Audit Trails Or How To Display Arabic Text in Blaise

Leif Bochis
Statistics Denmark

Abstract

Audit trails is a feature in Blaise 4 which is very useful for - as stated in the Developers's Guide - methodological research of questionnaires, analysis of interviewers' use of the system, debugging of Blaise code and backup/recovery.

The core of the audit trail mechanism is a sort of 'event trigger' where each kind of event causes a call to an external procedure defined in a DLL. These DLL procedures may be programmed to do exactly what may be the actual need for such an audit trail.

Because the audit trail mechanism is implemented in such a general way it allows the programmer to exploit the information from this event trigger in various ways and for various purposes.

This paper describes an example of passing information from the audit trail event trigger to an other program which in turn displays supplementary information in a separate window - in this example field texts and answer texts in Persian for a multi language survey.

Immigrant Surveys in Statistics Denmark

From autumn 1998 to summer 1999 Statistics Denmark carried out series of interviews on a number of studies of immigrants in Denmark.

The largest of the surveys was carried out in a number of steps:

1. An initial pilot study in CATI, in order to test and refine the questionnaire - carried out in October 1998.
2. A CATI survey carried out from the end of November 1998 until June 1999.
3. Supplementary CAPI interviewing in order to raise the response rate for groups difficult to catch by telephone - carried out April to June 1999.

The respondents were a sample from major immigrant nationalities to Denmark from outside Scandinavia, i.e. immigrants and descendants of immigrants from among other countries Lebanon, Iran and Vietnam.

A Blaise III solution

Some of the important questions concerned language skills of the respondents and it was decided that the respondents should be interviewed in Danish if possible, otherwise in their own language if applicable, or in English as a third alternative.

In other words, the interviewers should be able to change interview language at contact, which led to the decision to use Blaise III as the interviewing tool, because of its capability to handle a number of interviewing languages that could be changed on the fly. At the time Blaise 2.5 was the tool used by the interviewing section of Statistics Denmark, but as we had to hire interviewers capable of interviewing in

Danish as well as the relevant languages for this survey, we could start this survey just training these specific interviewers in Blaise III usage.

Blaise 4 was too young to be a realistic alternative at that time.

Because of very short time to develop the instruments, we had no time to prepare our system to use the relevant Dos codepages for the selected language, but by removing some diacritical symbols from the translated texts it was possible to include for example the languages Polish, Serbo Croatian and Somali in the list of languages available on the screen. Arabic, Farsi and Vietnamese, however, had to be read up by the CATI-interviewers from a print-out next to the screen. It was a somewhat awkward solution, but the - quite few - interviewers soon learned to translate on the fly, so the problems with this solution almost disappeared in a couple of weeks.

A possible Blaise 4 solution

At the Blaise Conference in Lillehammer (November 1998 - between the Pilot Study and Real Interviewing), I got a couple of ideas of how to do this in a more professional manner using Blaise 4 Windows. There was too short time after the conference to get a solution ready for the CATI Survey (and to get assured that Blaise 4 was mature), but still some time to get it ready for the scheduled CAPI Survey.

The main goals were :
- get the translated texts in Arabic, Farsi and Vietnamese in a machine-readable form
- get a way to display it on the screen
- get a way to make Blaise display it on the screen

Choosing Arabic Font

Arabic and Farsi are written with the Arabic alphabet - a number of fonts are available on the Internet supporting the number of codepages defined for the Arabic alphabet. The choice was a proportional Windows TrueType Font, which could be used for Arabic as well as Farsi. The chosen font also comprised the Latin alphabet (letters from A to Z).

As some of the questions referred to specific terms it was an advantage that these Danish terms could be displayed as part of the question text.

The disadvantage of the chosen font was the missing of certain characters - such as special hamza combinations.

Choosing efforts

We chose to concentrate on the Farsi questionnaire by several practical reasons.

The Arabic text was not available in a single machine-readable form, so this was dropped.

The vietnamese text was available as a Word document, but after a couple of tries, I gave up converting it to a simple text file. Besides, the Vietnamese respondents were generally well-integrated, so only few of them really needed to be interviewed in Vietnamese.

