XML, XSLT, VB und ASP
Praktisches XML-Wissen für Webprojekte
Elmar Geese, Markus Heiliger, Matthias Lohrer
Galileo Computing
350 S., 2001, geb., mit CD
34,90 Euro, ISBN 3-89842-109-0

2 Volldampf voraus: Ein XML-, XPath-, XSLT-Crashkurs

Sind Sie ungeduldig? Wollen Sie ohne große Vorreden jetzt gleich XML mit Visual Basic bearbeiten? Dann ist dieser Crashkurs genau das richtige für Sie. Mit kleinen Visual-Basic-Programmen bearbeiten Sie XML-Dokumente. Sie setzen XPath und XSLT ein.

In diesem einleitenden Crashkurs werfen Sie einen ersten Blick auf die Grundlagen des XML-Programmierhandwerks. Sie werden lernen, wie Sie

- eine XML-Datei in Visual Basic (VB) laden, bearbeiten und speichern,
- auf einzelne Elemente einer XML-Datei gezielt zugreifen,
- eine Transformation eines XML-Dokuments mit XSLT-Stylesheets durchführen,
- die Ursachen typischer Fehlermeldungen erkennen und beseitigen.

Auf diese Weise erarbeiten Sie sich rasch einige grundlegende Programmgerüste, die Sie für Ihre eigenen Programmierexperimente verwenden können. In den folgenden Kapiteln vertiefen Sie kontinuierlich Ihr Wissen.

2.1 Systemvoraussetzungen

Bevor es richtig losgehen kann, müssen Sie Ihre Systemvoraussetzungen überprüfen. Auf Ihrem Windows-PC benötigen Sie:

- einen installierten Internet Explorer 5.0 (oder höher)
- eine Visual-Basic-Entwicklungsumgebung (für das Einüben reicht auch der Visual Basic Editor aus, wie er unter anderem auch in Microsoft Word 97 oder Word 2000 enthalten ist)
- den XML-Parser msxml 3.0 (oder höher)

Der XML-Parser ist die Softwaremaschine, die XML-Daten verarbeiten kann. Zum Zeitpunkt, als dieses Kapitel erstellt wurde (Januar 2001), war der msxml 3.0 die aktuelle Version des Microsoft-XML-Parsers. Diese Version war bis zu diesem Zeitpunkt ausschließlich über das Internet zu beziehen (*http://msdn.microsoft.com/xml/general/xmlparser.asp*). mxsml

3.0 ist bis zu diesem Zeitpunkt noch mit keiner Version von Windows oder des Internet Explorers zusammen ausgeliefert worden, auch nicht mit dem Internet Explorer 5.5.

Download von msxml 3.0 meistens erforderlich

Daher ist es ziemlich wahrscheinlich, daß auch Sie sich den aktuellen XML-Parser und die zugehörige Dokumentation per Download besorgen und auf Ihrem PC installieren müssen.

Beachten Sie: Wenn Sie sich nicht den msxml 3.0-Parser besorgen und auf Ihrem PC installieren, werden Sie die Beispiele in diesem Kapitel und viele Beispiele im gesamten Buch nicht nachvollziehen können!

Vielleicht fragen Sie sich jetzt: Warum dieser Aufwand? Hätte für die ersten Schritte eine ältere Version des Parsers nicht ausgereicht? Das grundlegende Problem wurde im Eingangskapitel über den Frust mit XML geschildert. Die Funktionalität von früheren Versionen des Parsers ist so stark eingeschränkt, daß Sie damit nur wenig Freude haben werden. Erst die Version 3 bietet eine volle Implementierung der Standards des W3C in den Bereichen *Extensible Stylesheet Language Transformations* (XSLT) und *XML Path Language* (XPath). Das ist der entscheidende Vorteil der Version 3, und er wiegt schwer. Erst jetzt können Sie eine beliebige XML-/XSLT-/XPath-Dokumentation der W3C-Standards aufschlagen und sich sicher sein, daß Ihr Parser diese Daten auch korrekt verarbeitet. Ältere Versionen des msxml-Parsers setzen die Spezifikation nur teilweise um.

