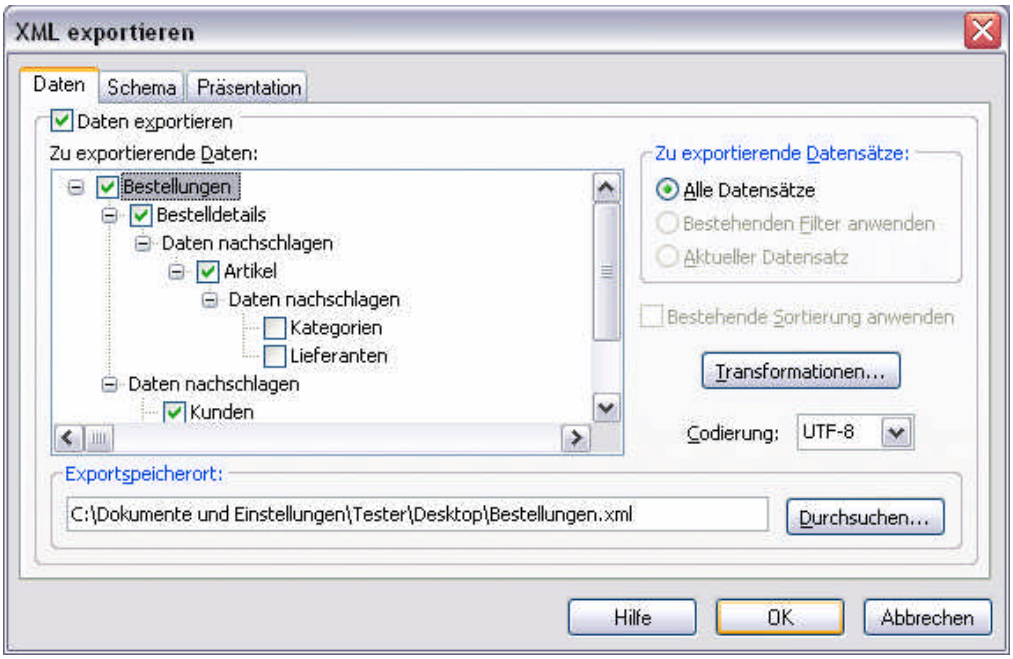
Access 2003

Die vorgenannten Fähigkeiten der älteren Access-Versionen können natürlich auch in Access 2003 genutzt werden.

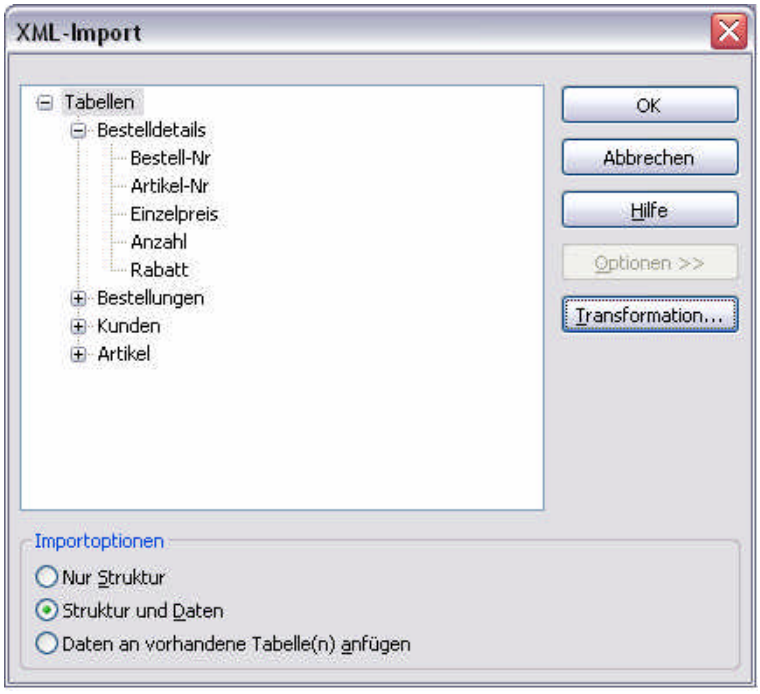
Die erste wesentliche Neuerung in Access 2003 ist die Erweiterung von Import und Export um **Transformationen**.



Damit ist eine flexible Überführung der real vorliegenden Quelldateien in das gewünschte Importformat ebenso möglich wie das Exportieren nahezu aller Dateiformate, denn XSL-Transformationen können als Ziel neben XML auch HTML oder Text, ja sogar Binärformate haben.



