ADO Klassen Beispiel

Hier nun wie bereits angekündigt die beiden Beispiele Datenbank XML-Export und SQL Dump über ADO in über eigene Klassen. Die angelegten Klassen sind nicht vollständig sondern haben gerade mal eben die Eigenschaften und Funktionen die für die beiden Beispiele benötigt werden oder meiner Meinung sinnvoll sind!

Der Gesamte resultierende Code ist nun länger, die Ausführungszeit wird natürlich auch minimal langsamer aber der eigentliche Code der beiden Export Routinen ist nun recht überschaubar. Für weitere Entwicklungen ergibt sich nun der Vorteil das die Feld-Attribute direkt zu nutzen sind und nicht über weitere Abfragen ermittelt werden müssen.

ADOClassExample.lpr (/code/ADOClassExample.lpr?mode=download) Pascal (24,95 kByte) 01.01.2014 23:22

```
program ADODBDump;
{$APPTYPE CONSOLE}
{$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
  adox_6_0_tlb; // ADO Extensions 2.5 for DDL and Security Library Reference

type
  TResultType = (rtSet, rtDump, rtXml);

  TADOConnection = class(TComponent)
  private
    FConnection: _Connection;
    FCatalog: _Catalog;
    FTables: TStringList;
    function GetProvider: UTF8String;
    procedure SetProvider(const AProvider: UTF8String);
    function GetProperty(const Idx: OleVariant): OleVariant;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function Open(const AConnectionString: string = ''): boolean;
    procedure Close();
    property Provider: UTF8String read GetProvider write SetProvider;
    property Properties[const Idx: OleVariant]: OleVariant read GetProperty;
    property Tables: TStringList read FTables write FTables;
  end;

  TADOField = class(TComponent)
  private
    FField: Field;
    FBaseTableName: UTF8String;
    FPrimaryKey: boolean;
    FIndex: boolean;
    FForeignKey: UTF8String;
    Flcid: UTF8String;
    FDefinedSize: integer;
    FPrecision: integer;
    FNumericScale: integer;
    FSigned: boolean;
    FIsNullable: boolean;
    FDefault: UTF8String;
    FIsAutoIncrement: boolean;
    function GetName: WideString;
    function GetNameAsUTF8String: UTF8String;
    function GetValue: OleVariant;
    procedure SetValue(Value: OleVariant);
    function GetProperty(const Idx: WideString): OleVariant;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function GetValueAsUTF8String(AResultType: TResultType = rtSet): UTF8String;
    function GetTypeAsUTF8String(AResultType: TResultType = rtSet): UTF8String;
    property Name: WideString read GetName;
    property UTF8Name: UTF8String read GetNameAsUTF8String;
    property Value: OleVariant read GetValue write SetValue;
    property AsUTF8String: UTF8String read GetValueAsUTF8String;
    property TypeAsUTF8String: UTF8String read GetTypeAsUTF8String;
    property Properties[const Idx: WideString]: OleVariant read GetProperty;
```

```
    property PrimaryKey: boolean read FPrimaryKey;
    property Index: boolean read FIndex;
    property ForeignKey: UTF8String read FForeignKey;
    property lcid: UTF8String read Flcid;
    property DefinedSize: integer read FDEfinedSize;
    property Precision: integer read FPrecision;
    property NumericScale: integer read FNumericScale;
    property Signed: boolean read FSigned;
    property isNullable: boolean read FIsNullable;
    property DefaultValue: UTF8String read FDefault;
    property isAutoIncrement: boolean read FIsAutoIncrement;
  end;

  TADOFields = class(TComponent)
  private
