6.13 Klasse CsvFile

Die Klasse CsvFile (gb.util) kann eingesetzt werden, um eine CSV-Datei zu lesen und den Inhalt zu dekodieren.
Achtung: Es wird davon ausgegangen, dass in der ersten Zeile der CSV-Datei notwendigerweise die Feldnamen stehen. Sind nicht alle Feld-Namen angegeben, dann wird der fehlende Feldname durch ein #-Zeichen ersetzt, dem der Index des Feldes in der imaginären Feldliste folgt.
Generell gilt: Die Klasse CsvFile kann nicht mit CSV-Dateien umgehen, die keine (erste) Zeile mit den Feld-Namen (Header) besitzen, obgleich der Header mit den Feldnamen nach RFC 4180 optional ist!

Sie können ein Objekt der Klasse erzeugen. Es wird eine CSV-Datei zum Lesen geöffnet:

```
Dim hCsvFile As CsvFile
hCsvFile = New CsvFile ( Path As String [ , Separator As String, Escape As String ] )
```

- Path – Pfad der CSV-Datei.
- Separator – Das Feld-Trennzeichen ist optional. Standardmäßig ist es ein Komma.
- Escape – Das Zeichen, das einen Feld-Wert umschließt, ist optional. Standardmäßig sind es doppelte Anführungszeichen wie zum Beispiel bei "FieldValue".

6.13.1 Eigenschaften

Die Klasse *CsvFile* verfügt über drei Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
Eof	Boolean	Es wird True zurückgegeben, wenn das Ende der CSV-Datei beim Lesen erreicht wurde.
Fields	String[ ]	Zurückgeben wird eine Liste der Feld-Namen als String-Array.
Line	Integer	Gibt die Nummer der zuletzt gelesenen Zeile zurück.

Tabelle 6.13.1.1 : Eigenschaften der Klasse CsvFile

Alle Feldnamen in der Liste der Feld-Namen werden nach dem folgenden Algorithmus normiert:

- Der Feld-Name wird in Kleinbuchstaben umgewandelt.
- Newlines und Tabs werden durch Leerzeichen ersetzt.
- Leerzeichen mit Unicode 160 werden durch normale Leerzeichen ersetzt.
- Wenn ein Leerzeichen einem anderen Leerzeichen folgt, wird es entfernt.
- Wenn ein (Feld-)Name leer ist, so wird er durch ein #-Zeichen ersetzt – gefolgt von der Feldindex-Nummer.

6.13.2 Methoden

Die Klasse *CsvFile* verfügt nur über diese eine Methode:

```
Function Read() As Collection
```

Die Funktion liest eine CSV-Datei und gibt sie als Collection von Feld-Werten zurück, die durch ihre Feld-Namen indiziert sind: Eigenschaft CsvFile.Fields. Die Feld-Werte werden nach dem folgenden Algorithmus normiert:

- Leerzeichen mit Unicode 160 werden durch normale Leerzeichen ersetzt.
- Zusätzliche Leerzeichen oder Steuerzeichen (ASCII-Code kleiner als 32) am Anfang oder am Ende eines Feld-Wertes werden entfernt.

6.13.3 Hinweise

Die folgenden Hinweise sollten Sie kennen und beachten:

- In Bezug auf die RFC 4180 stand in einem Kommentar: "In einigen CSV-Implementierungen werden führende und nachgestellte Leerzeichen und Tabs getrimmt (ignoriert). Ein solches

Trimmen ist nach RFC 4180 verboten, was bedeutet: "Leerzeichen werden als Teil eines Feldes betrachtet und sollten nicht ignoriert werden.""

- Diese klare Regel wird aber in der vorliegenden Klasse CsvFile sowohl bei der Extraktion der Feld-Namen – wenn es welche in der 1. Zeile gibt – als auch beim Auslesen der Feld-Werte übergangen.
- In einigen CSV-Dateien werden fehlende oder unbekannte Feld-Werte durch "" oder durch NULL gekennzeichnet. Liest man dann aber mit der Read()-Methode der Klasse CsvFile eine Zeile aus, dann fehlt (mindestens) ein Key-Value-Paar in dem als Collection zurückgegebenen Funktionswert der Read-Methode. Die Erklärung für dieses Verhalten ist einfach, denn der leere String ist in Gambas das Gleiche wie Null und Null in eine Collection zu packen ist das Gleiche wie den Schlüssel aus der Collection zu löschen!
- Eine Collection ist eine Hashtabelle. Das bedeutet u.a. dass Sie keinenfalls eine bestimmte Reihenfolge der Schlüssel bei der Iteration mit einer FOR-EACH-Kontrollstruktur durch eine Collection annehmen dürfen. Im Klartext: Mit For-Each bekommen Sie die Werte in der Collection a priori in beliebiger, nicht-deterministischer Reihenfolge. Sie dürfen also nicht annehmen, dass Sie sie in genau der gleichen Reihenfolge bekommen, wie sie in einer Zeile einer CSV-Datei stehen!