Die zweite wesentliche Neuerung ist die Möglichkeit, beim Import und Export **abhängige Tabellen** einzubeziehen.



Dazu werden beim Export die in der Datenbank vorhandenen Beziehungen genutzt und ggf. hierarchische XML-Strukturen erzeugt. Dabei sind die erzeugten Kindelemente aus den Detailtabellen der Datenbank passend zum jeweiligen Elternelement gefiltert.

Beim Import wird die XML-Datei analysiert und es wird in einem Dialog angezeigt, welche abhängigen Tabellen in der XML-Datei vorhanden sind. Damit lassen sich auch umfangreichere Datenbank-Strukturen sinnvoll in einer einzelnen XML-Datei ablegen. Sinnvoll wäre hier eine Selektion der zu importierenden Tabellen analog zum Export - dies ist momentan nur mit einer Transformation möglich.

Insbesondere die Kombination aus Transformation und Nutzung hierarchischer XML-Strukturen ist sehr nützlich. Die Mächtigkeit dieser Änderungen wird sich in den Praxisbeispielen zeigen.

Für den Entwickler interessant ist außerdem die Möglichkeit der Nutzung der neuen bzw. erweiterten **Methoden**

```
Application.TransformXML
Application.ImportXML
Application.ExportXML
```

Diese ermöglichen auch programmgesteuert den Import und Export und auch vorbereitende oder nachbereitende Transformationen.

Leider sind diese Methoden ausschließlich Datei-basiert – eine Nutzung für XML-Strings muß man sich zunächst selber kapseln. Ein Beispiel hierzu findet sich in der Demo-Datenbank.

Theorie

Es gibt sehr reichlich frei zugängliche Ressourcen zur Theorie von XML & Co. Dieser Vortrag wird daher nur einen kurzen Abriß der wesentlichen Grundlagen liefern. Details lassen sich jederzeit im Internet (siehe Abschnitt Links) und in der Literatur (siehe Abschnitt Bücher) finden.

XML

XML-Dateien bestehen aus Elementen, Attributen, Verarbeitungsanweisungen und Kommentaren.

Ein **Element** besteht aus einem Start-Tag und einem Ende-Tag:

```
<Elementname>Inhalt</Elementname>
```

Leere Elemente sollen im Starttag abgekürzt werden:

```
<Elementname/>
```

Elemente können weitere Elemente beinhalten:

```
<Element> gemischter Inhalt
  <Element2>
  Inhalt Element 2
  </Element2>
</Element>
```

Attribute können bei einem Starttag geschrieben werden und haben einen Namen und einen Wert:

```
<Element
Attributname="Attributwert">
Inhalt
</Element>
```

Verarbeitungsanweisungen liefern Zusatzinformationen zur Datei und können (Pseudo-) Attribute haben:

```
<?xml-stylesheet href="my.css" ?>
```

Kommentare dienen der besseren Lesbarkeit:

```
<!-- Kommentar -->
```

Eine **XML-Datei** hat genau ein Wurzelemelement. Zu Beginn der Datei steht meist, aber nicht verpflichtend die **XML-Deklaration**:

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

Namensräume

Da Elementnamen zunächst beliebig sind, bei einer Vielzahl unterschiedlicher XML-Anwendungen aber Dopplungen wahrscheinlich sind, hat man eine Identifizierung von Namen über **Präfixe** definiert.

Im Beispiel

```
<xsl:template>
```

liegt damit nicht irgendein Template vor, sondern ein XSL-Template, da zu Beginn der Datei das Präfix `xsl:` definiert wurde:

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/TR/WD-xsl" >
```

Das Präfix `xmlns:` selbst ist für die Definition von Namensräumen (namespaces) reserviert.

XML-Schema

Mit einer Schema-Datei werden die in einer XML-Datei zulässigen und erforderlichen Datentypen und Strukturen definiert.

Eine XML-Datei ohne Schema kann wohlgeformt sein. Ihre semantische **Gültigkeit** kann man jedoch nicht überprüfen.

Daher kann man innerhalb oder außerhalb einer XML-Datei auf ein zugehöriges Schema verweisen, mit dessen Hilfe der XML-Prozessor die Gültigkeit des Dokumentes testen kann. Dies ist für die weitere fehlerfreie Verarbeitung des Dokumentes enorm wichtig.

Im einzelnen können alle gängigen **Datentypen** definiert werden sowie eigene Datentypen nahezu beliebig konstruiert werden.

Die **Struktur** des XML-Dokumentes mit seinen Verschachtelungen, Reihenfolgen und Vorkommenshäufigkeiten von Elementen und Attributen können sehr genau festgelegt werden.

XPath

XML-Dokumente können hierarchisch komplex sein. XPath ermöglicht eine **Navigation** innerhalb solcher Dokumente im Sinne exakter Positionsangaben für einzelne oder Mengen von Elementen oder Attributen.

Die Grundidee ist dabei, das XML-Dokument als Baum aufzufassen, in dem man jeden einzelnen Knoten mit einem bestimmten **Pfad** adressieren kann. Ausgangspunkt ist dabei entweder das Wurzelelement des Dokumentes oder der jeweils betrachtete Knoten (Kontext).

Für die relative Navigation ausgehend von einem einzelnen Knoten gibt es verschiedenen sog. **Achsen**, die z.B. die Kinder, Geschwister oder Eltern des Knotens adressieren. Außerdem sind zur Filterung von Knoten sog. **Knotentests** und **Prädikate** möglich.

Hierzu einige Beispiele:

```
/dataroot
/dataroot/Tabelle[@name="Test"]
/dataroot/Tabelle/Element1
/dataroot/Tabelle[1]
//Element2[Inhalt]
```

- im Rahmen der Praxisbeispiele werden einige konkrete Beispiele erläutert.

XSLT

Transformationen sind das "Schweizer Taschenmesser" der XML-Familie. Sie dienen dazu, XML-Dokumente ineinander zu überführen - und zwar nahezu beliebig.

Für den **Datenaustausch** kann man mit XSLT XML-Dokumente in anders strukturierte XML-Dokumente transformieren.

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

Die zweite wesentliche Aufgabe für Transformationen ist die **Präsentation** in Richtung HTML oder PDF (mit XSL-FO).

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

Das Grundprinzip der Transformationen basiert auf **Templates**. Jedes Template repräsentiert einen Baustein, der für eine bestimmte Knotenmenge Teile der Ausgabe erzeugt. Der XSLT-Prozessor arbeitet das Dokument vom Wurzelknoten aus ab. Sinnvollerweise enthält daher jedes Stylesheet ein Template für den Wurzelknoten.

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

Innerhalb dieser Templates gibt es vielfältige **Bausteine**, um z. B. Bedingungen, Schleifen, XML-Elemente und XML-Attribute zu erzeugen.

```
<xsl:if>
<xsl:for-each>
<xsl:element>
<xsl:attribute>
```

Neben diesen Bausteinen gibt es eine Vielzahl von **Funktionen**, die darin genutzt werden können, z. B. zur Positionsbestimmung innerhalb des Baumes, zur Zeichenkettenbearbeitung oder zur mathematischen Bearbeitung.

```
position()
substring()
sum()
```

Außerdem können innerhalb von Stylesheets **Variablen** verwendet werden.

