27.2 XmlWriter

Die Klasse XmlWriter ermöglicht es Ihnen, den Inhalt einer XML-Datei zu schreiben. Sobald Sie eine XML-Datei zum Schreiben geöffnet haben, müssen Sie jeden Knoten nacheinander schreiben – vom ersten Knoten bis zum Letzten in der notwendigen Tiefe. Die Klasse *XmlWriter* ist weniger flexibel als die Klasse XmlDocument nach dem 'Document Object Model' (DOM). Sie können die Klasse XmlWriter aber einsetzen, um schnell und wenig fehleranfällig den Inhalt einer (neuen) XML-Datei zu schreiben.

Die XML-Zeilen können sofort in eine Datei geschrieben oder temporär im Speicher gespeichert werden, so dass deren Inhalt wie eine Zeichenkette verwendet werden kann. Sie können ein Objekt der Klasse erzeugen:

```
DIM hXMLWriter AS XmlWriter
hXMLWriter = NEW XmlWriter
```

27.2.0 Eigenschaften

Die Klasse XmlWriter verfügt u.a. über diese Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
DTD	_XmlWriterDTD	Gibt den Inhalt eines existierenden (internen) Document-Type-Definition-Abschnitts (DTD) zurück.
Data	String	Gibt den Inhalt des XML-Streams als String zurück.

Tabelle 27.2.0.1 : Eigenschaften der Klasse XmlWriter

27.2.1 Methoden

Die Klasse *XmlWriter* verfügt nur über diese Methoden:

Methode	Rückgabetyp	Beschreibung
Open ([fileName As String, Indent As Boolean, Encoding As String])	-	Öffnet einen XML-Stream für das Schreiben. Sie müssen die Open()-Methode vor allen anderen XmlWriter-Methoden aufrufen. - fileName: Wenn Sie direkt in eine XML-Datei schreiben wollen, so muss fileName ein gültiger Datei-Pfad sein. Setzen Sie fileName auf Null, um nach dem Aufruf der EndDocument-Methode (wenn das gesamte Dokument geschrieben wurde) den Inhalt als String zu erhalten. - Indent: Wenn der Parameter auf True gesetzt ist, dann wird die Datei mit Einrückungen formatiert. - Encoding: Standardmäßig verwendet XmlWriter UTF-8-Codierung um die Datei zu schreiben. Sie können jede andere Kodierung angeben, sofern diese von der Bibliothek libxml2 unterstützt wird.
Close()	String	Schließt den XML-Stream.
Comment (sComment As String)	-	Fügt einen Kommentar in den XML-Stream ein. Die Syntax entspricht der von HTML-Kommentaren.
PI (Target As String, Content As String)	-	Fügt Prozess-Informationen in den XML-Stream ein. Das sollten Sie nur dann tun, wenn Sie sicher sind, dass der XML-Parser diese umsetzen kann.
StartElement (TagName As String [, Attributes As String[], Prefix As String, URI As String])	-	Startet einen XML-Knoten mit dem Namen 'TagName'. Wenn dieser Knoten Attribute enthält, können Sie diese Attribute in ein String-Array einfügen. Jedes Attribut muss ein Name-Wert-Paar sein, wobei der erste String der Attributname und der zweite der Attributwert ist. Prefix ist das Namespace-Präfix und URI der Namespace-URI dieses Knotens.
Element (TagName As String [, Value As String, Prefix As String, URI As	-	Fügt ein XML-Element in den XML-Stream ein. Die drei Parameter Value, Prefix und URI vom Typ String

Methode	Rückgabetyp	Beschreibung
String])		sind optional. - TagName: Name des Elements - Value: Wert des Elements - Prefix: Präfix für den Name-Space - URI: Angabe für einen Link.
EndElement()	-	Schließt das aktuelle Element mit </element_name> ab.
Text (sText As String)	-	Fügt sText in einem Element ein.
CDATA (Data As String)	-	Fügt eine CDATA-Sektion mit dem Inhalt 'Data' ein.
Attribute (Name As String, Value As String [, Prefix As String, URI As String])	-	Fügt Attribute ein. Die Parameter Prefix und URI sind optional.
Flush()	-	Löscht alle Elemente.
EndDocument()	String	Rendert das XML-Dokument und gibt das komplette, *wohlgeformte* XML-Dokument als String zurück.