Die korrekte Konvertierung der Zeilen in einer CSV-Datei zum Beispiel nach Variant[] statt nach Collection würde so aussehen, wobei Sie alle Feld-Namen über die Eigenschaft *CsvFile.Fields* erhalten:

```
Private Function GetData() As Variant[]
  Dim aFieldList As Variant[]
  Dim cLine As Collection
  Dim sField As String, aFields As String[]

  aFieldList = New Variant[]

  While Not hCSVFile.Eof
    cLine = hCSVFile.Read()
    aFields = New String[]
    For Each sField In hCSVFile.Fields
      aFields.Add(cLine[sField])
    Next
    aFieldList.Add(aFields)
  Wend

  Return aFieldList

End
```

Wenn Sie zum Beispiel wissen wollen, ob das Feld mit dem Feld-Namen "ort" in der aktuellen Zeile gesetzt ist, dann müssen Sie die Exist()-Methode für die Collection einsetzen:

```
Dim cLine As Collection
...
cLine = hCSVFile.Read()
If Not cLine.Exist("ort") Then ...
```

Hier sind weitere Hinweise für die Erzeugung und Bearbeitung von CSV-Dateien - gefunden auf der Website https://www.thoughtspot.com/blog/6-rules-creating-valid-csv-files - von Jon Avrach, deren Beachtung vor allem beim Einsatz der CSV-Dateien beim Daten-Import in Datenbanken Probleme minimieren:

- Trennen Sie Datenfelder mit einem Feld-Trennzeichen (Separator), in der Regel ein Komma. Wenn Sie kein Komma benutzen, sollten Sie zum Beispiel ein Tab-Zeichen oder ein Pipe-Zeichen oder ein Semikolon wählen.
- Geben Sie der Datei in der ersten Zeile einen vollständigen Header mit einer Liste der Feld-Namen mit. Dies ist zwar optional, wird aber dringend empfohlen, denn es wirkt wie eine Dokumentation des Inhalts der Datei, da man aus den Feldnamen oft auf den Datentyp der Feld-Werte schließen kann.
- Stellen Sie sicher, dass der Header auf die gleiche Weise wie die weiteren Zeilen mit dem *gleichen* Feld-Trennzeichen und dem *gleichen* Escape-Zeichen – wenn es verwendet wird – formatiert wird.
- Halten Sie jeden Datensatz auf einer separaten Zeile. Dabei gilt: Jeder Datensatz muss auf ei-

ner eigenen Zeile beginnen. Ein einziger Datensatz kann aber mehrere Zeilen überspannen.
- Nach dem letzten Datensatz sollte die letzte Zeile *ohne* ein Zeilenende-Zeichen enden.
- Denken Sie daran, dass das umschließende Escape-Zeichen (typischerweise doppelte Anführungszeichen) verwendet werden *muss*, wenn zum Beispiel das Feld-Trennzeichen in einem Feld vorhanden ist wie bei Zahlen in der (deutschen) Notation x*,y* wie bei 23,345.

Für weitere Details zu diesen Hinweisen können Sie sich auf Wikipedia mit passenden Suchkriterien und in den RFC 4180 (CSV-Spezifikation) gut informieren.

Bei den Projekten zum Thema CSV wurden 2 Fälle unterschieden. Im ersten Fall geht es um das Schreiben von CSV-Dateien mit Daten aus unterschiedlichen Quellen. Die Quellen waren Daten in Steuerelementen wie zum Beispiel GridView oder TextArea oder XML-Dateien oder Datenbank-Tabellen. Als Quellen für den zweiten Fall dienten CSV-Dateien, deren Inhalt ausgelesen wird und in unterschiedlicher Weise verarbeitet, abgespeichert oder angezeigt wird. Nur für diesen Fall kann die Klasse CsvFile aus der Komponente gb.util eingesetzt werden.