Also los: ab ins Internet zu dieser Web-Adresse:

http://msdn.microsoft.com/xml/general/xmlparser.asp

Dort finden Sie den msxml 3.0 und eine entsprechende Dokumentation. Die laden Sie auf jeden Fall auch herunter. Erst danach können Sie hier weitermachen (siehe Abbildung 2.1).

Ja, wir wissen, daß es lästig ist, ganz am Anfang erst etwas installieren zu müssen, aber wir garantieren Ihnen: Der geringe Installationsaufwand am Anfang lohnt sich. Es ist auf jeden Fall vernünftiger, ganz am Anfang eine halbe Stunde lang neue Software zu installieren, als sich wochenlang mit einem halbfertigen XML-Parser herumzuärgern, der sich an keine Standards hält.

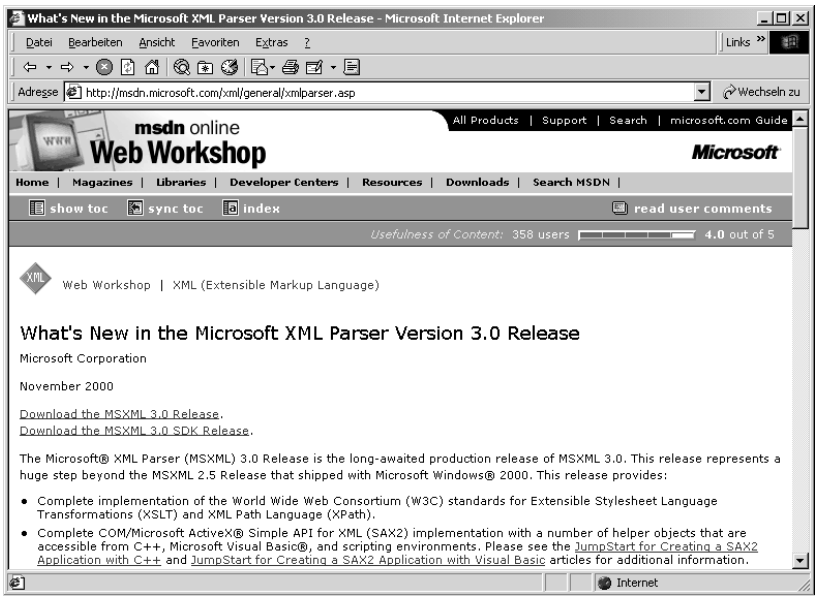

Abbildung 2.1 Erst die Version 3.0 des Microsoft-XML-Parsers ist kompatibel zu den W3C-Standards

2.2 Mit VB eine XML-Datei erstellen

Den ersten Schritt haben Sie jetzt hinter sich gebracht, Sie haben den XML-Parser msxml 3.0 installiert. Im ersten Programmierbeispiel erstellen Sie nun mit Hilfe von VB-Code eine XML-Datei.

Dieses Beispiel wird zweimal durchgearbeitet. Einmal mit dem Visual Basic Editor von Word 2000 und einmal mit Visual Basic 6. Die Unterschiede sind gering. In den späteren Beispielen gehen wir davon aus, daß Sie die Prozeduren selbständig erstellen und ausführen können, und gehen dann nicht mehr auf die kleinen Unterschiede zwischen diesen beiden Entwicklungsumgebungen ein.

2.2.1 So arbeiten Sie mit dem Visual Basic Editor von Word 2000

1. Öffnen Sie Word 2000 und speichern Sie ein neues Dokument unter einem Namen Ihrer Wahl.
2. Öffnen Sie mit dem Menübefehl **Extras · Makro · Visual Basic-Editor** den Visual Basic-Editor (schneller geht es mit Alt F11).

3. Fügen Sie im Visual Basic Editor über den Menübefehl **Extras · Verweise** einen Verweis auf **Microsoft XML, v3.0** hinzu (die zugehörige Datei `msxml3.dll` ist standardmäßig unter `//WINNT/System32` zu finden). Siehe hierzu Abbildung 2.2.