    FFields: array of TADOField;
    procedure GetItems();
    procedure ClearItems();
    procedure UpdateItems();
    function GetItemById(const Index: integer): TADOField;
    function GetItemByName(const ItemName: WideString): TADOField;
    function GetItem(const Idx: OleVariant): TADOField;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function Count: integer;
    property Field[const Idx: OleVariant]: TADOField read GetItem; default;
  end;

  TADORecordset = class(TComponent)
  private
    FRecordset: _Recordset;
    FFields: TADOFields;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    function Open(const ASelect: string; ACursorType, ALockType, ACommandType: integer): boolean;
    procedure Close();
    function OpenDisconnected(const ASelect: string; ACursorType, ALockType, ACommandType: integer):
boolean;
    procedure CloseDisconnected();
    function BOF: boolean;
    function EOF: boolean;
    procedure MoveNext;
    property Fields: TADOFields read FFields write FFields;
  end;

function GetAsUTF8String(const value: OleVariant; AResultType: TResultType = rtSet): UTF8String;
var
  index, lowVal, highVal: integer;
  oleArray: PSafeArray;
  oleObj: OleVariant;
  s: string;
begin
  try
    case TVarData(Value).vType of
      varEmpty:
        result := '';
      varNull:
        if AResultType=rtSet then result := '' else result := 'NULL';
      varSmallint, varInteger, varByte, varError:
        result := IntToStr(Value);
      varSingle, varDouble, varCurrency:
        result := FloatToStr(Value);
      varDate:
         if AResultType=rtDump then
           result := '''' + DateTimeToStr(Value) + ''''
         else
           result := DateTimeToStr(Value);
      varOleStr, varStrArg, varString:
        begin
          s := Value;
          if AResultType<>rtSet then
          begin
            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]);
            if AResultType=rtDump then
            begin
              s := StringReplace(s, '''', '\''', [rfReplaceAll]);
              s := '''' + s + '''';
            end;
          end;
          result := UTF8Encode(s);
        end;
      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 + GetAsUTF8String(oleObj, AResultType) + #13#10;
          end;

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

function IIF(b: boolean; sTrue: UTF8String; sFalse: UTF8String = ''): UTF8String;
begin if b then result := sTrue else result := sFalse; end;

{$REGION ' - TADOConnection - '}
function TADOConnection.GetProvider: UTF8String;
begin
  result := UTF8Encode(FConnection.Provider);
end;

procedure TADOConnection.SetProvider(const AProvider: UTF8String);
begin
  FConnection.Provider := UTF8Decode(AProvider);
end;

function TADOConnection.GetProperty(const Idx: OleVariant): OleVariant;
begin
  result := FConnection.Properties[Idx];
end;

constructor TADOConnection.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FConnection := CoConnection.Create;
  FCatalog := CoCatalog.Create;
  FTables := TStringList.Create;
end;
```

```
destructor TADOConnection.Destroy;
begin
  if FConnection.State = adStateOpen then Close;
  FTables.Free;
  FCatalog := nil;
  FConnection := nil;
  inherited;
end;

function TADOConnection.Open(const AConnectionString: string = ''): boolean;
var rs: _Recordset;
begin
  result := false;
  FConnection.Open(WideString(AConnectionString), '', '', 0);
  if FConnection.State = adStateOpen then