Im Kapitel → https://www.gambas-buch.de/doku.php?id=k17:k17.7:k17.7.6:start finden Sie ein Projekt zum Export von ausgewählten Daten in eine CSV-Datei und im Kapitel unter https://www.gambas-buch.de/doku.php?id=k17:k17.7:k17.7.7:start ein Projekt zum Daten-Import aus einer CSV-Datei. Interessant ist in beiden Fällen die Möglichkeit sowohl das Separator-Zeichen als auch das Escape-Zeichen anzugeben. Sie können auch festlegen, ob die Feldnamen als Header mit gespeichert werden sollen oder in einer Vorschau feststellen, ob eine (erste) Zeile mit den Feld-Namen als Header existiert.

Abbildung 6.13.3.1: Daten-Export

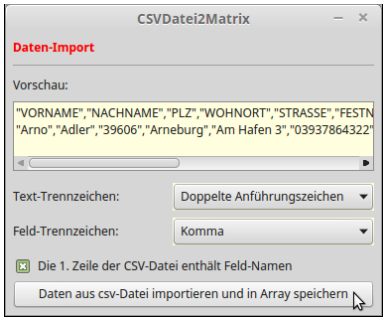

Abbildung 6.13.3.2: Daten-Import

Die Archive für diese beiden überarbeiteten Projekte finden Sie im Downloadbereich.

6.13.4 Projekt 1 – CSV2XML

Das erste Projekt liest zuerst im Dialog eine CSV-Datei ein. Dann wird der Inhalt der CSV-Datei so präpariert, dass

- Leerzeilen ignoniert werden,
- Feld-Werte ohne Escape-Zeichen durch das fest vorgegeben Escape-Zeichen 'eingewickelt' werden,
- fehlende Feldwerte durch einen leeren String oder durch NULL ersetzt werden und
- geprüft wird, dass die letzte Zeile kein Zeilenende-Zeichen '\n' besitzt.

Der so präparierte Inhalt wird unter dem originalen Datei-Namen abgespeichert. Anschließend wird

der Inhalt der präparierten CSV-Datei unter Angabe des Feld-Trennzeichens (Separator) und des Escape-Zeichens mit einem neuen CsvFile-Objekt dekodiert und in ein XML-Dokument (Kapitel 27.2.1) geschrieben. Abschließend wird das XML-Dokument abgespeichert. Eine Prüfung, ob die CSV-Datei in der ersten Zeile eine Liste der Feld-Namen enthält, erfolgt nicht. Ein Header wird einfach erwartet, weil sonst der Einsatz der Klasse CsvFile nicht möglich ist!

