



## WIE KANN ICH ACCESS XML-FÄHIGKEITEN UNABHÄNGIG VON DER VERSION BEBRINGEN?

Mit den verschiedenen Versionen von Access wurde die Unterstützung von XML immer mehr verbessert. Vollständig ist sie aber immer noch nicht. So kann man mit Access 2003 zwar XML-Dateien importieren und exportieren, aber selbst dabei bleiben die Funktionen ziemlich beschränkt. Z.B. kann man keine Attribute importieren.

Das Referenzieren eines externen XML-Parsers verbessert nicht nur die aktuelle XML-Fähigkeit in Microsoft Access, man kann damit auch noch jede Version von Microsoft Access auf das gleiche Niveau bringen. Das liegt daran, dass der Parser ein unabhängiges Tool ist.

### Parser einbinden

Um auf den Parser zu verweisen, rufen Sie einfach im VBA-Editor den Menüpunkt Extras – Verweise auf, um das Dialogfeld Verweise zu öffnen. Scrollen Sie nun durch die Liste, und suchen Sie nach Microsoft XML. Der Verweis wird eine Versionsnummer beinhalten – es funktioniert aber mit jeder Version. Wenn Sie sich dafür interessieren, wodurch die Versionen sich unterscheiden, sollten Sie mal einen Blick auf die Webseite von Microsoft werfen: <http://msdn.microsoft.com/xml/default.aspx>. Hier finden Sie auch die Möglichkeit, den neusten Parser herunterzuladen.

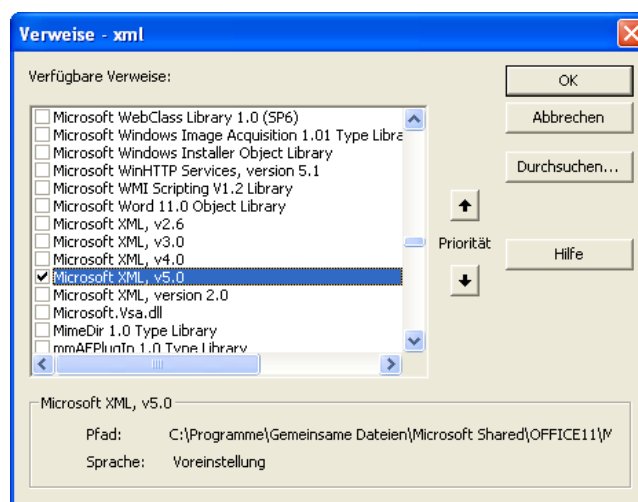


Abbildung 1 - Der Verweisdialog

Wenn der Verweis gesetzt wurde, können Sie mit **XML** auf vielerlei raffinierte Weise arbeiten. Unser Tipp kann Ihnen nicht beibringen, wie mit dem Parser umzugehen ist. Stattdessen werden Sie den Parser nutzen, um eine einfache **XML-Daten zu laden** und in eine **Access-Tabelle einzufügen**. Dabei werden Sie zwei Techniken kennen lernen:

- ➔ wie man die XML-Daten filtert
- ➔ und wie man Attribute laden kann

## Vorbereitung

Wir haben Ihnen eine **XML-Datei** mit **Kunden-Informationen** vorbereitet. Sie können die Datei hier: <http://kulpa-online.com/files-1040.html> downloaden.

