SQLite Character Encodings - Converting from ANSI to UTF8

Written by BitBumper Ingenieurbüro, Heidelberg. 4/4/2016. This Text is Public Domain.

Introduction

SQLite is a pretty good database. It does what you want - which is not always given with other databases.

Unfortunately what you want is not always that what you really do. This is quite often found with the character encoding within the database. SQLite is quite simple: It works in UTF8 char mode (as long as you do not explicit switch to UTF16). What the database never does is store strings in native ANSI mode. This is quite good - because it is definitly not up do date.

The problem of some programmers is that they just stored their data as ANSI in a UTF8 database. As long as they are in their own world this is not a problem. The problem occur, if you either want to use the database with other programs which expect the data in the correct UTF8 encoding or if you want to port your application and the new database driver only can handle the data as UTF8.

This situation occurs quite often within the Embarcadero RAD Studio (Delphi/C++) environment. Versions below Delphi 2009 handled string by default as ansistring. Since Version 2009 everything is handled by default as unicode string.

If you port your application to a new version you recognize, that your data are not those you expect.

Detect the problem

How can you find out in what format your data are stored?

You only see this if you have some chars in your database, which are different encoded in ANSI and UTF8. All characters with bytecode 0..127 are just the same. If you have no other chars in your database you definitly have not problem: There is no difference with the encoding - so you have nothing to do.

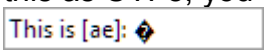
All Chars between code 128..255 are coded as 2-4 byte code in UTF8.

The ANSI encoded data between 128..255 are encoded according to the codepage you use – often it uses "ISO Latin 1 (ISO 8859-1)" standard.

Sample:

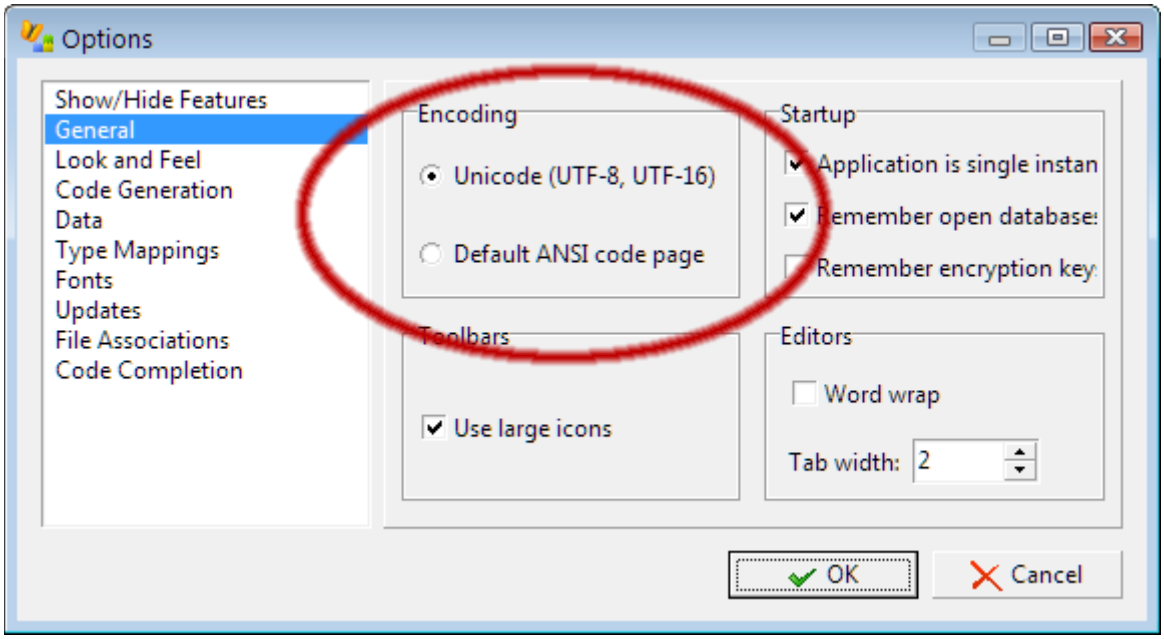
This is what we want to see: This is [ae]: ä

If you have a UTF8 encoded char in your database and you access this as ANSI, you will see 2 ANSI byte. The sample above will be shown as This is [ae]: Ã¤ .

If you have a ANSI encoded char above charcode 128 in your database and you access this as UTF8, you will see a <?> by most applications due it is a non valid UTF8 char: This is [ae]: 

For details about this see some tables of some encodings: http://www.utf8-zeichentabelle.de/ and the Wikipedia UTF8 article.

A quite good program to detect your encoding and simple switch the view to your data is Sqlite Expert. In the options of the program you can select in which encoding you want your data to be displayed.



Convert the data

To convert the data from the "wrong" ANSI format to the native Sqlite3 format UTF8 you have different options. But: Some of them are may not simple – and some drivers just refuse e.g. to encode the data in a different encoding than the database want.

We had problems to use newer Zeos versions or the FireDAC (formerly AnyDAC) Database drivers cause they always assume that you store UTF8 encoded strings and you have to use binary access or strange tricks to do the work there.

This method we show here works finally and is also very simple to use and to understand.

We use Delphi 2007 (which work by default with non unicode strings) and the Zeos Library in Version V7.0.0 (SVN revision 940). The Zeos library is freeware and can be used for all delphi versions as well as for Lazarus.

We also want that our application, which used ANSI encoding so far, convert the data after the startup and work from now on with the UTF8 encoding.