  begin
    FCatalog.Set_ActiveConnection_(FConnection);
    rs := CoRecordset.Create;
    rs := FConnection.OpenSchema(adSchemaTables, EmptyParam, EmptyParam);
    while not (rs.EOF_) do
    begin
      if (rs.Fields['TABLE_TYPE'].Value = 'TABLE') then // only tables no 'SYSTEM TABLE' or 'VIEW'
        FTables.Add(rs.Fields['TABLE_NAME'].Value);
      rs.MoveNext;
    end;
    rs.Close;
    rs := nil;
    result := true;
  end;
end;

procedure TADOConnection.Close();
begin
  FCatalog.Set_ActiveConnection_(nil);
  FConnection.Close;
end;
{$ENDREGION}

{$REGION ' - TADOField - '}
function TADOField.GetName: WideString;
begin
  if (self = nil) or not assigned(FField) then
    result := ''
  else
    result := FField.Name;
end;

function TADOField.GetNameAsUTF8String: UTF8String;
begin
  if (self = nil) or not assigned(FField) then
    result := ''
  else
    result := UTF8Encode(FField.Name);
end;

function TADOField.GetValue: OleVariant;
begin
  if (self = nil) or not assigned(FField) then
    result := unassigned
  else
    result := FField.Value;
end;

procedure TADOField.SetValue(Value: OleVariant);
begin
  if not (self = nil) and assigned(FField) then
  FField.Value := Value;
end;

function TADOField.GetTypeAsUTF8String(AResultType: TResultType = rtSet): UTF8String;
begin
  result := '';
  if AResultType = rtXml then
  begin
    case FField.Type_ of
```

```
      adBoolean: result := '"boolean"';
      adDate, adDBDate: result := '"date"';
      adDBTime: result := '"time"';
      adDBTimeStamp: result := '"datetime"';
      adTinyInt, adSmallInt, adInteger, adBigInt, adUnsignedTinyInt, adUnsignedSmallInt,
      adUnsignedInt, adUnsignedBigInt:
        if FField.Properties['ISAUTOINCREMENT'].Value='True' then
          result := '"i4" AUTOINCREMENT="true"'
        else
          result := '"i4"';
      adSingle, adDouble, adDecimal, adNumeric:
        result := '"r8" PRECISION="' + IntToStr(FPrecision) + '" NUMERICALSCALE="' +
IntToStr(FNumericScale) + '"';
      adCurrency: result := '"r8" SUBTYPE="Money"';
      adBSTR, adChar, adVarChar, adLongVarChar, adWChar, adVarWChar, adLongVarWChar:
        result := '"string" WIDTH="' + IntToStr(FDefinedSize) + '"';
    end;
    result := 'fieldtype=' + result;
  end
  else
  begin
    if (((Owner As TADOFields).Owner As TADORecordset).Owner As TADOConnection).Provider =
'SQLOLEDB.1' then
      case FField.Type_ of
      adSmallInt:          result := 'SmallInt';
      adInteger:           result := 'Int';
      adSingle:            result := 'Real';
      adDouble:            result := 'Float';
      adCurrency:          result := 'Money';
      adDate:              result := 'DateTime';
      adBoolean:           result := 'Bit';
      adVariant:           result := 'Sql_Variant';
      adUnsignedTinyInt:   result := 'TinyInt';
      adBigInt:            result := 'BigInt';
      adGUID:              result := 'UniqueIdentifier';
      adBinary:            result := 'Binary';
      adChar:              result := 'Char';
      adWChar:             result := 'NChar';
      adNumeric:           result := 'Numeric';
      adDBTimeStamp:       result := 'DateTime';
      adVarChar:           result := 'VarChar';
      adLongVarChar:       result := 'Text';
      adVarWChar:          result := 'NVarChar';
      adLongVarWChar:      result := 'NText';
      adVarBinary:         result := 'VarBinary';
      adLongVarBinary:     result := 'Image';