Tabelle 27.2.1.1 : Methoden der Klasse XmlWriter

Attribute können Sie so festlegen:

```
hXMLElement.SetAttribute("src", Application.Path &/ "images/augentrost.jpg\" width=\"255\"
height=\"148\" alt=\"Augentrost")
```

Die folgende mehrzeilige Variante ist m.E. besser lesbar, weil man bestimmte Zeichen nicht maskieren muss:

```
hXMLElement.SetAttribute("src", Application.Path &/ "images/augentrost.jpg")
hXMLElement.SetAttribute("width", "255")
hXMLElement.SetAttribute("height", "148")
hXMLElement.SetAttribute("alt", "Augentrost")
```

CDATA-Bereiche sind geschützte Datenbereiche, in denen Beschränkungen für die enthaltenen Zeichen nicht gelten. Die CDATA-Bereiche beginnen immer mit der Zeichenfolge <![CDATA[und enden mit]]>. Dazwischen ist jedes Zeichen gültig.

Beispiel für einen validen CDATA-Bereich:

```
<![CDATA[hXMLElement.SetAttribute("height", "148")]]>
```

27.2.2 Exkurs: Bash-Skript

Eine flotte Möglichkeit, den Inhalt einer CSV-Datei in eine XML-Datei zu schreiben, bietet ein vom Autor entwickeltes Bash-Skript, dem Sie den Pfad zur CSV-Datei, das Separator-Zeichen (Feld-Trennzeichen) und das Escape-Zeichen als Parameter übergeben. Das ist der Quelltext von csv2xml.sh:

```
#!/bin/bash

  if [ $# -ne 3 ] # Wenn die Anzahl der Parameter (Wert in $#) ungleich 3 ist ...
  then
    # FARB-WERTE:
    RO="\033[31m"     # rot
    NO="\033[0m"      # normal
    # echo -e "${RO}Syntax: $0 CSV-Datei-Pfad Feld-Trennzeichen Escape-Zeichen"
    echo -e "${RO}Syntax: $0 'CSV-File-Path' 'Field separator' 'Escape character'"
    exit 1
  fi

  # Sicherung der Variablen IFS in der Variablen backIFS (IFS = Internal Field Separator)
  backIFS=$IFS

  file_in=$1 # Parameter 1
  echo $file_in
  # Dateiname für file_out (XML-Datei) erzeugen
  filebasename=${file_in%.*}
  fileextension=".xml"
  file_out=$filebasename$fileextension
  echo $file_out
```

```
separator=$2 # Parameter 2
escape=$3    # Parameter 3

# 1. Zeile (= Liste der Feld-Namen) aus CSV-Datei lesen
read header_line < "$file_in"
IFS=$separator
# echo "IFS = $IFS"
# Liste der Feld-Namen im Array 'header' speichern -> Operator =()
header=($header_line)
# echo "Anzahl der Elemente im Array 'header' : "${#header[@]}
# Alle Elemente im Arrray 'header' ausgeben
# echo "Elemente im Array 'header': "${header[*]}
IFS='*'
# Inhalt der CSV-Datei zeilenweise einlesen und im Array 'content' speichern
i=0
while read content[$i]
do
  # echo "Zeile $i: ${content[i]}"
  ((i++))
done < $1
# 1. Zeile im Array 'content' löschen
unset content[0]
# Array kopieren - aber ohne das jetzt leere erste Element
content=(${content[*]})
# Alle Elemente im Arrray 'content' ausgeben
# for record in ${content[@]}
# do
#   echo "ZEILE : "$record
# done
# Inhalt der XML-Datei schreiben
#-------------------------------------------------------------------------------
# XML-Prolog
echo '<?xml version="1.0" encoding="UTF-8"?>' > $file_out
# XML-Root-Element
echo '<Data>' >> $file_out
# XML-Elemente
#.............................................................
for record in "${content[@]}"
do
echo '  <Record>' >> $file_out
  index=0
  #.............................................................
  IFS=$separator list=($record)
  # echo "Anzahl aller Elemente im Array 'list' = "${#list[@]}
  # Alle Elemente im Arrray 'list' ausgeben
  # echo ${list[*]}
  for (( c=0; c<=${#header[@]}-1; c++ ))
  do
    tag=${header[$c]#$escape}
    tag=${tag%$escape}
    value=${list[$c]#$escape}
    value=${value%$escape}
    echo '    <'${tag}'>'${value}'</'${tag}'>' >> $file_out
  done
  #.............................................................
  echo '  </Record>' >> $file_out
  ((index++))
done
#-------------------------------------------------------------------------------
 echo '</Data>' >> $file_out
IFS=$backIFS
# sleep 2s
# cat $file_out
exit 0
```

Der Aufruf des *ausführbaren* Bash-Skripts ist simpel und liefert *db.xml* als XML-Datei:

```
$ ./csv2xml.sh "db.csv" "," "\""
db.csv
db.xml
```

27.2.3 Projekte

Alle vorgestellten Projekte demonstrieren jeweils unterschiedliche Ansätze und Daten-Quellen, um Daten in eine XML-Datei zu exportieren.

