ADO Datenbank XML-Export

datapacket Version 2.0

Um einfach einmal einige lesende Zugriffe auf die Datenbank zu realisieren, ist eine Export-Routine immer sehr aufschlussreich, in einen weiteren Beispiel möchte ich dann auch noch einen SQL-Dump erzeugen, aber dazu später!

ADOXMLExport.lpr (/code/ADOXMLExport.lpr?mode=download) Pascal (7,96 kByte) 26.12.2013 01:16

```
program ADOXMLExport;
{$APPTYPE CONSOLE}
{$mode objfpc}{$H+}
uses
  Classes, // for TFileStream
  SysUtils, ActiveX, Variants,
  Interfaces, // used packages: LazActiveX
  adodb_6_1_tlb; // Microsoft ActiveX Data Objects 2.x-Objektbibliothek

function IIF(b: boolean; sTrue: string; sFalse: string = ''): string;
begin if b then result := sTrue else result := sFalse; end;

procedure CreateXMLExport(const Filename: string);
  function getAsString(value: oleVariant): UTF8String;
  var
    index, lowVal, highVal: integer;
    oleArray: PSafeArray;
    oleObj: oleVariant;
  begin
    try
      case TVarData(Value).vType of
        varEmpty, varNull:
          result := '';
        varSmallint, varInteger, varByte, varError:
          result := IntToStr(Value);
        varSingle, varDouble, varCurrency:
          result := FloatToStr(Value);
        varDate:
          result := DateTimeToStr(Value);
        varOleStr, varStrArg, varString:
          result := UTF8Encode(Widestring(Value));
        varBoolean:
          if Value then
            result := 'true'
          else
            result := 'false';
        varDispatch, varVariant, varUnknown, varTypeMask:
          begin
            VarAsType(Value, varOleStr);
            result := UTF8Encode(Widestring(Value));
          end;
        else
```

```
          if VarIsArray(Value) then
          begin
            VarArrayLock(Value);
            index := VarArrayDimCount(Value);
            lowVal := VarArrayLowBound(Value, index);
            highVal := VarArrayHighBound(Value, index);
            oleArray := TVariantArg(Value).pArray;

            for index := lowVal to highVal do
            begin
              SafeArrayGetElement(oleArray, @index, oleObj);
              result := result + getAsString(oleObj) + #13#10;
            end;

            VarArrayUnlock(Value);
            Delete(result, length(result) - 1, 2);
          end
          else
            result := ''; // varAny, varByRef
      end;
    except
      result := '#error#'
    end;
  end;

var
  cn: _Connection;
  rsTables, rsTable, rsKeys: _Recordset;
  sTable: Widestring;
  s, lcid, pk, fk, index: UTF8String;
  f: TFileStream;
  i: integer;
begin
//connect db
  cn := CoConnection.Create;
  cn.Open('Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' + Filename, '',
'', 0);
  if cn.State = adStateOpen then
  begin
    rsTables := CoRecordset.Create;
    rsKeys := CoRecordset.Create;
    rsTable := CoRecordset.Create;
    try
      rsTables := cn.OpenSchema(adSchemaTables, EmptyParam, EmptyParam);
      while not (rsTables.EOF_) do
      begin
        if (rsTables.Fields['TABLE_TYPE'].Value = 'TABLE') then // only
tables
        begin
          sTable := rsTables.Fields['TABLE_NAME'].Value;
```

```
          f := TFileStream.Create(sTable + '.xml', fmOpenWrite or fmCreate);
          s := '<?xml version="1.0" encoding="UTF-8" standalone="yes"?
>'#13#10 +
//                '<?xml-stylesheet version="1.0" href="datenpacket.xsl"
type="text/xsl"?>'#13#10 +
                '<datapacket Version="2.0">'#13#10 +
                '  <metadata>'#13#10;
          f.Write(Pchar(s)^, Length(s));

          // table properties
          s := '    <fields>'#13#10;
          f.Write(Pchar(s)^, Length(s));
          lcid := '';
          rsTable.Open(sTable, cn, adOpenForwardOnly, adLockReadOnly,
adCmdTable);
          for i := 0 to rsTable.Fields.Count-1 do
          begin
            case rsTable.Fields[i].Type_ of
              adBoolean: s := '"boolean"';
              adDate, adDBDate: s := '"date"';
              adDBTime: s := '"time"';
              adDBTimeStamp: s := '"datetime"';
              adTinyInt, adSmallInt, adInteger, adBigInt, adUnsignedTinyInt,
adUnsignedSmallInt,
              adUnsignedInt, adUnsignedBigInt:
                  if
rsTable.Fields[i].Properties['ISAUTOINCREMENT'].Value='True' then
                    s := '"i4" AUTOINCREMENT="true"'
                  else
                    s := '"i4"';
              adSingle, adDouble, adDecimal, adNumeric:
                s := '"r8" PRECISION="' +
IntToStr(rsTable.Fields[i].Precision) + '" NUMERICALSCALE="' +
IntToStr(rsTable.Fields[i].NumericScale) + '"';
              adCurrency: s := '"r8" SUBTYPE="Money"';
              adBSTR, adChar, adVarChar, adLongVarChar, adWChar, adVarWChar,