The Farsi text was available in a format which was possible to get hold of and a quite simple Pascal program could scan the source and convert the contents into a simple text file format that matched the chosen font.

Demands to the conversion program

The conversion program dealt with

- how to manage switching between left-to-right and right-to-left displaying (in some of the field texts or answer texts short explanations in Danish were inserted in the Farsi text)
- the four Arabic presentation forms (the characters are different in shape decided by the position in the word : In the beginning of the word, in the end, between other characters or alone - in Latin alphabets you can compare these presentation forms with upper case and lower case)
- how to convert from an Arabic codepage in the original file into the codepage of the chosen font

With this approach it was possible to get a print-out from the Notepad text editor using the Persian Font, which looked pretty much the same as the print-out from the Farsi word processor the translator used.

Making Blaise 4 Display Arabic Letters

In Blaise 4 it is possible to define alternative fonts to display field texts and answer texts - however: Blaise 4 source files were compatible with Blaise III source files, so Blaise 4 source files are converted to OEM when saved and converted to ANSI when read - I didn't dare to think of how to manage these conversions of text files when applied on characters not defined in the two relevant character sets - and I didn't even dream of how eventually to edit these characters!

So the solution was to use the Audit trail event trigger ...

What is the Blaise 4 Audit Trail ?

A generalized event trigger that leaves the actual event-caused actions to the user-defined functions assigned to the events.

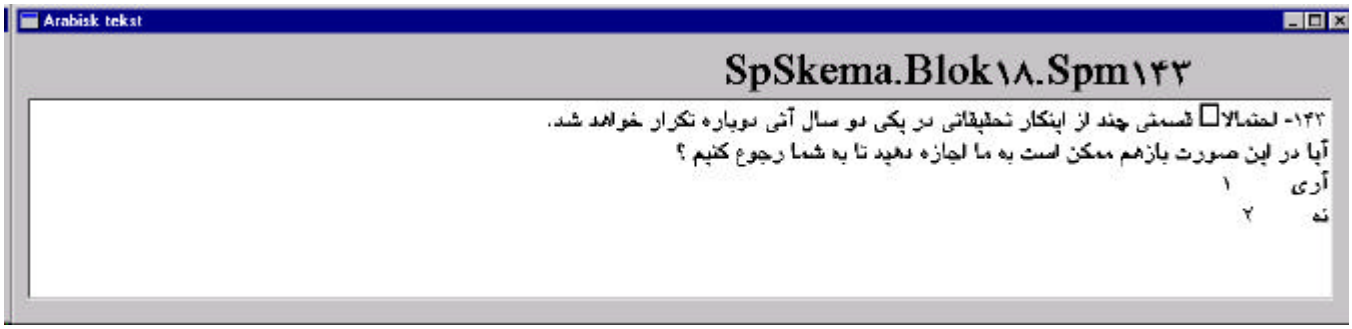
With this approach it is possible to use Delphi to design a supplementary window with a text display control. The proper events triggered causes some text to be displayed through this control.

In this study only three of the defined controls were relevant:

- AuditTrailInitialization, used to initialize the system, i.e. read the Farsi text file into the arrays.
- AuditTrailEnterField, used to display the Farsi text relevant to the proper field.
- AuditTrailFinalization, used to close down the system.

Implementation of a Supplementary Window

The supplementary window should contain a few controls where the most important should be able to display Farsi text, line by line, aligned to the right, using the chosen Persian Font.



We decided to use the built-in standard Rich Text Control. This Rich Text Control is able to display text with a chosen font, alignment, size etc. The advantage of this choice was that we didn't need to develop a tailored text display window so the actual text display control was developed in a very short time.

A text file containing the Farsi text

A few conventions for the text file was decided:

1. Every new field text was identified by a line starting with the characters ###.
2. The characters ### is followed by the full name of the field as it is delivered from the Audit Trail Event trigger *AuditEnterField.FieldName* property.
3. Following a number of lines including the Farsi text, until
4. A new line starting with ### denotes the beginning of a new field.