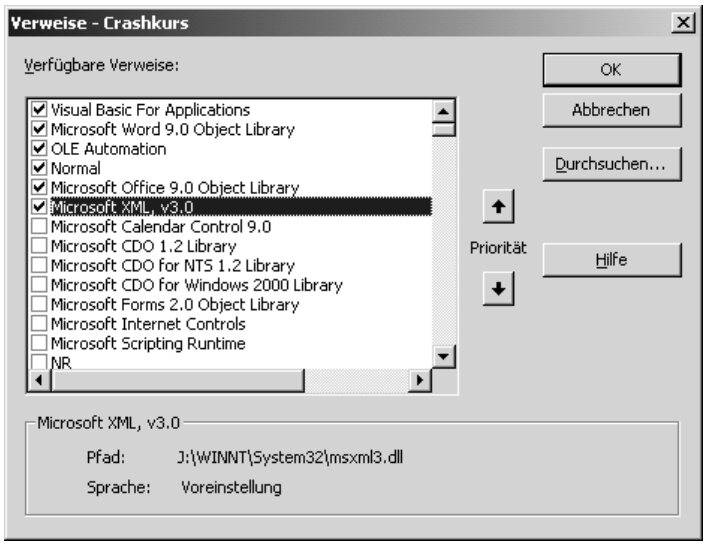

Abbildung 2.2 Der Verweis auf den Microsoft XML-Parser, Version 3.0, ist unbedingt erforderlich

4. Wählen Sie den Befehl **Einfügen · Modul.**
5. Erstellen Sie folgende Prozedur:

```
Sub firstXML()
Dim xml As New DOMDocument
Dim e As IXMLDOMElement
   xml.loadXML "<Kaufhaus/>"
   Set e = xml.createElement("Produkt")
   e.Text = "Mantel"
   e.setAttribute "Farbe", "blau"
   xml.documentElement.appendChild e
   xml.Save "c:\test.xml"
End Sub
```

6. Führen Sie die Prozedur mit dem Menübefehl **Ausführen · Sub · UserForm ausführen** aus.

Weiter unten im Text wird die Prozedur erläutert.

2.2.2 So arbeiten Sie mit Visual Basic 6

1. Öffnen Sie Visual Basic 6.
2. Erstellen Sie über den Menübefehl **Datei · Neues Projekt** ein Standard-EXE-Projekt.
3. Fügen Sie über den Menübefehl **Projekt · Verweise** einen Verweis auf **Microsoft XML, v3.0** hinzu (die zugehörige Datei msxml3.dll ist standardmäßig unter //WINNT/System32 zu finden).
4. Doppelklicken Sie auf das Formular »Form1«. Automatisch finden Sie hier folgenden Code vor:

```
Private Sub Form_Load()
End Sub
```

5. Erstellen Sie hinter **Form_Load()** die Prozedur **firstXML()** und rufen Sie diese Prozedur von **Form_Load()** her auf. Am Schluß lautet der Code:

```
Private Sub Form_Load()
    firstXML
End Sub
Sub firstXML()
Dim xml As New DOMDocument
Dim e As IXMLDOMElement
   xml.loadXML "<Kaufhaus/>"
   Set e = xml.createElement("Produkt")
   e.Text = "Mantel"
   e.setAttribute "Farbe", "blau"
   xml.documentElement.appendChild e
   xml.Save "c:\test.xml"
End Sub
```

6. Führen Sie über den Menübefehl **Ausführen · Starten** das Programm aus. Beim Laden des `Form1`-Fensters wird automatisch die Prozedur `firstXML()` ausgeführt.

2.2.3 Erläuterungen zum Code

Wenn Sie den Code ausführen, wird die Datei `c:\test.xml` erstellt. Wenn Sie diese Datei mit einem Editor, z. B. dem Notepad, öffnen, finden Sie folgenden Inhalt vor:

```
<Kaufhaus><Produkt Farbe="blau">Mantel</Produkt></Kaufhaus>
```

Sie können jetzt also bereits mit Hilfe von Visual Basic XML-Dateien erstellen.

Verweis auf msxml 3.0 unbedingt nötig

Gehen wir die Programmzeilen einzeln durch: Ohne den Verweis auf die Bibliothek `Microsoft XML, v3.0` geht gar nichts. Die Datei `msxml3.dll` enthält den XML-Parser, der für alle Arbeiten mit XML-Daten unbedingt erforderlich ist.