27.2.3.1 Projekt 1

Im Projekt 1 (data2xml_writer) wird Ihnen gezeigt, wie ein XML-Dokument unter Verwendung der Klasse XmlWriter neu geschrieben wird. Die zu exportierenden Daten werden in einer Struktur (Datentyp

Struct) *programmintern* zur Verfügung gestellt. Dieser Datentyp sichert u.a. den Vorteil, die einzelnen Daten über ihren Namen anzusprechen statt über einen anonymen Index:

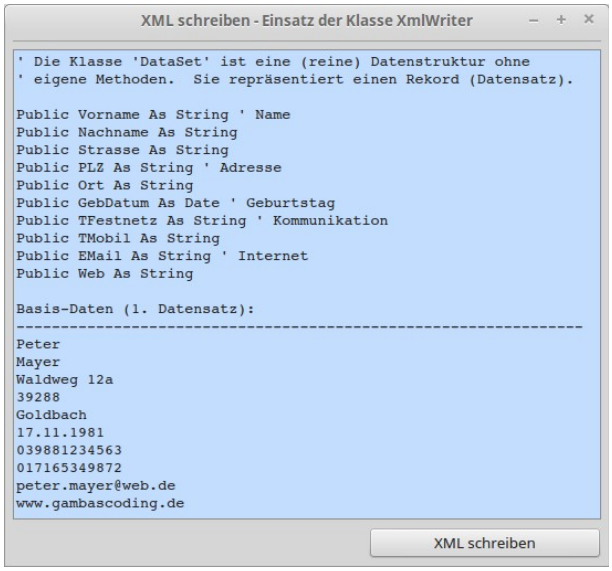

Abbildung 27.2.3.1.1: Datentyp Struct (1. Datensatz)