See example in Appendix I.

The lines of Farsi text was produced by the conversion program automatically in such a way that a text was split after 80 characters (the first space met after character no. 80 produced a line split). It was done this way in lack of a Windows control that was able to display left-directional text and manage line shifts by itself. With a maximum of slightly more than 80 characters per line the Farsi text almost filled the screen width at a 800x600 resolution, and therefore the text could be displayed quite properly by the standard Rich Text Control, using the Persian font and right-aligned.

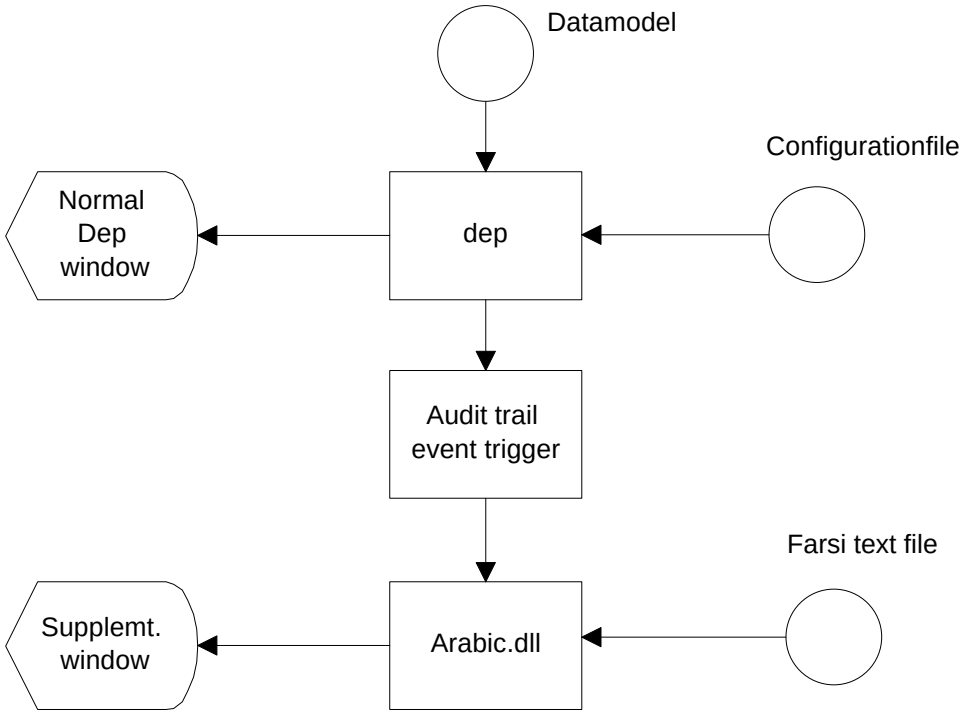
A list of field names were produced by a Cameleon script and were merged into the field texts manually. After this followed some minor editing of the automatically produced field texts.

At the initialization of the 'Audit trail' - *AuditTrailInitialization* procedure - the text file was read into memory in two arrays: One array holding the names of the fields, and the other holding the text lines (in the earlier mentioned built-in Delphi-structure).

When the instrument enters a new field the *AuditTrailEnterField* procedure triggers the name of the field, searches for it in the field names array and - if found - displays the corresponding Farsi text in the Rich Text Control in the Supplementary Window.

The data structure could be more refined, but as it worked very fast in the test, there wasn't really a need for it in this study.

Overview



The Blaise instrument

The instrument was prepared using Blaise III, and while Blaise III and Blaise 4 was able of sharing the datafiles, we left the interviewers the choice of selecting the preferred tool - actually it was a standard installation, so every interviewer had the choice to select either Blaise III or Blaise 4 version.

The three Farsi-speaking interviewers, however, were already used to translation on the fly, so they didn't really need the solution - and in practical interviewing they just carried out using the tool they were used to.

Conclusions

Though the system was not used in practical interviewing, the study showed that it was possible to exploit the Audit Trail mechanism for these purposes, and to make a workable solution that way.