Als 1. benötigen wir nun die Tabelle **tblKunden** für den Import, falls diese noch nicht vorhanden ist. Öffnen Sie ein neues Modul und binden Sie die **DAO-Referenz** über **Extras – Verweise** ein. Kopieren Sie nachstehenden Code in das Modul und führen Sie die Prozedur **CreateImportTable** im Direktfenster aus:

```
Public Function CreateImportTable() As Long

    Dim dbs As DAO.Database
    Dim sSql As String
    Dim sCheck As String
    Dim sTable As String

    On Error GoTo CreateImportTable_Err

    Set dbs = CurrentDb()
    ' Als 1. sollten wir überprüfen, ob die Tabelle
    ' existiert. Wenn ja, löschen wir diese.
    sTable = "tblKunden"
    ' Fehlerbehandlung ausschalten
    On Error Resume Next
    sCheck = dbs.TableDefs(sTable).Name
    ' Fehlerbehandlung wieder einschalten
    On Error GoTo CreateImportTable_Err
    ' Falls zuvor kein Fehler, dann existiert
    ' die Tabelle
    If Err.Number = 0 Then
        ' SQL-DDL-Löschabfrage basteln
        sSql = "DROP TABLE " & sTable
        ' Abfrage ausführen
        dbs.Execute sSql, dbFailOnError
    End If
    ' Nun bauen wir uns eine SQL-DDL-Abfrage für das Erstellen
    ' der Tabelle zusammen
    sSql = "CREATE TABLE " & sTable & " ( Kundennr CHAR(10) CONSTRAINT " & _
        "PK_tbl PRIMARY KEY, Firma CHAR(100), " & _
        "Kontaktperson CHAR(100), Strasse CHAR(100), PLZ " & _
        "CHAR(30), Ort CHAR(100), Telefon CHAR(50), Land " & _
        "CHAR(100))"
    ' Tabellenerstellungsabfrage ausführen
    dbs.Execute sSql, dbFailOnError
    ' Alles in Ordnung
    MsgBox "Die Tabelle: " & sTable & " wurde erfolgreich erstellt!", _
        vbInformation, "Hinweis"

CreateImportTable_Exit:
    On Error GoTo 0
    CreateImportTable = True
    Exit Function

CreateImportTable_Err:
    CreateImportTable = Err
```

```

MsgBox "Fehler " & Err.Number & ": " & _
      Err.Description, vbCritical, _
      "modXmlImport.CreateImportTable"
Resume CreateImportTable_Exit
End Function

```

## Der XML-Import – Ein Beispiel

Kopieren Sie jetzt nachstehenden Code in das zuvor erstellte Modul und führen Sie die Prozedur **ReadAndWriteXMLData** im Direktfenster aus.

**Hinweis:** Wir benötigen die Hilfsroutine **MakeQuotes**, damit die SQL-Anweisungen korrekt geschrieben werden (siehe auch: <http://kulpa-online.com/tipps-access-datenzugriff-4013.html>).

```

Public Function ReadAndWriteXMLData() As Long

    Dim dbs As DAO.Database

    ' Benötigte XML-Objekte
    Dim xml_obj As MSXML2.DOMDocument           ' Datei
    Dim xml_list As MSXML2.IXMLDOMNodeList     ' Liste
    Dim xml_nod As MSXML2.IXMLDOMNode         ' Element

    Dim sSql As String

    On Error GoTo ReadAndWriteXMLData_Err

    Set dbs = CurrentDb()

    Set xml_obj = New MSXML2.DOMDocument

    With xml_obj
        ' Asynchronausführung
        .async = False
        ' XML-Datei laden
        .Load CurrentProject.Path & "\kundenliste.xml"
        ' Kundenliste zuordnen
        Set xml_list = .selectNodes("Kundenliste/Land/Kunde")
        ' Schleife über alle Kunden in der Kundenliste
        For Each xml_nod In xml_list
            With xml_nod
                ' Anfügeabfrage zusammenbasteln
                sSql = "Insert Into tblKunden ( Kundennr, Firma, Kontaktperson, Strasse,
                PLZ, Ort, Telefon, Land ) "
                sSql = sSql & "Values ( "
                ' Auslesen der einzelnen Node-Elemente im Knoten Kunden
                sSql = sSql & MakeQuotes(.childNodes(0).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(1).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(2).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(3).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(4).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(5).Text) & ", "
                sSql = sSql & MakeQuotes(.childNodes(6).Text) & ", "
                ' Auslesen des übergeordneten Elementes Land
                sSql = sSql & MakeQuotes(.parentNode.Attributes(0).Text) & ")"
            End With
            ' Abfrage ausführen
            dbs.Execute sSql, dbFailOnError
        Next xml_nod
    End With

    ' Speicher freigeben
    If Not dbs Is Nothing Then dbs.Close: Set dbs = Nothing
    If Not xml_nod Is Nothing Then Set xml_nod = Nothing