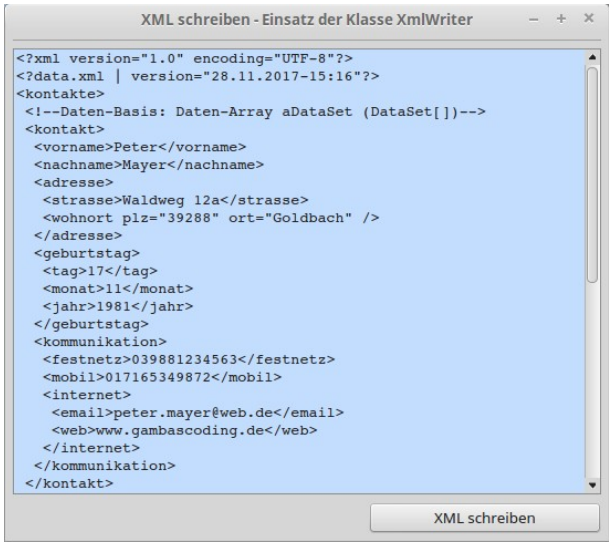

Abbildung 27.2.3.1.2: Ausschnitt XML

Es wird Ihnen nur die wichtigste Prozedur *WriteXML()* vorgestellt und kommentiert:

```
[1]  Private Sub WriteXML()
[2]
[3]    Dim i As Integer
[4]
[5]    hXmlWriter = New XmlWriter
[6]    hXmlWriter.Open(Null, True, "UTF-8")
[7]  ' hXmlWriter.Open(xmlFilePath, True, "UTF-8")
[8]
[9]    hXmlWriter.PI(File.Name(xmlFilePath), "| version=\"" & Format(Now, "dd.mm.yyyy-hh:nn") & "\"")
[10]
[11]   hXmlWriter.StartElement("kontakte")
[12]   hXmlWriter.Comment("Daten-Basis: Daten-Array aDataSet (DataSet[])")
[13]
[14]   For i = 0 To aDataSet.Max
[15]     hXmlWriter.StartElement("kontakt")
[16]       hXmlWriter.Element("vorname", aDataSet[i].Vorname)
[17]       hXmlWriter.Element("nachname", aDataSet[i].Nachname)
[18]       hXmlWriter.StartElement("adresse")
[19]       hXmlWriter.Element("strasse", aDataSet[i].Strasse)
[20]       hXmlWriter.StartElement("wohnort", ["plz", aDataSet[i].PLZ, "ort", aDataSet[i].Ort])
```

```
[21]      ' hXmlWriter.StartElement("wohnort")
[22]      ' hXmlWriter.Attribute("plz", aDataSet[i].PLZ)
[23]      ' hXmlWriter.Attribute("ort", aDataSet[i].Ort) ' 3-line alternative
[24]        hXmlWriter.EndElement ' wohnort
[25]        hXmlWriter.EndElement ' adresse
[26]        hXmlWriter.StartElement("geburtstag")
[27]          hXmlWriter.Element("tag", Day(aDataSet[i].GebDatum))
[28]          hXmlWriter.Element("monat", Month(aDataSet[i].GebDatum))
[29]          hXmlWriter.Element("jahr", Year(aDataSet[i].GebDatum))
[30]        hXmlWriter.EndElement ' geburtstag
[31]        hXmlWriter.StartElement("kommunikation")
[32]          hXmlWriter.Element("festnetz", aDataSet[i].TFestnetz)
[33]          hXmlWriter.Element("mobil", aDataSet[i].TMobil)
[34]          hXmlWriter.StartElement("internet")
[35]            hXmlWriter.Element("email", aDataSet[i].EMail)
[36]            hXmlWriter.Element("web", aDataSet[i].Web)
[37]          hXmlWriter.EndElement ' internet
[38]        hXmlWriter.EndElement ' kommunikation
[39]      hXmlWriter.EndElement ' kontakt
[40]    Next
[41]
[42]    hXmlWriter.EndElement ' kontakte
[43]    hXmlWriter.EndDocument
[44]
[45]    txaXML.Text = hXmlWriter.Data
[46]    File.Save(xmlFilePath, hXmlWriter.Data)
[47]
[48] End
```

Kommentar:

- In der Zeile 5 wird ein neues XmlWriter-Objekt erzeugt und in der Zeile 6 im Speicher geöffnet. Alternativ könnten Sie auch alle zu exportierenden Daten sofort in eine XML-Datei schreiben, wie es in der Zeile 7 angegeben wird.
- Die Zeile 9 schreibt Prozess-Informationen in das XmlWriter-Objekt. Es sind nur Pseudo-Daten, die das Schreiben von Prozess-Informationen demonstrieren. Echte Prozess-Informationen könnten zum Beispiel Angaben über das Ausgabe-Format für einen XML-Parser enthalten und damit den Lese-Prozess steuern.
- Zwischen den Zeilen 11 bis 41 werden alle Daten – auch in einer For-To-Next-Kontrollstruktur – in das XmlWriter-Objekt geschrieben.
- Die Anzeige des XML-Inhalts in einer TextArea erfolgt in der Zeile 45.
- Erst in der Zeile 46 wird der XML-Inhalt in einer XML-Datei gespeichert. Ein Datei-Speichern-Dialog wird nicht angeboten.
- Das Projekt kann leicht an Daten aus einer anderen Daten-Quelle angepasst werden.

27.2.3.2 Projekt 2

Im Projekt 2 (dbdata2xml_writer) werden Daten aus einer (SQLite3-)Datenbank-Tabelle über ein (internes) Filter ausgelesen und dann angezeigt. Die selektierten Datensätze werden unter Verwendung der Klasse *XmlWriter* in eine XML-Datei exportiert.