      else                 result := 'Unknown';
      end
    else //if Provider = 'Microsoft.Jet.OLEDB.4.0' then
      case FField.Type_ of
      adSmallInt:          result := 'Integer';
      adInteger:           result := 'Integer';
      adSingle:            result := 'Single';
      adDouble:            result := 'Double';
      adCurrency:          result := 'Currency';
      adDate:              result := 'Date';
      adBoolean:           result := 'YesNo';
      adUnsignedTinyInt:   result := 'Byte';
      adGUID:              result := 'ReplicationID';
      adNumeric:           result := 'Decimal';
      adDBTimeStamp:       result := 'DateTime';
      adVarChar:           result := 'Text';
      adLongVarChar:       result := 'Memo';
      adVarWChar:          result := 'Text';
      adLongVarWChar:      result := 'Memo';
      adVarBinary:         result := 'ReplicationID';
      adLongVarBinary:     result := 'OLEObject';
      else                 result := 'Unknown';
      end;
    case FField.Type_ of
    adBinary, adBSTR, adChar, adWChar, adVarChar, adVarWChar:
      result := result + '(' + IntToStr(FDefinedSize) + ')';
    adSingle, adDouble, adCurrency, adNumeric, adVarNumeric:
      begin
        result := result + '(' + IntToStr(FPrecision) + ',' + IntToStr(FNumericScale) + ')';
```

```
        if not (FSigned) then result := result + ' unsigned';
      end;
    end;
    if not (FIsNullable) then result := result + ' NOT NULL';
    if FDefault<>'' then result := result + ' DEFAULT ' + FDefault;
    if FIsAutoIncrement then result := result + ' AUTO_INCREMENT';
  end;
end;

function TADOField.GetValueAsUTF8String(AResultType: TResultType = rtSet): UTF8String;
begin
  result := GetAsUTF8String(FField.Value, AResultType);
end;

function TADOField.GetProperty(const Idx: WideString): OleVariant;
var i: Integer;
begin
  if self = nil then result := unassigned;
  for i:=0 to FField.Properties.Count-1 do
    if FField.Properties[i].Name = Idx then
      Result := FField.Properties[i].Value;
end;

constructor TADOField.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FPrimaryKey := false;
  FIndex := false;
  FForeignKey := '';
  Flcid := '';
  FDefinedSize := 0;
  FPrecision := 0;
  FNumericScale := 0;
  FSigned := false;
  FIsNullable := false;
  FDefault := '';
  FIsAutoIncrement := false;
end;

destructor TADOField.Destroy;
begin
  inherited Destroy;
end;
{$ENDREGION}

{$REGION ' - TADOFields - '}
procedure TADOFields.GetItems();
var
  i: integer;
begin
  setLength(FFields, (Owner as TADORecordset).FRecordset.Fields.Count);
  for i := 0 to (Owner as TADORecordset).FRecordset.Fields.Count-1 do
  begin
    FFields[i] := TADOField.Create(Self);
    FFields[i].FField := (Owner as TADORecordset).FRecordset.Fields[i];
  end;
end;

procedure TADOFields.ClearItems();
var i: integer;
begin
  for i := length(FFields)-1 downto 0 do
    FFields[i].Free;
  setlength(FFields,0);
end;

procedure TADOFields.UpdateItems();
var i: integer;
begin
  for i := 0 to (Owner as TADORecordset).FRecordset.Fields.Count-1 do
    FFields[i].FField := (Owner as TADORecordset).FRecordset.Fields[i];
end;

function TADOFields.GetItemById(const Index: integer): TADOField;
begin
```

```
  if Index<length(FFields) then
    result := FFields[Index]
  else
    result := nil;
end;

function TADOFields.GetItemByName(const ItemName: WideString): TADOField;
var i: integer;