```

```

If Not xml_list Is Nothing Then Set xml_list = Nothing
If Not xml_obj Is Nothing Then Set xml_obj = Nothing

' Alles in Ordnung
MsgBox "Der Import wurde erfolgreich durchgeführt!", _
    vbInformation, "Hinweis"

ReadAndWriteXMLData_Exit:
On Error GoTo 0
ReadAndWriteXMLData = True
Exit Function

ReadAndWriteXMLData_Err:
ReadAndWriteXMLData = Err
MsgBox "Fehler " & Err.Number & ": " & _
    Err.Description, vbCritical, _
    "modXmlImport.ReadAndWriteXMLData"
Resume ReadAndWriteXMLData_Exit

End Function

Function MakeQuotes(ByVal psValue As String) As String

'// -----
'// Methode   | Überprüft Text auf Anführungszeichen
'//           | und wandelt diesen in eine korrekten
'//           | Schreibweise um
'// -----
'// Parameter | psValue - Textinhalt
'// -----
'// Rückgabe  | SQL-String-Inhalt
'// -----
'// Erstellt  | Manuela Kulpa
'//           | EDV Innovation & Consulting - Dormagen
'// -----
'// Beispielaufruf:
'//   Inhalt des Datenbankfeldes Kunde = Hung "Owl" AllNight
'//   sSql = "... Where Firma = " & MakeQuotes(Kunde)
'// -----

Dim sOut As String
Dim sChar As String
Dim iFound As Integer

For iFound = 1 To Len(psValue)
    sChar = Mid$(psValue, iFound, 1)
    If Asc(sChar) = 34 Then
        sOut = sOut & Chr(34) & " & " & String$(4, Chr(34)) & " & " & Chr(34)
    Else
        sOut = sOut & sChar
    End If
Next

MakeQuotes = Chr(34) & sOut & Chr(34)

End Function

```

## Erklärung der Code-Zeilen:

---

Am Anfang **deklarieren** wir die **Objekte**, die über den Verweis auf den **XML-Parser** verfügbar gemacht wurden. Dann wird die XML-Datei **Kundenliste** in die Objektvariable **xml\_obj** geladen und anschließend erfolgt die **Bearbeitung** der einzelnen Knoten.

Normalerweise bestehen XML-Objekte aus **Knoten** und **Knotenlisten**. Eine Liste ist eine Sammlung von Knoten. Bei unserer Kunden-Liste handelt es sich bei den Knoten um die **Kunden-Elemente**, die den **Land-Knoten** (mit entsprechenden Attributen) untergeordnet sind.

Die Kunden-Knoten haben jeweils **sieben Elemente**: Kundennr, Firma, Kontaktperson, Straße, PLZ, Ort und Telefonnummer. Diese untergeordneten Elemente und das übergeordnete **Land-Element** dienen als Basis zum Erstellen unseres SQL-Insert-Befehls.

Nachdem die Routine **ReadAndWriteXMLData** durchgelaufen ist, wird die Tabelle **tblKunden** mit den XML-Daten gefüllt sein! Eigentlich ganz einfach :), oder?

## Schlussbemerkung

Mit dieser kurzen Routine haben Sie nun zwei Dinge erreicht, die normalerweise nicht realisierbar erscheinen. Zum einen kann nicht nur **Access 2003** gut mit **XML** umgehen, zum anderen lassen sich auch Attribute verarbeiten. Die Routine in diesem Tipp läuft mit **allen** Versionen von Access, die auf den Parser verweisen können und hat kein Problem damit, einen Wert eines Attributs in eine Access-Tabelle zu kopieren.