Abbildung 27.2.3.2.1: Anzeige der selektierten Datensätze

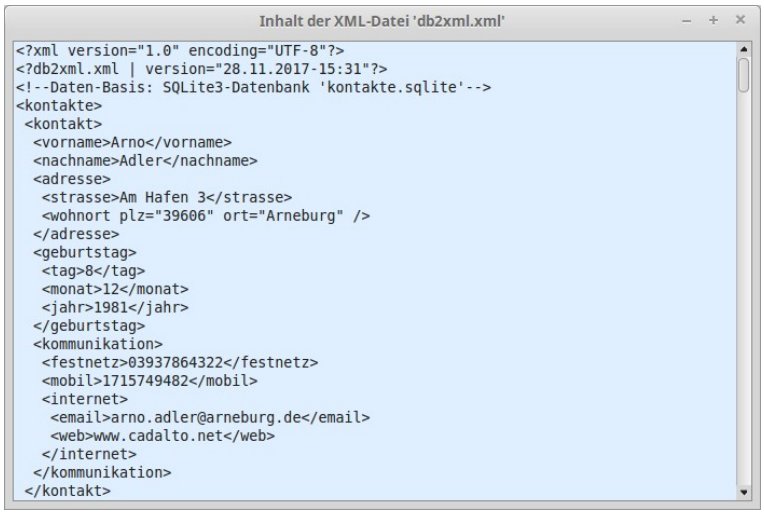

Abbildung 27.2.3.2.2: Ausschnitt XML

Es wird Ihnen der komplette Quelltext vorgestellt und kurz kommentiert:

```
[1]  ' Gambas class file
[2]
[3]  Private rDBResult As Result
[4]  Private xmlDoc As XmlWriter
[5]
[6]  Public Sub Form_Open()
[7]    FMain.Center
[8]    FMain.Resizable = False
[9]    FMain.Caption = ("Export: DB.Data > XML-File")
[10]
[11]   DataSource1.Connection = New Connection
[12]   DataSource1.Connection.Type = "sqlite3"
[13]   DataSource1.Connection.Host = Application.Path &/ "DataBase"
[14]   DataSource1.Connection.Name = "kontakte.sqlite" ' Name of the SQLite3 database
[15]
[16]   If Not Exist(DataSource1.Connection.Host &/ DataSource1.Connection.Name) Then
[17]      Message(("<font color='red'>(No SQLite DB available.)</font><hr>The program is terminated!"))
[18]      Quit
[19]   Endif
[20]
[21]   DataSource1.Table = "kontakte" ' Name of the DB
[22] ' DataSource1.Filter = "wohnort = \"Leipzig\"  Or wohnort = \"Hamburg\"" ' Filter-Example
[23]
[24]   DataBrowser1.Orientation = Align.Bottom
[25]   DataBrowser1.Grid = True
[26]   DataBrowser1.Columns = ["vorname", "nachname", "wohnort", "plz", "strasse", "gebdatum", "tfestnetz",
       "tmobil", "email", "web"]
[27]   DataBrowser1.Labels = ["Vorname", "Nachname", "Wohnort", "PLZ", "Straße", "GebDatum", "Festnetz-Num-
       mer", "Mobilfunk-Nummer", "EMail-Adresse", "URL"]
[28]
[29]   DataBrowser1.View.Columns[0].Width = 95     ' Vorname
[30]   DataBrowser1.View.Columns[1].Width = 110    ' Nachname
[31]   DataBrowser1.View.Columns[2].Width = 130    ' Wohnort
[32]   DataBrowser1.View.Columns[3].Width = 80     ' PLZ (Postleitzahl)
[33]   DataBrowser1.View.Columns[4].Width = 150    ' Strasse
[34]   DataBrowser1.View.Columns[5].Width = 140    ' GebDatum
[35]   DataBrowser1.View.Columns[6].Width = 140    ' Telefon-Festnetz
[36]   DataBrowser1.View.Columns[7].Width = 140    ' Telefon-Mobilfunk
[37]   DataBrowser1.View.Columns[8].Width = 160    ' EMail-Adresse
[38]   DataBrowser1.View.Columns[9].Width = 160    ' Web-Adresse
[39]
[40]   DataSource1.MoveFirst ' Internal DB pointer to the first record
[41]   DataBrowser1.View.MoveTo(0, 0) ' Select the 1st line in the DataBrowser
[42]
[43]   dcVorname.Field = "vorname" ' DB-Field (gb.db.form)
[44]   dcNachname.Field = "nachname"
[45]   dcPLZ.Field = "plz"
[46]   dcWohnort.Field = "wohnort"
[47]   dcStrasse.Field = "strasse"
[48]   dcGebDatum.Field = "gebdatum"
[49]   dcTelefonFestnetz.Field = "tfestnetz"
[50]   dcTelefonHandy.Field = "tmobil"
[51]   dcEMail.Field = "email"
[52]   dcWeb.Field = "web"
```

```
[53] End
[54]
[55] Private Sub Export2XML()
[56]
[57]   Dim sSQL As String
[58]   Dim xmlFilePath As String = Application.Path &/ "db2xml.xml"
[59]
[60]   sSQL = "SELECT * FROM " & DataSource1.Table
[61] ' sSQL = "SELECT * FROM " & DataSource1.Table & " WHERE " & DataSource1.Filter
[62]
[63]   Try rDBResult = DataSource1.Connection.Exec(sSQL)
[64]
[65]   If Error Then
[66]      Message("Error!<br />" & Error.Text & "<br />" & Error.Where)
[67]      Return
[68]   Endif
[69]
[70]   If rDBResult.Count = 0 Then
[71]      Message(("The number of selected records is zero!"))
[72]      Return
[73]   Endif
[74]
[75]   xmlDoc = New XmlWriter
[76]   xmlDoc.Open(xmlFilePath, True, "UTF-8")
[77]
[78]   xmlDoc.PI(File.Name(xmlFilePath), "| version=\"" & Format(Now, "dd.mm.yyyy-hh:nn") & "\"")
[79]   xmlDoc.Comment(("Data base: SQLite3 database 'kontakte.sqlite'"))
[80]   xmlDoc.StartElement("kontakte")
[81]   For Each rDBResult
[82]       xmlDoc.StartElement("kontakt")
[83]       xmlDoc.Element("vorname", rDBResult["vorname"])
[84]       xmlDoc.Element("nachname", rDBResult["nachname"])
[85]       xmlDoc.StartElement("adresse")
[86]         xmlDoc.Element("strasse", rDBResult["strasse"])
[87]         xmlDoc.StartElement("wohnort", ["plz", rDBResult["plz"], "ort", rDBResult["wohnort"]])
[88]       ' xmlDoc.StartElement("wohnort")
[89]       ' xmlDoc.Attribute("plz", rDBResult["plz"])
[90]       ' xmlDoc.Attribute("ort", rDBResult["wohnort"]) ' 3-line alternative
[91]         xmlDoc.EndElement ' wohnort
[92]       xmlDoc.EndElement ' adresse
[93]       xmlDoc.StartElement("geburtstag")
[94]         xmlDoc.Element("tag", Day(rDBResult!gebdatum))
[95]         xmlDoc.Element("monat", Month(rDBResult!gebdatum))
[96]         xmlDoc.Element("jahr", Year(rDBResult!gebdatum))
[97]       xmlDoc.EndElement ' geburtstag
[98]       xmlDoc.StartElement("kommunikation")
[99]         xmlDoc.Element("festnetz", rDBResult["tfestnetz"])
[100]          xmlDoc.Element("mobil", rDBResult["tmobil"])
[101]          xmlDoc.StartElement("internet")
[102]            xmlDoc.Element("email", rDBResult["email"])
[103]            xmlDoc.Element("web", rDBResult["web"])
[104]          xmlDoc.EndElement ' internet
[105]        xmlDoc.EndElement ' kommunikation
[106]      xmlDoc.EndElement ' kontakt
[107]    Next
[108]
[109]    xmlDoc.EndElement ' kontakte
[110]    xmlDoc.EndDocument()
[111]
[112]  End
[113]
[114]  Public Sub btnDBData2XML_Click()
[115]    Export2XML()
[116]    Wait
[117]    FShowData.ShowModal()
[118]  End
[119]
[120]  Public Sub btnEnde_Click()
[121]    FMain.Close()
[122]  End
```