adLongVarWChar:
                s := '"string" WIDTH="' +
IntToStr(rsTable.Fields[i].DefinedSize) + '"';
            end;
            s := '      <field attrname="' + rsTable.Fields[i].Name + '"
fieldtype=' + s + ' />'#13#10;
            f.Write(Pchar(s)^, Length(s));
            lcid :=
getAsString(rsTable.Fields[0].Properties['COLLATINGSEQUENCE'].Value);
          end;

          // primary key & index
          pk := '';
          index := '';
```

```
          rsKeys := cn.OpenSchema(adSchemaIndexes, VarArrayOf([Unassigned,
Unassigned, Unassigned, Unassigned, sTable]), EmptyParam);
          while not rsKeys.EOF_ do
          begin
            s := getAsString(rsKeys.Fields['INDEX_NAME'].Value);
            if s = 'PrimaryKey' then
              pk := pk + IIF(pk<>'', ' ') +
getAsString(rsKeys.Fields['COLUMN_NAME'].Value)
            else
              index := index + IIF(index<>'', ' ') + s;
            rsKeys.MoveNext;
          end;
          rsKeys.Close;

          // foreign keys
          fk := '';
          rsKeys := cn.OpenSchema(adSchemaForeignKeys,
VarArrayOf([UnAssigned, UnAssigned, sTable]), EmptyParam);
          while not rsKeys.EOF_ do
          begin
            s := '';
            if TVarData(rsKeys.Fields['PK_COLUMN_NAME'].Value).vType <>
varNull then
              s := 'FOREIGN KEY (`' +
getAsString(rsKeys.Fields['PK_COLUMN_NAME'].Value) + '`)';
            if TVarData(rsKeys.Fields['FK_TABLE_NAME'].Value).vType <>
varNull then
              s := s + ' REFERENCES `' +
getAsString(rsKeys.Fields['FK_TABLE_NAME'].Value) + '`';
            if TVarData(rsKeys.Fields['FK_COLUMN_NAME'].Value).vType <>
varNull then
              s := s + ' (`' +
getAsString(rsKeys.Fields['FK_COLUMN_NAME'].Value) + '`)';
            fk := fk + IIF(fk<>'', ', ') + s;
            rsKeys.MoveNext;
          end;
          rsKeys.Close;

          s := '    </fields>'#13#10 +
            '    <params primary_key="' + pk + '" lcid="' + lcid + '"
index="' + index + '" foreign_key="' + fk + '" />'#13#10 +
            '  </metadata>'#13#10;
          f.Write(PChar(s)^, Length(s));

          // values
          if not rsTable.EOF_ then
          begin
            s := '  <rowdata>'#13#10;
            f.Write(Pchar(s)^, Length(s));
            while not rsTable.EOF_ do
```

```
            begin
              s := '    <row>'#13#10;
              f.Write(Pchar(s)^, Length(s));
              for i := 0 to rsTable.Fields.Count-1 do
              begin
                s := getAsString(rsTable.Fields[i].Value);
                s := StringReplace(s, '\', '\\', [rfREplaceAll]);
                s := StringReplace(s, #13#10, '\n', [rfREplaceAll]);
                s := StringReplace(s, '&', '&amp;', [rfREplaceAll]);
                s := StringReplace(s, '<', '&lt;', [rfReplaceAll]);
                s := StringReplace(s, '>', '&gt;', [rfReplaceAll]);
                s := '      <col name="' + rsTable.Fields[i].Name + '">' + s
+ '</col>'#13#10;
                f.Write(Pchar(s)^, Length(s));
              end;
              s := '    </row>'#13#10;
              f.Write(Pchar(s)^, Length(s));
              rsTable.MoveNext;
            end;
             s := '  </rowdata>'#13#10;
             f.Write(Pchar(s)^, Length(s));
          end;
          rsTable.Close;
          s := '</datapacket>';
          f.Write(PChar(s)^, Length(s));
          f.Free;
        end;
        rsTables.MoveNext;
      end;
      rsTables.Close;
    finally
      rsTable := nil;
      rsKeys := nil;
      rsTables := nil;;
    end;
    cn.Close;
  end;
  cn := nil;
end;

begin
  CreateXMLExport(ParamStr(1));
end.
```

Natürlich ließe sich diese Routine auch gut mit dem Beispiel ADO ConnectionString (/ADO-ConnectionString) verbinden so müsste lediglich der Result-Wert also der komplette ConnectionString an die Procedure CreateXMLExport übergeben werden.

Es müssten dafür die Funktion getConnectionString in die Datei aufgenommen werden und natürlich die Benutzten Dateien aus dem Bereich uses zur Verfügung stehen! Diese Möglichkeiten stehen natürlich

auch bei dem SQL-Dump zur Verfügung, ich wollte in den Beispielen jedoch nicht zu viele Bedingungen kombinieren, damit sie nicht zu komplex werden und abschreckend sind!

```
procedure CreateXMLExport(const Filename: string);
// change to -> procedure CreateXMLExport(const ConnectionString: string);
cn.Open('Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' + Filename, '', '',
0);
// change to -> cn.Open(ConnectionString, '', '', 0);
CreateXMLExport(ParamStr(1));
// change to -> CreateXMLExport(getConnectionString);
```

Autor: Udo Schmal (#udo.schmal) , veröffentlicht: 22.12.2013, letzte Änderung: 06.09.2023