Lot of things could be refined, the DLL could easily be made more general in order to dynamically select a language, a font and a text file - either on request by the user or for the particular user select the proper values from the Registry or Environment variables.

Sparse tries in Blaise 4.3 shows that when there is no OEM-ANSI conversion involved it is possible to fill the text right into the Datamodel source the same way as the other languages, so if the proper text can be represented by a suited font, there is no need to keep it and edit it separate. However, the font selection capabilities of Blaise 4.3 still lacks the property *Alignment.*

Though we probably won't need this way to display Arabic text again, the study was good to get an overview of the large potentials of the Blaise 4 Audit Trail mechanism - and to learn the characteristics of the Arabic Alphabet.

References

Blaise 4.1 Developers' Guide, Chapter 5.6 Audit Trail, pp.282-293

APPENDIX I : Farsi.txt-sample:

```
###SpSkema.Blok01.Spm01i
ö‘ðȼ÷û‘÷• øöþ¤›‘úõ ) ö‘þ¤‘¡ á‘®ø¥ • ë þ Ÿ— øü¨¤¤“ ñ‘Ÿ¤¢î¤‘õ÷• ¢¤‘õ o¤• ¢•
ñ‘¨¤• ö‘— þ¤“ « ø¬¡ öþ¤ ¢ý• ùõ‘÷ üð¥ ‘—“ ‘õ .¢ª ‘“þõ î¤‘õ÷• ¢¤¢‘ú÷ ü ðȼ÷¥ oøŸ÷ ø(
‘õª «¡ª ‘“ •¤ ù“Ÿ¬õ öþ¤ ‘— ô þóþõ ‘õ •¤ ùî –¨• ù—ê¤ ù¤‘ª • ö ¤ ¢ùî ôþ¤ ù—ª • ¢
.Ȼõ÷ üõ öþõ®— •¤ ‘ õª ö¢÷‘õ §‘÷ª ‘÷ Š ü—‘ì þ Ÿ— ¤‘î øü¨¤¤“ öþ¤ ù÷øð øôþª ‘“ ù—ª • ¢

ü—‘õȼ õ –ò• ¸Þ : • – õ¨ì
¥ ‘è Ȼþ ü õ ö ¥ • ùî ý¤øª î øö‘“¥ o¤‘“ ¤¢ñ• ¸Þ¢÷• ‘“ •¤ ù“Ÿ¬õ ö‘— ù¥ ‘›• ‘“ ö õ
.ô÷îþõ
###SpSkema.Blok01.Spm01
ŒȾ¤þ£ • –¤ø¬ ‘õª ‘“ ù“Ÿ¬õ öþ¤ ùî Ȼóþõ ü÷‘“¥ ù ù“ -•
•  ü“¤ä
,  üîî¤‘õ÷• ¢
ƒ  ü¨þóð÷•
„  ü¨¤‘ê
…  ü÷‘—¨úó
†  ü—• ø¤î Š ü“¤¬
‡  üŽ‘ þó‘õø¨
ˆ  üî¤—
‰  ü“‘›÷• /øȼ•
•  üõ‘÷—þø
###SpSkema.Blok01.Spm05
ü“û£õ ‘þøü÷‘“¥ à‘Ÿõ ¥ • ù ) ü õøì –þóì • í þä¥ › ö‘— öúþõ ¤¢‘õª oȼ• ø÷‘¡ / ‘õª ‘þ -…
ŒȾþ—ê¤ üõ ¤‘õª ù“ (
•  ý¤
,  ù÷
###SpSkema.Blok01.Spm06
.....Ȼþ‘õ¤ê ¤î£ ‘ê¯ó Œ–þóì • ô• Ȼ ù“ ý¤ ¤ ð• -†
```