To detect, if we have already decoded we save a flag in the database with the information if we are already in UTF8 mode.

```
procedure TdmBase.ConvertUtf8P;
// convert all defined fields from ansi to utf8
var
  iIdx: integer;
begin
  // detect if we need to convert
  if not f_InternalCheckOkF(cdUtf8) or (i_InternalGetF(cdUtf8) = 0) then begin
    // convert all tables, which may contain ansi>128 chars
```

```
    for iIdx:= 0 to Length(axUtfFields) - 1 do begin
      if not axUtfFields[iIdx].fDisplayOnly then begin
        iUtf8CurrentIdx := iIdx;
        // convert this table
        TfmDlgWait.i_ShowModalWaitF(@fmDlgWait, self, OnUtfConvertInitP,
OnUtfConvertContinueF, OnUtfConvertDoP);
        axUtfFields[iUtf8CurrentIdx].xDs.Close;
      end;
    end;
  end;
  // remember, that we have converted
  InternalSetIntegerP(cdUtf8, 1);
end;
```

Above we use some callback functions to do the conversion:

```
procedure TdmBase.OnUtfConvertInitP(Sender: TObject);
// callback from wait window: start conversion
begin
  axUtfFields[iUtf8CurrentIdx].xDs.Open;
  axUtfFields[iUtf8CurrentIdx].xDs.First;
  fmDlgWait.btnCancel.Enabled := false;
  fmDlgWait.Width := 400;
  fmDlgWait.Left := (Screen.Width - fmDlgWait.Width) div 2;
  fmDlgWait.xlbCaption.Caption := 'Converting... Step
['+IntToStr(iUtf8CurrentIdx)+'].';
  fmDlgWait.xlbStep.Caption := 'Tabelle
'+S_ComponentNameWithoutPrefixF(axUtfFields[iUtf8CurrentIdx].xDs)+'... Please
wait.';
  fmDlgWait.pbr.Max := Max(100,axUtfFields[iUtf8CurrentIdx].xDs.RecordCount);
end;

function TdmBase.OnUtfConvertContinueF(Sender: TObject; i_Param: integer):
boolean;
// callback from wait window: detect end of table
begin
  result:= not axUtfFields[iUtf8CurrentIdx].xDs.Eof;
end;

procedure TdmBase.OnUtfConvertDoP(Sender: TObject);
// callback from wait window: do the conversion for the current table record
var
  fPost: boolean;
  sPre: AnsiString;
  sUtf: UTF8String;
  iFld: integer;
  xFld: TField;
  xDs: TDataSet;
begin
  xDs:= axUtfFields[iUtf8CurrentIdx].xDs;
  fPost := false;
  try
    for iFld:= 0 to length(axUtfFields[iUtf8CurrentIdx].asFields) - 1 do begin
      xFld:= xDs.FieldByName(axUtfFields[iUtf8CurrentIdx].asFields[iFld]);
      // read the string as ansi string from the database
      sPre := xFld.AsString;
      // convert this string to utf8
      sUtf:= AnsiToUtf8(sPre);
      // save (just if changed)
      if (sPre <> sUtf) then begin
        if not fPost then begin
          // on first difference: we need to post this record
          xDs.Edit;
          fPost := true;
```

```
        end;
        xFld.AsString:= sUtf;
      end;
    end;
    if fPost then begin
      xDs.Post;
    end;
  except
    xDs.Cancel;
    raise;
  end;
  xDs.Next;
end;
```

After we converted the database, we have to ensure, that we access this fields from now as utf8. In Delphi this is easy: Every Field hat a OnGetText and OnSetText event, where we can do the correct encoding/decoding.
We implement 2 general eventhandlers:

```
procedure TdmBase.OnGetTextUtf8(Sender: TField; var Text: string; DisplayText:
Boolean);
// do the utf8 decoding prior using the field internal as ansi string
begin
  inherited;
  Text:= Utf8ToAnsi(Sender.AsString);
end;

procedure TdmBase.OnSetTextUtf8(Sender: TField; const Text: string);
// do utf8 encoding before write to database
begin
  inherited;
  Sender.AsString:= AnsiToUtf8(Text);
end;
```

Those routines does the assignments of the field eventhandlers and
call those for every field need to be handled.

```
procedure TdmBase.Utf8FieldAssignP(xFld: TField);
begin
  xFld.OnGetText:= OnGetTextUtf8;
  xFld.OnSetText:= OnSetTextUtf8;
end;

procedure TdmBase.Utf8InitP;
// init all utf8 fields: assign to GetText/SetText
  procedure IntInitP(xDs: TDataSet; asFields: array of string);
  var
    iFld: integer;
    xFld: TField;
  begin
    for iFld:= 0 to length(asFields) - 1 do begin
      xFld:= xDs.FieldByName(asFields[iFld]);
      // avoid that we have a static eventhandlers for your events already
      AssertP(not assigned(xFld.OnSetText) and not assigned(xFld.OnGetText),
        Format('Utf8 init fail for [%s]', [xDs.Name]));
      Utf8FieldAssignP(xFld);
    end;
  end;
var
  iIdx: integer;
begin
  if f_InternalCheckOkF(cdUtf8) and (i_InternalGetF(cdUtf8) <> 0) then begin
```

```
    for iIdx:= 0 to Length(axUtfFields) - 1 do begin
      IntInitP(axUtfFields[iIdx].xDs, axUtfFields[iIdx].asFields);
    end;
  end;
end;
```