Kommentar:

- In den Zeilen 11 bis 52 werden alle Daten aus der Datenbank-Tabelle ausgelesen und angezeigt. Es wäre auch möglich, nur bestimmte Daten auszufiltern. Das erfolgt zum Beispiel in der aktuell auskommentierten Zeile 22, die mit der ebenfalls auskommentierten Zeile 61 korrespondiert.
- In der Zeile 110 wird der vollständige XML-Inhalt in einer XML-Datei gespeichert, die in der Zeile 76 angelegt wurde. Ein Datei-Speichern-Dialog wird nicht angeboten.
- Die Hauptlast im zweiten Projekt trägt die Prozedur Export2XML.

Zur Anzeige des Inhalts der XML-Datei dient eine TextArea auf dem Formular FShowData, das modal geöffnet wird:

```
[1]  ' Gambas class file
[2]
[3]  Public Sub Form_Open()
[4]    FShowData.Resizable = True
[5]    FShowData.Caption = ("XMLData")
[6]    If Exist("db2xml.xml") Then txaXMLData.Text = File.Load("db2xml.xml")
[7]    txaXMLData.Pos = 0
[8]  End
```

Bevor Sie das Projekt 2 umsetzen sollten Sie prüfen, ob das eingesetzte Datenbank-Management einen Daten-Export in das XML-Format zur Verfügung stellt.

27.2.3.3 Projekt 3

Im dritten Projekt werden Daten aus einer CSV-Datei unter Verwendung der Klassen XmlWriter und CsvFile (gb.utils) in eine XML-Datei exportiert. Die Klasse CsvFile (gb.util) kann eingesetzt werden, um eine CSV-Datei zu lesen und den Inhalt zu dekodieren. Dabei ist zu beachten, das die Klasse CsvFile nicht mit allen CSV-Formaten umgehen kann. Daher wird die verwendete CSV-Datei erst so präpariert, dass sie den Regeln für das Format von CSV-Dateien von Jon Avrach – gefunden auf auf der Website *https://www.thoughtspot.com/blog/6-rules-creating-valid-csv-files* → Kapitel 6 entspricht.

