Different Methods of Working with Blaise® in the

Israel Central Bureau of Statistics

Shifra Har and Evgenia Luskin, Israel Central Bureau of Statistics

1. Introduction

The Israel Central Bureau of Statistics (CBS) is using the Blaise software for more than 10 Years. This software is used in the development of many surveys – especially in the arena of families' surveys and individuals' surveys. More than 20 systems have been developed using the Blaise software. In particular I'd like to mention that Blaise was used to develop the CBS population and housing census questionnaire that was conducted in 2008.

Some additional surveys that use Blaise include the labor force survey, the family income survey, the social survey, the health survey, the first degree graduates survey, the crime survey and more.

At first we have used the Blaise tool for questionnaire development and Manipula for the survey management systems. We overcame the problems we had with the Hebrew language, which is written from right to left, in developing the surveys. During time we have also found solutions to the problems of keying-in two additional languages: Arabic and Russian. Today the questionnaires for all of our surveys are allowing enumeration in these three different languages.

As time went by the developers work procedures have changed, new tools, software development languages and databases were introduced – so we combined the use of several tools to get the maximum benefit from each tool. Despite all of the above, we still keep the Blaise as the main tool for developing surveys.

Massive work with the Blaise software gave us the ability to gain huge experience in using the different Blaise tool components. With time, the computerized Blaise questionnaires became more complex and combined different modes of data collection: CAPI (Computer-assisted personal interviewing), CATI (Computer-assisted telephone interviewing) and CADI (Computer-assisted data input). These kinds of complex surveys demanded more complex management systems as well.

We are about to start the development of a system for generic management of surveys that include different modes of data collection. The CBS decided to use Blaise for developing the questionnaires for all of the enumeration techniques for this generic survey management system.

The CBS puts a main emphasis on information security. One of the issues we are handling is the writing of secure code. The development methodology of the CBS demands that we put special attention to the fact that the enumerators are sending us information from all over the state of Israel through the internet network which is an unsecured network. This means that we must use encrypted data to transfer the survey information. It is not enough to develop safe communication; additional steps are taken in each individual computer so that the software code and the data itself are secure. "Information noise" is entered into the data, so if anyone is listening they will not be able to decipher anything.

We can distinguish between several methods of working with the Blaise tool when developing computerized surveys:

1.1. The "*Blaise + Manipula*" method

The first method is using the Blaise and Manipula in a natural way. The questionnaire is written using Blaise, the management system is written using Manipula and the data is stored in the Blaise database.

1.2. The "*Blaise + .Net*" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, the questionnaire data is stored in the Blaise database and the survey management data is stored in an SQL server database.

1.3. The "*.Net + SQL*" method

In the last method the questionnaire is written using Blaise, the management system is written using .Net, both the questionnaire data and the survey management data are stored in the an SQL server database.

Each of the above three methods has its own advantages and disadvantages. Before each new survey there is an examination of which method is the most suitable for that survey.

2. The different methods of working with Blaise

In the following sections I will go into detail and give examples for surveys that were done in each one of the three methods. I will note the advantages and disadvantages of each method as well.

In general, the CBS has been conducting surveys in all three methods, but in recent years the direction is to develop more systems with .Net (the last two methods) and less with Blaise only (the first method).

2.1. The "*Blaise + Manipula*" method

Historically, this is the first method that was used. The CBS labor force survey, social survey and health survey were written 100% in the Blaise tool. The survey management system in the survey gathering center and in the laptops was written in Manipula.

<u>Advantages of the *"Blaise + Manipula"* method:</u>

The Blaise language enables quick development which helped us move quickly from paper surveying to computerized surveying. The systems have proven their stability and efficiency.

<u>Disadvantages of the *"Blaise + Manipula"* method</u>:

Manipula is limited as a development tool. The limitations are especially noticed in the screen designs, batch activities in the management system and multi users' capabilities.

We still use the *"Blaise + Manipula"* method for relatively small and simple surveys. This means surveys that do not use the CAPI mode of data collection and there is one coordinator that controls the data gathering.

Examples for current usage of this method are the first degree graduates survey and the public transport satisfaction survey. The first labor force survey that was developed in this method has already been replaced with a more advanced system.

In the social survey we still use the old management system but the new management system is under development and will be in production in 2010.

2.2. The "*Blaise + .Net*" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, the questionnaire data is stored in the Blaise database and the survey management data is stored in an SQL server database.

Starting from Blaise version 4.6 and with the help of the module BCP 2 we are combining Blaise and .Net in two ways; from Data Entry Program (DEP) to .Net and from .Net to DEP.

Examples for usage of this method:

2.2.1. From DEP to DLL that is written in .Net

In the questionnaire of the social survey the finding and marking of streets is done with DLL. In the questionnaire of the health survey the DLL checks the validity of key fields and inserts data into the questionnaire from an SQL management table.