Der Code deklariert zunächst zwei Variablen, eine vom Typ `DOMDocument` und eine vom Typ `IXMLDOMElement`. Das `DOMDocument`-Objekt bietet eine Heimat für ein komplettes XML-Dokument. DOM steht dabei für Document Object Model. Dieses Objektmodell vereinheitlicht den Zugriff auf XML-Daten. Außerdem gibt es ein `IXMLDOMElement`-Objekt.

Die erste Codezeile

```
xml.loadXML "<Kaufhaus/>"
```

weist dem `DOMDocument`-Objekt einen ersten Inhalt zu. Bis jetzt enthält das `DOMDocument`-Objekt nur diesen Text: `<Kaufhaus/>`.

Als nächstes erzeugen Sie ein Element vom Typ »Produkt«. Nach dem Ausführen dieser Zeile

```
Set e = xml.createElement("Produkt")
```

enthält das Objekt `e` sozusagen den XML-Text `<autor></autor>`, aber noch keinen weiteren Inhalt. Mit der Codezeile

```
e.Text = "Mantel"
```

haben Sie den Inhalt von `e` auf `<Produkt>Mantel</Produkt>` erweitert. Als nächstes fügen Sie diesem Element noch ein Attribut hinzu:

```
e.setAttribute "Farbe", "blau"
```

Damit hat `e` jetzt den Inhalt:

```
<Produkt Farbe="blau">Mantel</Produkt>
```

Zuletzt müssen Sie dieses Fragment noch dem Dokument selbst hinzufügen:

```
xml.documentElement.appendChild e
```

Das Dokument `xml` enthält stets genau ein Dokumentelement, das ist die äußere Klammer, in unserem Fall `<Kaufhaus/>`. Mit `appendChild e` fügen Sie das Element `e` dem Dokument hinzu. Damit ist

```
<Kaufhaus><Produkt Farbe="blau">Mantel</Produkt></Kaufhaus>
```

entstanden. Genau diesen String speichern Sie abschließend mit der Zeile:

```
xml.Save "c:\test.xml"
```

Und fertig sind wir.

Wenn Sie jetzt sagen: »Oh, Klasse, das ist aber einfach!«, dann hat dieses Mini-Beispiel bereits seinen Zweck erfüllt. Genau darum geht es in diesem Crashkurs: Sie sollen das Gefühl bekommen, daß Sie mit Visual Basic sehr einfach XML-Daten bearbeiten können.

2.3 Mit XPath in XML-Dokumenten navigieren

Erstellen Sie im Notepad-Editor folgendes XML-Dokument oder laden Sie von der Buch-CD-ROM die Datei CD-LW:\Crashkurs\Kaufhaus.xml:

```
<?xml version="1.0" encoding="iso8859-1"?>
<Kaufhaus>
   <Abteilung Name="Bekleidung">
      <Produkt>
         <Name>Mütze</Name>
         <Farbe>weinrot</Farbe>
         <Preis Währung="DM">50</Preis>
         <!-- eine schöne Mütze -->
      </Produkt>
      <Produkt>
         <Name>Mantel</Name>
         <Farbe>schwarz</Farbe>
         <Preis Währung="Euro">300</Preis>
      </Produkt>
   </Abteilung>
   <Abteilung Name="Lebensmittel">
      <Produkt>
         <Name>Milch</Name>
         <Preis Währung="DM">1</Preis>
      </Produkt>
      <Produkt>
         <Name>Brot</Name>
         <Preis Währung="Euro">3</Preis>
      </Produkt>
      <Produkt>
         <Name>Kaffee</Name>
         <Preis Währung="Euro">8.5</Preis>
      </Produkt>
   </Abteilung>
```

```
</Kaufhaus>
```

iso8859-1 für das Speichern von Umlauten nötig

Beachten Sie hier insbesondere den XML-Prolog ganz am Anfang der Datei:

```
<?xml version="1.0" encoding="iso8859-1"?>
```

Hierbei handelt es sich um eine sogenannte *Processing Instruction*, die vom XML-Parser ausgewertet wird. Processing Instructions sind so etwas ähnliches wie Compiler-Direktiven. Sie sind stets in spitze Klammern mit Fragezeichen `<? ... ?>` gefaßt. Besonders wichtig ist der Bestandteil `encoding="iso8859-1"`. Durch diesen Verweis auf den ISO-Latin-1-Zeichensatz weiß der Parser, wie er Sonderzeichen behandeln soll. Damit können Sie im XML-Dokument problemlos die deutschen Umlaute verwenden.