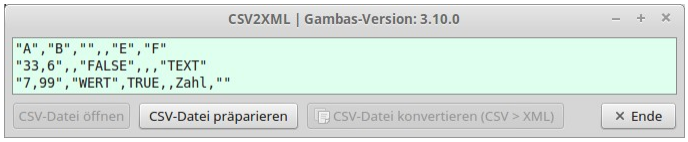

Abbildung 27.2.3.3.1: Originale CSV-Datei

Kommentar:

- Beachten Sie: CSV-Dateien ohne Feldliste können von der Klasse CsvFile (gb.utils) *nicht* verarbeitet werden.
- In der Liste der Feldnamen fehlen zwei – erkennbar an "" und ,, .
- Im ersten Datensatz fehlen drei Werte; im zweiten zwei.

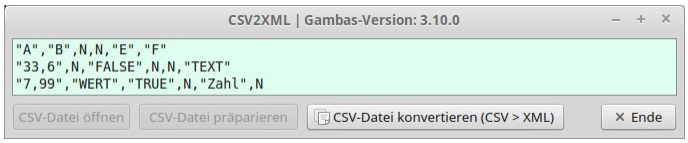

Abbildung 27.2.3.3.2: Inhalt einer präparierten CSV-Datei

Erst danach werden die Daten aus der präparierten CSV-Datei – die temporär in einer CSV-Datei gespeichert wird – in eine XML-Datei exportiert:

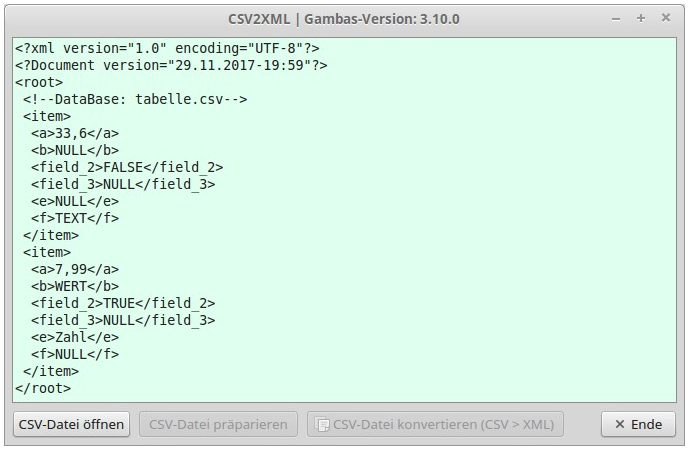

Abbildung 27.2.3.3.3: Inhalt der XML-Datei

Es wird der komplette Quelltext angezeigt. Interessant sind u.a. die Funktion *Prepare(fsFilePath As String) As String* und die Prozedur *WriteXML()*:

```
[1]  ' Gambas class file
[2]
[3]  Private $sSeparator As String
[4]  Private $sEscape As String
[5]
[6]  Public hXmlWriter As XmlWriter
[7]  Public sFilePath As String
[8]  Public hCSVFile As CsvFile
[9]
[10] Public Sub Form_Open()
[11]   FMain.Center
[12]   FMain.Resizable = True
[13]   FMain.Caption = "CSV2XML | Gambas-Version: " & System.FullVersion
[14]
[15]   $sSeparator = ","
[16]   $sEscape = "\"" ' Alternative: Chr(34) ~> "
[17]
[18]   btnConvertAndSave.Enabled = False
[19]   btnCSVPrepare.Enabled = False
[20]
[21] End
[22]
[23] Public Sub btnCSVOpen_Click()
[24]   sFilePath = FileOpen()
[25]   If sFilePath Then
[26]      txaData.Text = File.Load(sFilePath)
[27]      btnCSVPrepare.Enabled = True
[28]      btnCSVOpen.Enabled = False
[29]   Else
[30]      Return
[31]   Endif
[32] End
[33]
[34] Public Sub btnCSVPrepare_Click()
[35]   If sFilePath Then
[36]      txaData.Text = Prepare(sFilePath)
[37]      txaData.Pos = 0
[38]      File.Save(File.Dir(sFilePath) &/ File.BaseName(sFilePath) & "_pre" & ".csv", Prepare(sFilePath))
[39]      Wait
[40]      btnCSVOpen.Enabled = False
[41]      btnConvertAndSave.Enabled = True
[42]      btnCSVPrepare.Enabled = False
[43]   Else
[44]     Return
[45]   Endif
[46] End
[47]
[48] Public Sub btnConvertAndSave_Click()
[49]   WriteXML()
[50]   RemovePrepareFile()
[51]   btnCSVOpen.Enabled = True
[52]   btnConvertAndSave.Enabled = False
[53] End
[54]
[55] Public Sub Form_Close()
[56]   RemovePrepareFile()
[57] End
[58]
[59] Public Sub btnClose_Click()
[60]   FMain.Close()
[61] End
[62]
[63] Private Function Prepare(fsFilePath As String) As String
[64]
[65]   Dim sCSVContent, sCSVChanged, sLine, sField, sNewLine As String
[66]
[67]   sCSVContent = File.Load(fsFilePath)
[68] ' Parser
[69]   For Each sLine In Split(sCSVContent, gb.NewLine, $sEscape, False, True)
[70]     sLine = Trim(sLine)
[71]     If Not sLine Then Continue ' Leerzeilen werden ignoriert
[72]     sNewLine = ""
[73]     For Each sField In Split(sLine, $sSeparator, $sEscape, False, True)
[74]       If sField Not Begins $sEscape And If sField Not Ends $sEscape Then
[75]          sField = $sEscape & sField & $sEscape
[76]       Endif
[77]       If sField = $sEscape & $sEscape Or If sField = $sEscape & " " & $sEscape Then
[78]          sField = "N"
[79]       Endif
[80]       sNewLine &= sField & $sSeparator
```

```
[81]      Next
[82]       If sNewLine Ends $sSeparator Then sNewLine = Left(sNewLine, -1)
[83]       sCSVChanged &= sNewLine & gb.NewLine
[84]    Next
[85]    sCSVChanged = Left(sCSVChanged, -1) ' Das Zeilenende-Zeichen der *letzten* Zeile wird entfernt!
[86]
[87]    Return sCSVChanged
[88]
[89] End
[90]
[91] Private Sub WriteXML()
[92]
[93]    Dim k As Integer
[94]    Dim cLine As Collection
[95]    Dim sFieldName, sElementValue As String
[96]
[97]    hCSVFile = New CsvFile(sFilePath, $sSeparator, $sEscape)
[98]
[99]    hXmlWriter = New XmlWriter
[100]     hXmlWriter.Open(Null, True, "UTF-8")
[101]
[102]     hXmlWriter.PI("Document", "version=\"" & Format(Now, "dd.mm.yyyy-hh:nn") & "\"")
[103]
[104]     hXmlWriter.StartElement("root")
[105]       hXmlWriter.Comment("DataBase: " & File.Name(sFilePath))
[106]       While Not hCSVFile.Eof
[107]         hXmlWriter.StartElement("item")
[108]         cLine = hCSVFile.Read()
[109]         k = 0
[110]         For Each sFieldName In hCSVFile.Fields
[111]           If sFieldName Begins "#" Then
[112]              If cLine[sFieldName] = Null Then
[113]                 sElementValue = "NULL"
[114]                 hXmlWriter.Element("field_" & CStr(k), sElementValue)
[115]              Else
[116]                 hXmlWriter.Element("field_" & CStr(k), cLine[sFieldName])
[117]              Endif
[118]           Else
[119]              If cLine[sFieldName] Then
[120]                 hXmlWriter.Element(sFieldName, cLine[sFieldName])
[121]              Else
[122]                 hXmlWriter.Element(sFieldName, "NULL")
[123]              Endif
[124]           Endif
[125]           Inc k
[126]         Next
[127]         hXmlWriter.EndElement
[128]       Wend
[129]       hXmlWriter.EndElement ' root
[130]     hXmlWriter.EndDocument
[131]
[132]     txaData.Text = hXmlWriter.Data
[133]     txaData.Pos = 0
[134]
[135]     File.Save(File.Dir(sFilePath) &/ File.BaseName(sFilePath) & ".xml", hXmlWriter.Data)
[136]
[137]  End
[138]
[139]  Private Function FileOpen() As String
[140]    Dialog.Title = "Importieren Sie eine csv-Datei!"
[141]    Dialog.Filter = ["*.csv", "csv-Dateien"]
[142]    Dialog.Path = Application.Path &/ "CSV"
[143]    If Dialog.OpenFile(False) = True Then ' Multiselect=False (Standard)
[144]       Message.Info("Das Öffnen der csv-Datei wurde abgebrochen!")
[145]       Return
[146]    Else
[147]       Return Dialog.Path
[148]    Endif
[149]  End
[150]
[151]  Private Sub RemovePrepareFile()
[152]    If Exist(File.Dir(sFilePath) &/ File.BaseName(sFilePath) & "_pre.csv") Then
[153]       Try Kill File.Dir(sFilePath) &/ File.BaseName(sFilePath) & "_pre.csv"
[154]    Endif
[155]    Wait
[156]  End
```