In figure #1 in the appendix we can see two procedures:

- The procedure GetMezahe returns the output from the validity checks in the parameter p_TozaatPkida
- The procedure GetSqlData brings data from the SQL database to the export parameter list.

2.2.2. From .Net to DEP

In a management system that is written in .Net there are calls to DEP for surveying or questionnaire editing, for calls to Manipula programs and for different activities done with Blaise tables.

Advantages of the *"Blaise + .Net"* method:

Utilizing the strong abilities of the Blaise language which include quick development, ease of changing a questionnaire and stability. Using this with .Net gives us a solution for the screen design problems and gives us a wide variety of development options. Performance of activities with SQL tables is much faster than with Blaise tables so the management system is much more friendly and quick.

Disadvantages of the *"Blaise + .Net"* method:

The performance of activities using Blaise tables is relatively slow compared to SQL tables.

Examples for current usage of this method are the labor force survey which is already in use and the social survey which is still under development.

2.3. The "*.Net + SQL*" method

In this method the questionnaire is written using Blaise, the management system is written using .Net, both the questionnaire data and the survey management data are stored in the an SQL server database. During the survey only one questionnaire exists in the Blaise questionnaire table. After exiting from the questionnaire all of the data is transferred to an SQL table and deleted from the Blaise table.

Actually we are using Blaise as the GUI for the surveying screens. In this method we need to develop two modules. The first is for storing the Blaise data into SQL and the second is for uploading the Blaise questionnaire from data stored in the SQL. Both modules are written in .Net.

In figure #2 in the appendix we can see part of a management program that creates a Blaise table, fills it with data from an SQL table, entering DEP, exiting the questionnaire and saving the data in SQL and at the end deleting the Blaise table.

Advantages of the *".Net + SQL"* method:

Using this method gives us great performance compared to the first two methods. It utilizes the Blaise strong abilities as a language for questionnaire development, the ease of changing a questionnaire and the stability of the DEP software.

Disadvantages of the *".Net + SQL"* method

This method requires the creation of an SQL table for keeping the questionnaire data. Each change in the questionnaire like adding a question or changing the definition, requires a change in the SQL table as well.

The two additional modules that are required for transferring the data between the Blaise tables and the SQL tables are very large. There must be reference in these modules for each field and the structure of the fields should be identical. This requires a long development and debugging time.

The Blaise advantage of fast development is diminished with the need to update these additional modules with every change.

So the main disadvantages of this method are lack of flexibility and long development time.

A good example for the *".Net + SQL"* method can be found in the 2008 CBS population and housing census.

The methodology for the population and housing census survey is a new methodology called the integrated census. This census is based on an integrated use of data from administrative files, together with sample data gathered in surveys, i.e., in the census field work of 20% of the population. For the questioning of 20% of the population, which is about 320,000 households, we used laptops that contained the Blaise questionnaire.

The questionnaire held approximately 300 questions and was comprised of three parts: questions about the apartment, questions about the household and questions about the individual. The questionnaire format is similar to the labor force survey questionnaire.

Approximately 2500 enumerators and coordinators participated in this survey. Two management systems have been developed, one for the enumerator and one for the coordinator. Due to the amount of users and the survey requirements the main criteria for the system was good performance – that is why the development method that was chosen is the *".Net + SQL"* method.

Modules for data transfer between Blaise and SQL where written in C-Sharp. This enabled good performance in the laptop system and also in the management systems that were located in the regional offices and the national management system.

For the survey data two SQL tables were created; one for the survey answers and the second for different statuses: "response", "don't know" and "refuse". In figure #3 in the appendix we can see part of the software that was developed for the population and housing census in which the questionnaire is being filled from the SQL data and entered into DEP.

The objective of the developers of the census questionnaire was to create a surveying procedure for an enumerator that is not skilled. The enumerators training was very intensive but still many of the enumerators and coordinators didn't have experience working with laptops or didn't have surveying field experience before being recruited for the job.

This is why the questionnaire developers made the text of the questions very detailed with many comments for the enumerators. The enumerator had no freedom to make decisions based on experience; each question was very structured and written properly. The surveying was done in three languages; Hebrew, Arabic and Russian. Most of the text was not constant but was matched to second or third person according to the specific grammar of the chosen language. A strong emphasis was given to the logical checks in order to avoid wrong data entered into the questionnaire.

In this questionnaire of the population and housing census we made the first try of calling DLL from DEP for finding and marking of streets. From the perspective of development languages the flow was .Net -> Blaise -> .Net and this flow was not 100% stable. We got a few complaints from the field about a lost of focus in the street selection screen. There weren't too many such complaints, but we were not able to ignore this issue. The problem is that Blaise does not support work in a multi-threading environment. This is one of the lessons that we learned from the 2008 population and housing census.