begin
  result := nil;
  for i:=0 to length(FFields)-1 do
    if lowercase(FFields[i].Name)=lowercase(ItemName) then
    begin
      result := FFields[i];
      Break;
    end;
end;

function TADOFields.GetItem(const Idx: OleVariant): TADOField;
var VarData : TVarData;
begin
  VarData := TVarData(Idx);
  case VarData.VType of
  varSmallInt, varInteger, varShortInt, varByte, varWord, varLongWord, varInt64 :
    result := GetItemById(Idx);
  else
    result := GetItemByName(Idx);
  end;
end;

function TADOFields.Count: integer;
begin
  result := length(FFields);
end;

constructor TADOFields.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  GetItems();
end;

destructor TADOFields.Destroy;
begin
  ClearItems();
  inherited Destroy;
end;
{$ENDREGION}

{$REGION ' - TADORecordset - '}
constructor TADORecordset.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FRecordset := CoRecordset.Create;
  FRecordset.Set_ActiveConnection((AOwner as TADOConnection).FConnection);
end;

destructor TADORecordset.Destroy;
begin
  if FRecordset.State = adStateOpen then Close;
  FRecordset := nil;
  inherited;
end;

function TADORecordset.Open(const ASelect: string; ACursorType, ALockType, ACommandType: integer):
boolean;
var
  rs: _Recordset;
  s: UTF8String;
  i: integer;
begin
  result := false;
  if FRecordset.State = adStateOpen then Close;
  FRecordset.Open(ASelect, EmptyParam, ACursorType, ALockType, ACommandType);
  FFields := TADOFields.Create(Self);
  if ACommandType = adCmdTable then
```

```
  begin
    rs := CoRecordset.Create;
    rs := (Owner As TADOConnection).FConnection.OpenSchema(adSchemaColumns, VarArrayOf([Unassigned,
Unassigned, ASelect, Unassigned]), EmptyParam);
    while not rs.EOF_ do
    begin
      if (TVarData(rs.Fields['COLUMN_NAME'].Value).vType <> varNull) and
         (TVarData(rs.Fields['COLUMN_HASDEFAULT'].Value).vType <> varNull) and
         rs.Fields['COLUMN_HASDEFAULT'].Value then
        FFields[rs.Fields['COLUMN_NAME'].Value].FDefault :=
GetAsUTF8String(rs.Fields['COLUMN_DEFAULT'].Value, rtDump);
      rs.MoveNext;
    end;
    rs.Close;

    rs := (Owner As TADOConnection).FConnection.OpenSchema(adSchemaIndexes, VarArrayOf([Unassigned,
Unassigned, Unassigned, Unassigned, ASelect]), EmptyParam);
    while not rs.EOF_ do
    begin
      if TVarData(rs.Fields['COLUMN_NAME'].Value).vType <> varNull then
      begin
        FFields[rs.Fields['COLUMN_NAME'].Value].FIndex := true;
        if (TVarData(rs.Fields['PRIMARY_KEY'].Value).vType <> varNull) and
rs.Fields['PRIMARY_KEY'].Value then
          FFields[rs.Fields['COLUMN_NAME'].Value].FPrimaryKey := true;
      end;
      rs.MoveNext;
    end;
    rs.Close;

     // find a better way to check if a Schema is supported
    if ((Owner As TADOConnection).Provider = 'Microsoft.Jet.OLEDB.4.0') then
    begin
      rs := (Owner As TADOConnection).FConnection.OpenSchema(adSchemaForeignKeys,
VarArrayOf([UnAssigned, UnAssigned, ASelect]), EmptyParam);
      while not rs.EOF_ do
      begin
        s := '';
        if TVarData(rs.Fields['FK_TABLE_NAME'].Value).vType <> varNull then
          s := 'REFERENCES `' + GetAsUTF8String(rs.Fields['FK_TABLE_NAME'].Value) + '`';
        if TVarData(rs.Fields['FK_COLUMN_NAME'].Value).vType <> varNull then
          s := s + ' (`' + GetAsUTF8String(rs.Fields['FK_COLUMN_NAME'].Value) + '`)';
        FFields[rs.Fields['PK_COLUMN_NAME'].Value].FForeignKey := s;
        rs.MoveNext;