Dieses XML-Dokument möchten Sie in Visual Basic einlesen und folgende Fragen beantworten: Gibt es im Kaufhaus eine Mütze zu kaufen? Wenn ja, welche Farbe hat sie und was kostet sie?

Hier ist die Lösung: Erstellen Sie folgende Visual-Basic-Prozedur (CD-LW:\Crashkurs\modCrashkurs.bas):

```
Sub sucheMuetze()
Dim XMLDatei As String
Dim ergebnisliste As IXMLDOMNodeList
Dim i As Integer
Dim anzahl As Integer
Dim xml As New DOMDocument
Dim XPATHAusdruck As String
XPATHAusdruck = "//Produkt[Name='Mütze']"
XMLDatei = "L:\galileo\XML-Buch\kap-2-selectnodes.xml"
xml.Load XMLDatei
If (xml.parseError <> 0) Then
   If xml.parseError.reason <> "" Then
      Debug.Print xml.parseError.reason
      Debug.Print xml.parseError.Line
   End If
   Exit Sub
End If
xml.setProperty "SelectionLanguage", "XPath"
Set ergebnisliste = xml.selectnodes(XPATHAusdruck)
anzahl = ergebnisliste.Length
For i = 0 To anzahl - 1
   Debug.Print ergebnisliste.Item(i).xml
```

```
Next
End Sub
```

Passen Sie in der Zeile

```
XMLDatei = "D:\Crashkurs\kaufhaus.xml"
```

die Pfadangabe und den Dateinamen Ihren lokalen Gegebenheiten an. Wenn Sie die Prozedur ausführen, erhalten Sie im Direktfenster die gesuchte Ausgabe:

```
<Produkt>
    <Name>Mütze</Name>
    <Farbe>weinrot</Farbe>
    <Preis Währung="DM">50</Preis>
    <!-- eine schöne Mütze -->
</Produkt>
```

Im Kapitel über XPath wird das Beispiel ausführlich besprochen. An dieser Stelle nur einige kurze Bemerkungen. Die Variable `XPATHAusdruck` enthält den nötigen XPath-Ausdruck. Der hier verwendete Ausdruck

```
//Produkt[Name='Mütze']
```

besagt so viel wie: Suche ein Element mit Namen »Produkt«, das ein Kindelement »Name« mit dem Wert »Mütze« hat. Mit

```
xml.Load XMLDatei
```

wird die XML-Datei eingelesen. Das `parseError`-Objekt ermöglicht die Fehlerbehandlung. Die eigentliche Suche erledigt die Zeile

```
Set ergebnisliste = xml.selectnodes(XPATHAusdruck)
```

Anschließend enthält die Variable `ergebnisliste` eine Liste aller gefundenen Knoten. Deren Inhalte werden mit

```
Debug.Print ergebnisliste.Item(i).xml
```

im Direktfenster ausgegeben. In unserem Fall wird nur ein einziges Element gefunden.

Die Objektbibliothek des Document Object Models und einige nützliche Microsoft-spezifische Erweiterungen lernen Sie in Kapitel 5 genauer kennen. Für den effizienten Einsatz von XML sind gute XPath-Kenntnisse unerläßlich. Diese erwerben Sie in Kapitel 6.