The main disadvantages of the *".Net + SQL"* method which are the lack of flexibility and long development time were not significant in the 2008 population and housing census since we had enough time allocated for development after the final changes were done to the questionnaire.

We found two other disadvantages to this method:

- ❖ Comments on fields are not downloaded from Blaise to SQL – so they are not used
- ❖ The received signal status is not stored in the SQL tables. So after uploading the data from SQL to the Blaise tables for further handling, the user is expected to re-approve signals that were already approved

3. Generic Management of Surveys

The number of different surveys that are conducted in the Israel CBS in the last few years has greatly increased. They were done in different modes of data collection: mail, field, telephone or a combination of the above. Internet surveys were introduced as well. The management systems for these surveys were written in Blaise + Manipula or in several other development languages. The main difference between these systems was the personal viewpoint of the person/department ordering the survey and the available resources for the project.

The purpose of the generic system for managing surveys is to reduce the resources needed to establish a new survey.

A strategic decision that was made in the generic management system is to separate between the questionnaire and the management system. Each questionnaire should contain a standard section with management data.

A generic survey will include the following sections:

- ❖ Survey definition and creation of required components
- ❖ Management for each of the different modes of data collection and integration between them
- ❖ Management of internal and external users
- ❖ Interface to the questionnaire
- ❖ Control and management tables
- ❖ Information security
- ❖ Archive management

The generic survey management system will use the *"Blaise + .Net"* method.

4. Summary

The three methods of developing surveys with Blaise each have their own advantages and disadvantages. The best development method for a specific survey is chosen according to the requirements of that specific survey, the different modes of data collection and other special requests.

The Israel CBS sees the Blaise as our main tool for questionnaire development. The development of a generic survey management system will allow managing a survey with efficiency and user-friendliness. In addition the interface between the management system and the questionnaire will be standard. This will reduce time and resources needed to develop new surveys.

Appendix

Figure #1:

```
PROCEDURE GetMezahe
   parameters
     import
     p_Mezahe  :string
     p_Shana    :integer
     p_Reva     :integer
     p_Shavua   :integer
     p_Shlav    : T_Shlav

    export
    p_TotzaatPkida        :Integer

     Alien ('checkMezahe.GetMezahe','TotzaaBdika')
 ENDPROCEDURE
-------------------------------------------------------------
 PROCEDURE GetSqlData
   parameters
     import
     p_Mezahe  :string
     p_Shana    :integer
     p_Reva     :integer
     p_Shavua   :integer
     p_Shlav    :integer

    export
      p_TaarSiumPkida : string
    export
      p_KnisaLeSviva  : integer
    export
      p_StatusSheelon  : integer
    export
      p_Ktovet         : string
    export
      p_ShemYishuv     : string
    export
      p_Mishpacha      : string
    export
      p_SugMidgam      : integer
    export
      p_PkidaHachnasot : integer
    export
      p_Merchav        : integer
    export
      p_MispMishkeyBait : integer

   Alien ('checkMezahe.GetMezahe','FillFieldsFromSql')
 ENDPROCEDURE
```

Figure #2:

```
'Creating instance of central BLAISE DB for rakaz
Dim InpDb, OutDb As String
InpDb = ConfigurationSettings.AppSettings("MaslulBikoret")
OutDb = "C:\"
OutDb = OutDb & System.Environment.UserName & "\" & "ShBikoret"

objOutBikoret = objdatabaseManager.OpenDatabase("")
objOutBikoret.DictionaryFileName = InpDb & .bmi"
objOutBikoret.DataFileName = OutDb & ".bdb"

Opening Blaise Db
objOutBikoret.Connected = True

'*******************
'Filling Fields of Blaise Questionnaire from SQL Server DB
 MiluiSadotBeShelon(i, Par)
objOutBikoret.WriteRecord()

Closing Blaise Db
 objOutBikoret.Connected = False

'*******************
Opening the Blaise Questionnaire for user
fl.Copy(InpDb & ".bmi", outDb & ".bmi", True)
fl.Copy(InpDb & ".bdm", outDb & ".bdm", True)
mystr = MaslulDepExe & " " & outDb & " /G /X /K" & par
        id = Shell(mystr)
        Dim myproc As Process
        Dim ip As Boolean

'Checking whether the user got out of Blaise Questionnaire
        myproc = System.Diagnostics.Process.GetProcessById(id)
        ip = myproc.HasExited()
        Do While ip = False 'Blaise is open
            ip = myproc.HasExited()
        Loop
        Return id

'*******************
'Saving changes from Blaise Questionnaire in SQL Server DB
      objdatabaseManager = New BlAPI4A2.DatabaseManager
      objOutBikoret = objdatabaseManager.OpenDatabase("")
      objOutBikoret.DictionaryFileName = InpDb & ".bmi"
      objOutBikoret.DataFileName = OutDb & ".bdb"
      objOutBikoret.Connected = True

     If KeyValueExists(objOutBikoret, Par) Then
         objOutBikoret.ReadRecord()
     End If
      GetNetunimFromShelon(i)
      SaveShelonInSql(i)
      objOutBikoret.Connected = False

'*******************
Deleting Blaise DB
If Not fl.Exists(OutDb & ".~lk") Then
                fl.Delete(OutDb & ".bfi")
                fl.Delete(OutDb & ".bjk")
            fl.Delete(OutDb & ".bpk")
            fl.Delete(OutDb & ".bsk")
            fl.Delete(OutDb & ".brd")
            fl.Delete(OutDb & ".bri")
            fl.Delete(OutDb & ".bdb")
            fl.Delete(OutDb & ".bmi")
            fl.Delete(OutDb & ".bdm")
End If
'*******************
Refreshing data set with updated data from SQL Server DB and showing it to the User
ds = exc.GetNetuneimFromBikoret(MyUser.Substring(5, 2), FlagSiyum, Shana_, Hodesh_)
dtReshima = ds.Tables(0)
grdReshLeBikoret.DataSource = dtReshima
grdReshLeBikoret.Refresh()
```