      end;
      rs.Close;
    end;
    rs := nil;
  end;

  // field properties
  for i := 0 to Fields.Count-1 do
  begin
    case FFields[i].FField.Type_ of
    adBinary, adBSTR, adChar, adWChar, adVarChar, adVarWChar:
      FFields[i].FDefinedSize := FFields[i].FField.DefinedSize;
    adSingle, adDouble, adCurrency, adNumeric, adVarNumeric:
      begin
        FFields[i].FPrecision := FFields[i].FField.Precision;
        FFields[i].FNumericScale := FFields[i].FField.NumericScale;
        FFields[i].FSigned := (FFields[i].FField.Attributes and adParamSigned)<>0;
      end;
    end;
    FFields[i].FIsNullable := (FFields[i].FField.Attributes and adFldIsNullable)<>0;
    if FRecordset.CursorLocation <> adUseClient then
    begin
      FFields[i].Flcid := GetAsUTF8String(FFields[i].FField.Properties['COLLATINGSEQUENCE'].Value);
      FFields[i].FBaseTableName :=
GetAsUTF8String(FFields[i].FField.Properties['BASETABLENAME'].Value);
//      FFields[i].FDefault := GetAsUTF8String((Owner As
TADOConnection).FCatalog.Tables[FFields[i].FField.Properties['BASETABLENAME'].Value].Columns[FFields
[i].FField.Name].Properties['Default'].Value, rtDump);
      FFields[i].FIsAutoIncrement := FFields[i].FField.Properties['ISAUTOINCREMENT'].Value = 'True';
    end;
  end;
```

```
  result := true;
end;

procedure TADORecordset.Close();
begin
  if (FRecordset.State = adStateOpen) then
    FRecordset.Close;
end;

function TADORecordset.OpenDisconnected(const ASelect: string; ACursorType, ALockType, ACommandType:
integer): boolean;
begin
  FRecordset.CursorLocation := adUseClient;
  result := Open(ASelect, ACursorType, ALockType, ACommandType);
  FRecordset.Set_ActiveConnection(nil);
end;

procedure TADORecordset.CloseDisconnected();
begin
  FRecordset := nil;
  FRecordset := CoRecordset.Create;
  FRecordset.Set_ActiveConnection((Owner as TADOConnection).FConnection);
end;

function TADORecordset.BOF: boolean;
begin
  result := FRecordset.BOF;
end;

function TADORecordset.EOF: boolean;
begin
  result := FRecordset.EOF_;
end;

procedure TADORecordset.MoveNext;
begin
  if not FRecordset.EOF_ then
    FRecordset.MoveNext;
  FFields.UpdateItems();
end;
{$ENDREGION}

procedure WriteUTF8ToStream(f: TFileStream; s: UTF8String);
begin
  f.Write(Pchar(s)^, Length(s));
end;

procedure CreateDBDump(const Filename: string);
var
  cn: TADOConnection;
  rs: TADORecordset;
  s, sInsert, sCreateTable, sPk, sFk, sIndex: UTF8String;
  f: TFileStream;
  i, o: integer;
begin
  cn := TADOConnection.Create(nil);
  if cn.Open('Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' + Filename) then
  begin
    f := TFileStream.Create(ChangeFileExt(Filename, '.sql'), fmOpenWrite or fmCreate);
    rs := TADORecordset.Create(cn);
    for o:=0 to cn.Tables.Count-1 do
    begin
      sCreateTable := 'DROP TABLE IF EXISTS `' + cn.Tables[o] + '`;'#13#10 +
                      'CREATE TABLE IF NOT EXISTS `' + cn.Tables[o] + '` (' ;
      sInsert := ''; sPk := ''; sFk := ''; sIndex := '';
      rs.Open(cn.Tables[o], adOpenForwardOnly, adLockReadOnly, adCmdTable);
      for i := 0 to rs.Fields.Count-1 do
      begin
        sCreateTable := sCreateTable + #13#10'  `' +
                        rs.Fields[i].UTF8Name + '`: ' + rs.Fields[i].getTypeAsUTF8String() +
IIF(i<rs.Fields.Count-1, ',');
        sInsert := sInsert + '`' + rs.Fields[i].UTF8Name + '`' + IIF(i<rs.Fields.Count-1, ',');
        if rs.Fields[i].PrimaryKey then