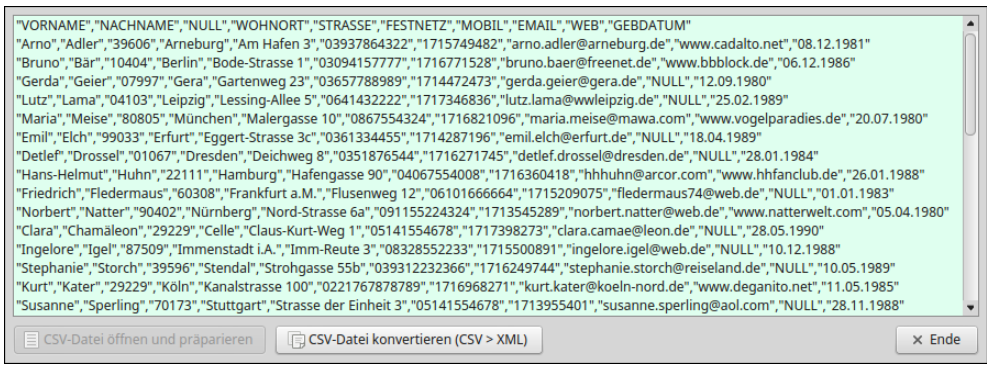

Abbildung 6.13.4.1: Inhalt der präparierten CSV-Datei

Abbildung 6.13.4.2: Ausschnitt aus dem Inhalt der XML-Datei

Der Quelltext wird vollständig angegeben und kommentiert:

```
[1]  ' Gambas class file
[2]
[3]  Private $sSeparator As String
[4]  Private $sEscape As String
[5]  Public hXmlWriter As XmlWriter
[6]  Public sFilePath As String
[7]  Public hCSVFile As CsvFile
[8]
[9]  Public Sub Form_Open()
[10]   FMain.Center
[11]   FMain.Resizable = True
[12]   FMain.Caption = "CSV2XML | Gambas-Version: " & System.FullVersion
[13]
[14]   $sSeparator = ","
[15]   $sEscape = "\"" ' Alternative: Chr(34) => "
[16]
[17]   btnConvertAndSave.Enabled = False
[18]
[19] End ' Form_Open()
[20]
[21] Public Sub btnPrepareAndSave_Click()
[22]   sFilePath = FileOpen()
[23]   If sFilePath Then
[24]      txaPrepare.Text = Prepare(sFilePath)
[25]      txaPrepare.Pos = 0
[26]      File.Save(sFilePath, Prepare(sFilePath))
[27]      Wait
[28]      btnPrepareAndSave.Enabled = False
[29]      btnConvertAndSave.Enabled = True
[30]   Else
[31]     Return
[32]   Endif
[33] End
[34]
[35] Public Sub btnConvertAndSave_Click()
```

```
[36]   WriteXML()
[37]   btnPrepareAndSave.Enabled = True
[38]   btnConvertAndSave.Enabled = False
[39] End
[40]
[41] Public Sub btnClose_Click()
[42]   FMain.Close()
[43] End ' Close
[44]
[45] Private Function Prepare(sFilePath As String) As String
[46]
[47]   Dim sCSVContent, sCSVChanged, sLine, sField, sNewLine As String
[48]
[49]   sCSVContent = File.Load(sFilePath)
[50]
[51]   For Each sLine In Split(sCSVContent, gb.NewLine, $sEscape, False, True)
[52]     sLine = Trim(sLine)
[53]     If Not sLine Then Continue ' Leerzeilen werden ignoriert
[54]     sNewLine = ""
[55]     For Each sField In Split(sLine, $sSeparator, $sEscape, False, True)
[56]       If sField Not Begins $sEscape And If sField Not Ends $sEscape Then
[57]          sField = $sEscape & sField & $sEscape
[58]       Endif
[59]       If sField = $sEscape & $sEscape Or If sField = $sEscape & " " & $sEscape Then
[60]        ' sField = $sEscape & "" & $sEscape ' Fall 1
[61]          sField = "NULL"                   ' Fall 2
[62]       Endif
[63]       sNewLine &= sField & $sSeparator
[64]     Next
[65]      If sNewLine Ends $sSeparator Then sNewLine = Left(sNewLine, -1)
[66]      sCSVChanged &= sNewLine & gb.NewLine
[67]   Next
[68]
[69]   sCSVChanged = Left(sCSVChanged, -1) ' Zeilenende-Zeichen der *letzten* Zeile wird entfernt
[70]
[71]   Return sCSVChanged
[72]
[73] End ' Prepare(...)
[74]
[75] Private Sub WriteXML()
[76]
[77]   Dim k As Integer
[78]   Dim cLine As Collection
[79]   Dim sField As String
[80]
[81]   hCSVFile = New CsvFile(sFilePath, $sSeparator, $sEscape)
[82]
[83]   hXmlWriter = New XmlWriter
[84]   hXmlWriter.Open(Null, True, "UTF-8")
[85]
[86]   hXmlWriter.PI("Document", "version=\"" & Format(Now, "dd.mm.yyyy-hh:nn") & "\"")
[87]
[88]   hXmlWriter.StartElement("root")
[89]   hXmlWriter.Comment("DataBase: " & File.Name(sFilePath))
[90]
[91]   While Not hCSVFile.Eof
[92]     hXmlWriter.StartElement("item")
[93]     cLine = hCSVFile.Read()
[94]     k = 0
[95]     For Each sField In hCSVFile.Fields
[96]       If sField = "null" Then
[97]          hXmlWriter.Element("field_" & CStr(k + 1), cLine[sField])
[98]       Else
[99]          hXmlWriter.Element(sField, cLine[sField])
[100]      Endif
[101]      Inc k
[102]    Next
[103]    hXmlWriter.EndElement
[104]  Wend
[105]
[106]   hXmlWriter.EndElement ' root
[107]   hXmlWriter.EndDocument
[108]   txaPrepare.Text = hXmlWriter.Data
[109]   txaPrepare.Pos = 0
[110]   File.Save(File.Dir(sFilePath) &/ File.BaseName(sFilePath) & ".xml", hXmlWriter.Data)
[111]
[112] End
[113]
[114] Private Function FileOpen() As String
[115]   Dialog.Title = "Importieren Sie eine csv-Datei!"
[116]   Dialog.Filter = ["*.csv", "csv-Dateien"]
[117]   Dialog.Path = Application.Path &/ "CSV"
[118]   If Dialog.OpenFile(False) = True Then ' Multiselect=False (Standard)
[119]      Message.Info("Das Öffnen der csv-Datei wurde abgebrochen!")
```

```
[120]       Return
[121]    Else
[122]       Return Dialog.Path
[123]    Endif
[124]  End ' FileOpen()
```