Figure #3:

```
private void SetStatus2BlField(FieldStatus dsStatus, ref Field blField)
{
            try
            {
            string sField4Status = blField.Parent.IndexedName + "." +   blField.LocalName;
            switch (dsStatus)
                        {
                        case FieldStatus.DoNotKnow:
                                    blField.Database.get_Field(sField4Status).Status =
                                    BlFieldStatus.blfsDontKnow;
                                    break;
                        case FieldStatus.Refusal:
                                    blField.Database.get_Field(sField4Status).Status =
                                                BlFieldStatus.blfsRefusal;
                                    break;
                        }
            }
                        catch(WriteToLogException ex)
            {
                        cWriteToLog.WriteToLogExceptionOnly(ex);
            }
                        catch(Exception ex)
            {
                        cWriteToLog.WriteToLog(ex,"");
            }
}
```

```
            int indexPerson = GetIndex(fld.Name);
            // new 2008
            if((fld.Name.IndexOf("Butal.Men")>-1 || fld.Name.IndexOf("Hativa01S")>-1 || fld.Name.IndexOf("Hativa01D")>-1) &&
                        indexPerson<=drAll.Length)
            {
                        pdRow=(Packet.PersonsDataRow)drAll[indexPerson-1];
            }
            if(fld.Name.IndexOf("Hativa01A2")==-1 && fld.Name.IndexOf("Hativa01A")>-1 &&
indexPerson<=drResidentsArray.Length)
            {
                        pdRow=(Packet.PersonsDataRow)drResidentsArray[indexPerson-1];
            }
            if(fld.Name.IndexOf("Hativa01A2")>-1 && indexPerson<=drNonResidentsArray.Length)
            {
```

```
		pdRow=(Packet.PersonsDataRow)drNonResidentsArray[indexPerson-1];
	}
	if(pdRow!=null && pdRow[strColumnNameInSql] != DBNull.Value)
	{
	if(fld.FieldDef.IsSet)
	{
	fld.TextAsSet = pdRow[strColumnNameInSql].
				ToString().Trim().Split(new  Char[]{'-'} );
		}
		else
		{
			fld.Text = pdRow[strColumnNameInSql].ToString().Trim();
		}
	}
	//set Status to Field(Blaise)
	if(pdRow!=null && dsHousehold.Tables[strTableNames[0]].Columns.Contains(strStatusFieldName))
	{
		if(pdRow[strStatusFieldName] != DBNull.Value)
		{
			string sStatus = pdRow[strStatusFieldName].ToString();
			int iStatus = Int32.Parse(sStatus);
			FieldStatus fs = (FieldStatus)iStatus;
			SetStatus2BlField(fs, ref fld);
		}
	}
```

```
	string param = pathBlaiseInfo+"SheelonU2006"+" /m"+pathBlaiseInfo+
		"SheelonU2006.bwm"+" /NOINITSCREEN"+" /g /x /l"+iLang.ToString()+" /k1"+
		" /e"+pathBlaiseInfo;
	if(kLastPressedKey==Keys.F9)
	{
		param=param+" /R";
	}
	broker.FromDsToBdbPath1_2(dsAllData,frCurrentFlat,pathBlaiseInfo+"SheelonU2006.bdb",sPrevDate,sUpdate);
	ProcessStartInfo psiB = new ProcessStartInfo(pathDep+"dep.exe",param);

	blaise= Process.Start(psiB);
	blaise.WaitForInputIdle();
	blaise.EnableRaisingEvents = true;
	blaise.Exited += new EventHandler(this.BlaiseEnd);
	bWait=false;
```