APPENDIX II : Delphi program

```
library Arabic;

uses
  SysUtils,
  DepAudit in 'DepAudit.pas',
  Forms,
  Unit1 in 'Unit1.pas'{Form1};

procedure AuditTrailInitialization(const AuditInitialization: TAuditInitialization); export; stdcall;
begin
      Application.CreateForm(TForm1, Form1);
      Form1.Show;
end;

procedure AuditTrailFinalization(const AuditFinalization: TAuditFinalization); export; stdcall;
begin
      Form1.RichEdit1.Clear;  Form1.RichEdit1.Free;  Form1.Close;
end;

procedure AuditTrailEnterField(const AuditEnterField: TAuditEnterField); export; stdcall;
var s: String;
begin
      Form1.FormUpdate(AuditEnterField.FieldName);
end;

{*** dummy procedures, defined for compatibility reasons only ***}
procedure AuditTrailLeaveField(const AuditLeaveField: TAuditLeaveField); export; stdcall;begin end;
procedure AuditTrailAction(const AuditAction: TAuditAction); export; stdcall;  begin end;
procedure AuditTrailEnterForm(const AuditEnterForm: TAuditEnterForm); export;stdcall;begin end;
procedure AuditTrailLeaveForm(const AuditLeaveForm: TAuditLeaveForm); export;stdcall;begin end;

exports
      AuditTrailInitialization     index 1,
      AuditTrailFinalization       index 2,
      AuditTrailLeaveField         index 3,
      AuditTrailEnterField         index 4,
      AuditTrailAction             index 5,
      AuditTrailEnterForm          index 6,
      AuditTrailLeaveForm          index 7;
begin
end.

unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, Menus;
```

```
type
  TForm1 = class(TForm)
    RichEdit1: TRichEdit;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure FormUpdate(SpmNavn : PChar);
  end;

const
  MaxEntries = 200;
  MaxTxtLng = 2000;
var
  Form1: TForm1;
  F : Text;
  FeltNavn : array [1..MaxEntries] of PChar;
  FeltIndh : array [1..MaxEntries] of PChar;
  NofEntries : Integer;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
   i : integer;
   S : String;
   P1 : PChar;
   atmp : array [0..MaxTxtLng] of char;
   atmp2 : array [0..100] of char;

   function Kopier (st : String) : PChar;
   var
     A : array [0..MaxTxtLng] of char;
     P : PChar;
   begin
      A:='';
      P := StrPCopy(A, St);
      Result := StrNew(P);
   end;

begin
   Label1.Caption := 'Spørgsmålsnavn';
   AssignFile(F, 'farsi.txt');       {* This ought to be defined somewhere else, but never mind now! *}
   reset(F);
   i:=0;
   NofEntries:=0;
```

```
  while not eof(F) do
  begin
    readln(F, S);
    if Copy(S, 1, 3) = '###' then
     begin
       if i>0 then FeltIndh[i] := StrNew(atmp);
       i:=i+1;
       FeltNavn[i] := Kopier( Copy(S,4,Length(S)) );
       atmp :='';
       FeltIndh[i] := atmp;
     end
    else
    begin
      P1:= StrPCopy(atmp2, S + Chr($0D) + Chr($0A) );
      FeltIndh[i] := StrCat(atmp, P1);
    end;
  end;
  FeltIndh[i] := StrNew(atmp);
  NofEntries:=i;
  CloseFile(F);
end;

function FindIndhold (FieldName : PChar): PChar;
var
  i : integer;
  S : PChar;
begin
  S:='';  i:=0;
  repeat
    i:=i+1;
    if StrComp(FeltNavn[i], FieldName) = 0 then S := FeltIndh[i];
  until (i >= NofEntries) or ( StrLen(S) > 0 );
  if StrLen(S) = 0 then FindIndhold := 'Kun dansk tekst til dette spoergsmaal!'
                   else FindIndhold := S;
end;

procedure TForm1.FormUpdate(SpmNavn : PChar);
var Navn : String;
   i : integer;
   P : PChar;
begin
  Navn:='';
  for i:=0 to StrLen(SpmNavn)-1 do
    if SpmNavn[i] in ['0'..'9']
    then
      Navn:=Navn + Chr(Ord(SpmNavn[i]) + $50)
    else
      Navn:=Navn + SpmNavn[i];

  Label1.Caption := Navn;
```

```
  with RichEdit1 do
  begin
     P := FindIndhold(SpmNavn);
     Text := P;
  end;
end;

end.
```