```
<xsl:variable>
```

XQuery

Während XSLT für die Transformationen kompletter XML-Dokumente gedacht ist, kann man mit XQuery Teile aus XML-Dokumenten abfragen.

Basis dafür sind sog. FLOWR-Ausdrücke (for, let, order by, where, return), die die Nähe zur Abfragesprache SQL zeigen.

Im Rahmen dieses Vortrages wird XQuery nicht benutzt.

LINQ

Sehr aktuell hat Microsoft in diesem Zusammenhang das Projekt LINQ vorgestellt, bei dem sprachintegriert in die modernen .net-Sprachen Abfrageelemente sowohl für relationale als auch für XML-Abfragen integriert werden sollen [LINQ].

Praxis

Die im folgenden kurz diskutierten Anwendungs-Beispiele bilden den Hauptteil des Vortrages und finden sich sämtlich in der zum Vortrag gehörigen **Demo-Datenbank**. Diese wird im Anschluß an die AEK zum Download bereitgestellt.

Hier wird jeweils kurz auf das **Anwendungsszenario** eingegangen, das den Beispielen zugrunde liegt. Die Beispiele zeigen einen Weg für eine Lösung auf und können daher als Schablonen für eigene Lösungen verwendet werden. Ein Anspruch auf Vollständigkeit oder Perfektion wird allerdings nicht erhoben.

SysInfo

Szenario: ein Administrator möchte für eine Vielzahl von Rechnern Systeminformationen systematisch verwalten. Das zum Betriebssystem gehörige Tool MSInfo32.exe liefert die gewünschten Informationen.

Die erzeugten .nfo-Dateien haben XML-Format, so daß mit einer einfachen Transformation ein Import nach Access möglich ist.

Word

Szenario: es soll ein Bericht in Word erstellt werden. Das Ziel dabei ist, das Word-Dokument direkt aus Access heraus als Bericht zu erzeugen – mit sämtlichen Daten und Formatierungen.

Seit Office 2003 kann man Word-Dokumente als XML-Datei abspeichern. Das zugehörige WordML-SDK findet man unter [WordML].

Excel

Szenario: umfangreiche Daten sollen für statistische Zwecke nach Excel exportiert werden, incl. verschiedener Berechnungs-Formeln. OLE-Automatisierung ist dafür zu langsam.

Seit Office 2003 kann man Excel-Dateien als XML-Datei abspeichern. Informationen dazu findet man unter [OfficeXML].

mySQL

Szenario: viele Web-Provider bieten als Web-Datenbank im Rahmen von LAMP-Installationen eine MySQL-Datenbank an, ohne jedoch den Zugriff per ODBC zuzulassen (aus Sicherheitsgründen). Wie kann man nun UPDATEs seiner Daten an eine solche MySQL-Datenbank senden?

HTML

Szenario: sich wöchentlich ändernde Daten sollen über XML mit Hilfe eines Stylesheets für die Veröffentlichung als Kreuztabelle im Web als HTML formatiert werden.

Die Exportfunktionen von Stylesheets nach HTML sind durch die vorgegebenen Templates und die spezielle Ausgabe-Direktive besonders einfach.

Google Earth

Szenario: im Rahmen einer Kundendatenbank liegen Orts- und Postleitzahl-Informationen vor. Diese sollen unter Nutzung einer Geo-Datenbank bei Google-Earth visualisiert werden.

Google-Earth nutzt für die Verwaltung von Favoriten XML-Dateien (mit der Endung .kml). Diese zu erzeugen ist leicht, wenn man die Geo-Koordinaten kennt. Hierzu wird ein Web-Aufruf bei einer frei verfügbaren Geo-Datenbank genutzt.

Verweise

Bücher

Der Autor ist ein Liebhaber von englischen Fachbüchern aus dem O'Reilly-Verlag – und das nicht nur wegen der schönen Tierbilder auf dem Umschlag!

"**XML in a nutshell**" von Elliotte Rusty Harold und W. Scott Means ist ein guter Überblick und Einstieg in das Thema. Alle relevanten Bereiche werden angeschnitten und ein Referenzteil rundet das Buch ab.

"**XML Schema**" von Eric van der Vilst gibt einen tieferen Einblick in die Erstellung und Verwendung von Schemata.

"**XSLT**" von Doug Tidwell ist unverzichtbar, wenn man nicht ständig mit Alt-Tab zwischen z. B. der W3C-Spezifikation und dem gerade bearbeiteten Stylesheet wechseln möchte. Ein sehr gutes Buch mit praktisch nutzbaren Beispielen und einer guten Referenz.

Grundsätzlich kann aber die Behauptung gewagt werden, daß man kein Buch kaufen muß, um XML "zu lernen" – die Web-Ressourcen (aktualisierte Liste unter [dIT-XML]) sind wie schon erwähnt hervorragend!

Glossar

DOM	Document Object Model
HTML	HyperText Markup Language
http:	HyperText Transfer Protocol
https:	http: Secure
KB	Knowledge Base (von MS)
LAMP	Linux - Apache - MySQL - PHP
MS	Microsoft
MSXML	MS-Implementierung einer XML-Bibliothek
MySQL	ein spezielles Datenbanksystem
ODBC	Open DataBase Connectivity
PDF	Portable Document Format
PHP	PHP Hypertext Preprocessor
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
VBA	Visual Basic for Applications
VPN	Virtual Private Networks
W3C	WWW-Consortium
XML	Extensible Markup Language
XSD	XML-Schema Definition
XPath	XML-Path language
XLink	XML-Linking language
XSL	XML-Stylesheet language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformations

Links

[W3C]
http://www.w3c.org

[KB]
http://support.microsoft.com/search/?adv=1

[WordML]
http://msdn.microsoft.com/library/en-us/WordXMLCDK/html/WelcomeWordCDK_HV01147170.asp

[OfficeXML]
http://www.microsoft.com/office/xml/default.mspx

[LINQ]
http://msdn.microsoft.com/netframework/future/linq/

[dIT-XML]
http://www.doerbandt.de/XML/
http://www.doerbandt.de/XML/Links mit umfangreicher Linkliste