2.4 XML-Dokumente mit XSLT in HTML transformieren

Als drittes und letztes Einführungsbeispiel soll die Transformation eines XML-Dokuments in ein HTML-Dokument mit Hilfe von XSLT durchgespielt werden. Ein Beispiel: Sie möchten die Lebensmittelabteilung des XML-Kaufhauses tabellarisch im Browser darstellen. Hierfür erstellen Sie zunächst eine XSLT-Datei. Eine VB-Prozedur transformiert anschließend die XML-Daten mit Hilfe der XSLT-Datei in ein HTML-Dokument. Hier ist die XSLT-Datei (CD-LW:\Crashkurs\xsltbeispiel.xsl):

```
<?xml version="1.0" encoding="iso8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
<xsl:template match="//Abteilung[@Name='Lebensmittel']">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=iso-
8859-1" />
<title><xsl:value-of select="@Name"/></title>
</head>
<body>
<h1><xsl:value-of select="@Name"/></h1>
<table border="1">
<tr>
<td>Name</td>
<td>Preis</td>
<td>Währung</td>
</tr>
<xsl:for-each select="Produkt">
<tr>
<td><xsl:value-of select="Name"/></td>
<td><xsl:value-of select="Preis"/></td>
<td><xsl:value-of select="Preis/@Währung"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
<xsl:template match="text()|@*">
```

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

Einige Elemente von XSLT sind leicht erkennbar. Offenkundig verfügt auch XSLT über so etwas wie eine `for-each`-Schleife, die in der gewöhnungsbedürftigen Form `<xsl:for-each select="Produkt"> ... </xsl:for-each>` auftritt. Innerhalb der Schleife werden die Tabellenzeilen erstellt. Die variablen Inhalte lassen sich mit Ausdrücken der Art `<xsl:value-of select="Name"/>` einfügen.

Um diese Schleife herum sind die konstanten Elemente wie `<html> ...` etc. aufgeführt. Das Ganze ist in ein Element vom Typ `xsl:template` eingebettet, das im `match`-Attribut anscheinend die Information erhält, auf welches Element es paßt. Lediglich die Bedeutung des zweiten, leeren `xsl:template`-Elements erschließt sich nicht auf Anhieb. Das Kapitel 7 wird dieses Rätsel lösen.

Die Umformung führt diese Visual-Basic-Prozedur durch (CD-LW:\Crashkurs\madCrashkurs.bas):

```
Sub xsltbeispiel()
Dim xmldatei As String
Dim xsltdatei As String
Dim htmldatei As String
Dim xml As New DOMDocument
Dim xsl As New DOMDocument
Dim html As String
Dim fs
Dim a
xmldatei = "D:\Crashkurs\kaufhaus.xml"
xsltdatei = "D:\Crashkurs\xsltbeispiel.xsl"
htmldatei = "C:\ergebnis.htm"
xml.Load xmldatei
If (xml.parseError <> 0) Then
   If xml.parseError.reason <> "" Then
      Debug.Print xml.parseError.reason
      Debug.Print xml.parseError.Line
   End If
   Exit Sub
End If
xsl.Load xsltdatei
If (xsl.parseError <> 0) Then
   If xsl.parseError.reason <> "" Then
```

```
      Debug.Print xsl.parseError.reason
      Debug.Print xsl.parseError.Line
   End If
   Exit Sub
End If
html = xml.transformNode(xsl)
Debug.Print html
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile(htmldatei, True)
a.write html
a.Close
End Sub
```

Den Kern der Prozedur bildet die Zeile

```
html = xml.transformNode(xsl)
```

die die Transformation durchführt. Nach dem Speichern des Strings verfügen wir über die gewünschte HTML-Datei:

```
<html><head>
<meta http-equiv="content-type" content="text/html; charset=iso-
8859-1" />
</head><body><h1>Lebensmittel</h1>
<table border="1">
<tr><td>Name</td><td>Preis</td><td>Währung</td></tr>
<tr><td>Milch</td><td>1</td><td>DM</td></tr>
<tr><td>Brot</td><td>3</td><td>Euro</td></tr>
<tr><td>Kaffee</td><td>8.5</td><td>Euro</td></tr>
</table>
```

Wenn die HTML-Datei im Browser geladen wird (siehe Abbildung 2.3), werden die Lebensmittel tabellarisch angezeigt.

Dieser Crashkurs hat Ihnen einen ersten Vorgeschmack auf das Buch gegeben. In den folgenden Kapiteln lernen Sie Theorie und Praxis von XML näher kennen, und Sie werden erfahren, wie Sie ASP-Applikationen mit Hilfe von XML realisieren können.

Abbildung 2.3 Mit XSLT können XML-Daten unter anderem in HTML transformiert werden