          sPk := sPk + IIF(sPk<>'', ' ,') + '`' + rs.Fields[i].UTF8Name + '`';
        if rs.Fields[i].Index then
```

```
        sIndex := sIndex + IIF(sIndex<>'', ' ,') + '`' + rs.Fields[i].UTF8Name + '`';
      if rs.Fields[i].ForeignKey<>'' then
        sFk := sFk + IIF(sFk<>'', ', ') + 'FOREIGN KEY (`' + rs.Fields[i].UTF8Name + '`) ' +
rs.Fields[i].FForeignKey;
    end;
    if sPk <> '' then
      sCreateTable := sCreateTable + ','#13#10'  PRIMARY KEY (' + sPk + ')';
    if sFk <> '' then
      sCreateTable := sCreateTable + ','#13#10 + '  ' + sFk;
    sCreateTable := sCreateTable + ');' + #13#10#13#10;
    sInsert := 'INSERT INTO `' + cn.Tables[o] + '` (' + sInsert + ')';
    WriteUTF8ToStream(f, sCreateTable);
    while not rs.EOF do
    begin
      s := '';
      for i := 0 to rs.Fields.Count-1 do
        s := s + IIF(s<>'', ', ') + GetAsUTF8String(rs.Fields[i].Value, rtDump);
      WriteUTF8ToStream(f, sInsert + ' VALUES (' + s + ');'#13#10);
      rs.MoveNext;
    end;
    rs.Close;
    WriteUTF8ToStream(f, #13#10);
  end;
  rs.Free;
  f.Free;
  cn.Close;
 end;
 cn.Free;
end;

procedure CreateXMLExport(const Filename: string);
var
 cn: TADOConnection;
 rs: TADORecordset;
 sPk, sIndex, sFk: UTF8String;
 f: TFileStream;
 i, o: integer;
begin
 cn := TADOConnection.Create(nil);
 if cn.Open('Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' + Filename) then
 begin
   rs := TADORecordset.Create(cn);
   for o:=0 to cn.Tables.Count-1 do
   begin
     rs.Open(cn.Tables[o], adOpenForwardOnly, adLockReadOnly, adCmdTable);
     f := TFileStream.Create(cn.Tables[o] + '.xml', fmOpenWrite or fmCreate);
     WriteUTF8ToStream(f, '<?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 +
          '    <fields>'#13#10);

     sPk := ''; sFk := ''; sIndex := '';
     for i := 0 to rs.Fields.Count-1 do
     begin
       WriteUTF8ToStream(f, '      <field attrname="' + rs.Fields[i].UTF8Name + '" ' +
rs.Fields[i].getTypeAsUTF8String(rtXml) + ' />'#13#10);
       if rs.Fields[i].PrimaryKey then
         sPk := sPk + IIF(sPk<>'', ', ') + '`' + rs.Fields[i].UTF8Name + '`';
       if rs.Fields[i].FForeignKey<>'' then
         sFk := sFk + IIF(sFk<>'', ', ') + '`' + rs.Fields[i].UTF8Name + '` ' +
rs.Fields[i].FForeignKey;
       if rs.Fields[i].Index then
         sIndex := sIndex + IIF(sIndex<>'', ', ') + '`' + rs.Fields[i].UTF8Name + '`';
     end;
     WriteUTF8ToStream(f, '    </fields>'#13#10 +
       '    <params primary_key="' + sPk + '" lcid="' + rs.Fields[0].lcid + '" index="' + sIndex +
'" foreign_key="' + sFk + '" />'#13#10 +
       '  </metadata>'#13#10);

     if not rs.EOF then
     begin
       WriteUTF8ToStream(f, '  <rowdata>'#13#10);
       while not rs.EOF do
```

```
        begin
          WriteUTF8ToStream(f, '    <row>'#13#10);
          for i := 0 to rs.Fields.Count-1 do
            WriteUTF8ToStream(f, '      <col name="' + rs.Fields[i].UTF8Name + '">' +
rs.Fields[i].GetValueAsUTF8String(rtXml) + '</col>'#13#10);
          WriteUTF8ToStream(f, '    </row>'#13#10);
          rs.MoveNext;
        end;
        WriteUTF8ToStream(f, '  </rowdata>'#13#10);
      end;
      WriteUTF8ToStream(f, '</datapacket>');
      rs.Close;
      f.Free;
    end;
    rs.Free;
    cn.Close;
  end;
  cn.Free;
end;

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

Autor: Udo Schmal (mailto:udo.schmal@t-online.de) , veröffentlicht: 28.12.2013, letzte Änderung: 04.06.2021