Kommentar:

- In den Zeilen 45 bis 73 wird der Inhalt des CSV-Datei – geöffnet in Zeile 49 – nach den o.a. Vorgaben präpariert.
- Dazu wird zuerst in der Zeile 51 eine Zeile gelesen und dann in Zeile 55 jedes Feld in der aktuellen Zeile untersucht und anschließend in Zeile 63 jeweils eine neue Zeile feldweise erzeugt.
- Der geänderte Inhalt der CSV-Datei wird in der Zeile 24 in einer TextArea angezeigt und in der Zeile 26 in eine CSV-Datei mit originalem Pfad gespeichert.
- In den Zeilen 75 bis 112 wird die XML-Datei geschrieben. Benutzt wird die Klasse XmlWriter → Kapitel 27.2.0. Die Hauptlast trägt die While-Kontrollstruktur in den Zeilen 91 bis 104. Wie Sie erkennen können, werden fast alle Eigenschaften und die Read()-Methode der Klasse CsvFile eingesetzt. Beachten Sie, dass die Read()-Methode als Funktionswert eine Collection zurückgibt, in der alle Schlüssel-Wert-Paare einer Zeile enthalten sind.
- Die Kontrollstruktur in den Zeilen 96 bis 100 erzeugt für nicht vorhandene Feld-Namen im Header Pseudo-Feldnamen, die aus einem #-Zeichen mit nachgestelltem Feld-Index bestehen.

6.13.5 Exkurs – CSV2XML

In diesem Exkurs wird Ihnen ein (Bash-)Skript vorgestellt, mit dem Sie ebenso den Inhalt einer CSV-Datei auf die Schnelle in den Inhalt einer XML-Datei transformieren. Das ausführbare Skript erwartet genau 3 Parameter:

- den Datei-Namen der CSV-Datei,
- das Feld-Trennzeichen und das
- das Escape-Zeichen:

```
hans@mint71 ~/GB3BUCH/.../Projekte/CSV_READ/SH-Skript $ ./test.sh "db.csv" "," "\""
```

Ist die Anzahl der Parameter *nicht* 3, dann gibt es eine Fehlermeldung in *roter Schrift* mit einem Hinweis auf die korrekte Syntax:

```
Syntax: ./test.sh 'CSV-File-Path' 'Field separator' 'Escape character'
# hans@mint71 ~/GB3BUCH/.../Projekte/CSV_READ/SH-Skript $ ./test.sh "db.csv" ","
```

Wenn das der Inhalt der CSV-Datei ist:

```
"Vorname","Alter","Wohnort"
"Hans","68","Pößneck"
"Peter","56","Cork"
"Anna","18","Jena"
```

dann sehen Sie hier das Ergebnis in der XML-Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<Data>
  <Record>
    <Vorname>Hans</Vorname>
    <Alter>68</Alter>
    <Wohnort>Pößneck</Wohnort>
  </Record>
  <Record>
    <Vorname>Peter</Vorname>
    <Alter>56</Alter>
    <Wohnort>Cork</Wohnort>
  </Record>
  <Record>
    <Vorname>Anna</Vorname>
    <Alter>18</Alter>
    <Wohnort>Jena</Wohnort>
  </Record>
</Data>
```

Der erprobte Quelltext vom (Bash-)Skript ist gut dokumentiert und wird vollständig angegeben:

```
#!/bin/bash # She-Bang Zeile → Bash-Interpreter

  if [ $# -ne 3 ] # Wenn die Anzahl der Parameter (= $#) ungleich 3 ist ...
  then
    # FARB-WERTE:
    RO="\033[31m"     # rot
    NO="\033[0m"      # normal
    echo -e "${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
  # Liste der Feld-Namen im Array 'header' speichern -> Operator =()
  header=($header_line)

  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[*]})

  # 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)

    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

  exit 0
```

6.13.6 Projekt 2 – CSV2GRID

Im zweiten Projekt wird eine CSV-Datei gelesen. Die mit der CsvFile-Read-Methode dekodierten Daten werden in einer GridView angezeigt.

Quelltext:

```
' Gambas class file

Public Sub btnReadCSV_Click()

  Dim i, iVal As Integer
  Dim sField As New String[]
  Dim sKey, sSeparator, sEscape As String
  Dim hCsvFile As CsvFile
  Dim cLine As Collection
  sSeparator = ","
  sEscape = "\"" ' Chr$(34)
' Open CSV file
  Try hCsvFile = New CsvFile(FileOpen(), sSeparator, sEscape)
  If Error Then Return
  GridView1.Columns.Count = 0
' Prepare header
  GridView1.Header = True
  sField = hCsvFile.Fields
  GridView1.Columns.Count = sField.Count
  For i = 0 To sField.Count - 1
    GridView1.Columns[i].Title = UCase(sField[i])
  Next
' Read CSV data
  While Not hCsvFile.Eof
    cLine = hCsvFile.Read()
    If cLine.Count > 0 Then ' Collection not empty?
       Inc GridView1.Rows.Count
       iVal = 0
       For Each sKey In hCsvFile.Fields
         Gridview1[GridView1.Rows.Count - 1, iVal].text = cLine[sKey]
         Inc iVal
       Next
    Endif
  Wend
 GridView1.MoveTo(0, 0)
 GridView1.Columns.Width = -1 ' Adjust column width automatically
End

Private Function FileOpen() As String

  Dialog.Title = ("Import csv-File!")
  Dialog.Filter = ["*.csv", "csv-Files"]
  Dialog.Path = Application.Path &/ "Files"
  If Dialog.OpenFile(False) = True Then ' Multiselect=False (Standard)
     Message.Info(("Opening the csv file has been canceled!"))
     Return "break"
  Else
     Return Dialog.Path
  Endif

End ' FileOpen()
```

	VORNAME	NACHNAME	PLZ	WOHNORT	STRASSE	FESTNETZ	MOBIL	EMAIL	WEB	GEBDATUM
1	Arno	Adler	39606	Arneburg	Am Hafen 3	03937864322	1715749482	arno.adler@arneburg.de	www.cadalto.net	08.12.1981
2	Bruno	Bär	10404	Berlin	Bode-Strasse 1	03094157777	1716771528	bruno.baer@freenet.de	www.bbblock.de	06.12.1986
3	Gerda	Geier	07997	Gera	Gartenweg 23	03657788989	1714472473	gerda.geier@gera.de		12.09.1980
4	Lutz	Lama	04103	Leipzig	Lessing-Allee 5	0641432222	1717346836	lutz.lama@wwleipzig.de		25.02.1989
5	Maria	Meise	80805	München	Malergasse 10	0867554324	1716821096	maria.meise@mawa.com	www.vogelparadies.de	20.07.1980
6	Emil	Elch	NULL	Erfurt	Eggert-Strasse 3c	0361334455	1714287196	emil.elch@erfurt.de		18.04.1989
7	Detlef	Drossel	01067	Dresden	Deichweg 8	0351876544	1716271745	detlef.drossel@dresden.de		28.01.1984
8	Hans-Helmut	Huhn	22111	Hamburg	Hafengasse 90	04067554008	1716360418	hhhuhn@arcor.com	www.hhfanclub.de	26.01.1988
9	Friedrich	Fledermaus	60308	Frankfurt a.M.	Flusenweg 12	06101666664	1715209075	fledermaus74@web.de		01.01.1983
10	Norbert	Natter	90402	Nürnberg	Nord-Strasse 6a	091155224324	1713545289	norbert.natter@web.de	www.natterwelt.com	05.04.1980
11	Clara	Chamäleon	29229	Celle	Claus-Kurt-Weg 1	05141554678	1717398273	clara.camae@leon.de		28.05.1990
12	Ingelore	Igel	87509	Immenstadt i.A.	Imm-Reute 3	08328552233	1715500891	ingelore.igel@web.de		10.12.1988
13	Stephanie	Storch	39596	Stendal	Strohgasse 55b	039312232366	1716249744	stephanie.storch@reiseland.de		10.05.1989

CSV_Datei lesen

Abbildung 6.13.6.1: Daten aus einer CSV-Datei – angezeigt in einer GridView

6.13.7 Projekt 3 – QSLITE2CSV

Das Projekt 3 widmet sich dem Fall, dass Daten aus einer Datenbank-Tabelle in einer CSV-Datei gespeichert werden. Achtung: Bei den Datenbank-Tabellen bieten MyQSL, PostgreSQL oder SQlite Dump- oder Copy-Methoden an, mit denen Daten direkt in eine CSV-Datei abgespeichert werden können. Deshalb wird nur die Methode vorgestellt, in der die selektierten Daten aus einer DB-Tabelle in einen String konvertiert werden, der den Inhalt einer CSV-Datei repräsentiert:

```
Private Function GetCSVData() As String

  Dim sCSVLine, sRequest, sSeparator, sEscape As String
  Dim hRequest As SQLRequest
  Dim dbResult As Result
  Dim rField As ResultField

  hRequest = hConnection.SQL.Select("").From(cmbDBTabellen.Text)
  sRequest = hRequest.Get()
  dbResult = hConnection.Exec(sRequest)

  If dbResult.Count = 0 Then
     Message(("The number of selected records is zero!"))
     Return
  Endif

  FMain.Caption = "Daten-Export aus Datenbank: '"
  FMain.Caption &= File.Name(sDBFilePath) & "' "
  FMain.Caption &= " Tabelle: '" & cmbDBTabellen.Text
  FMain.Caption &= "' in CSV-Datei"

  sSeparator = ","   ' Feld-Trenn-Zeichen
  sEscape = "\""     ' Text-Trennzeichen

' Field list
  For Each rField In dbResult.Fields
    sCSVLine &= sEscape & Upper(rField.Name) & sEscape & sSeparator
  Next
  sCSVLine = Left(sCSVLine, -1) ' Last field name WITHOUT field separator
  sCSVLine &= gb.NewLine

' Data lines
  For Each dbResult
    For Each rField In dbResult.Fields
      If dbResult[rField.Name] = "" Then
         sCSVLine &= sEscape & "NULL" & sEscape & sSeparator
      Else
         sCSVLine &= sEscape & dbResult[rField.Name] & sEscape & sSeparator
      Endif
    Next
    sCSVLine = Left(sCSVLine, -1) ' Last field name WITHOUT field separator
    sCSVLine &= gb.NewLine
  Next

  sCSVLine = Left(sCSVLine, -1) ' Last line WITHOUT gb.NewLine
  Return sCSVLine

End
```

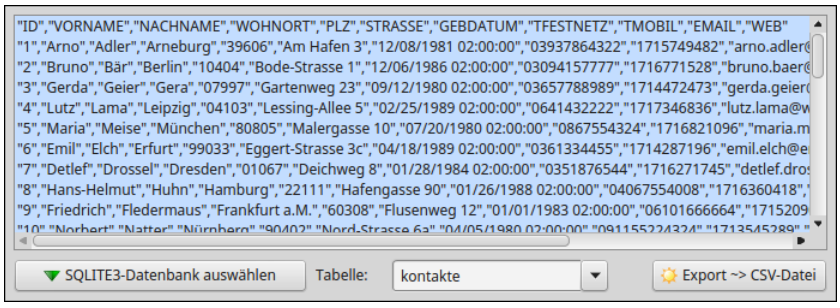

Abbildung 6.13.7.1: Daten-Export DB-Daten nach CSV-Datei

In der TextArea sehen Sie einen Ausschnitt aus dem Inhalt der CSV-Datei.

Hinweis: Im Kapitel https://www.gambas-buch.de/doku.php?id=k27:k27.6:start#projekt_2xml_csv wird ein Projekt vorgestellt, bei dem die Transformation XML → CSV umgesetzt wird.