Integrating External Assessments into a Blaise Questionnaire

Joseph M. Nofziger, Kathy Mason, Lilia Filippenko, Michael Roy Burke

RTI International

1. Introduction and Justification

Many CAPI projects require external assessments (e.g., Woodcock-Johnson, K-bit, psychological tests, etc) to be called from within a Blaise questionnaire. These assessments often have highly complex routing instructions and would be extremely difficult or impossible to code in Blaise. We decided to use Microsoft .NET 1.1 to create a generic framework in order for a Blaise questionnaire to call any number of tests or assessments as desired. The system uses a Microsoft Access 2002 database to store the data during execution of the assessment. It then uses the Blaise Component Pack (BCP 2.0), specifically a .boi (Blaise Ole DB Information) file, to accomplish the data transfer from the Access mdb to the Blaise database. This presentation will focus on two methods used to pass data between Blaise and the assessment. We will also discuss status codes stored in the Blaise database which can be used to specify invocation parameters when calling the assessment, and methods to transfer data directly from a .NET application to a Blaise database.

1.1 Data Storage

In integrating external assessments into Blaise we had to consider first whether to import the data items back into the Blaise instrument and secondly, how to accomplish that. Strictly speaking, one could leave the data in whatever format the assessment stores its data. However, we decided to import the data back into the Blaise database for two main reasons.

- We often need to use some or all of it to drive subsequent skip logic.
- We wanted to avoid having to maintain interview and assessment data in separate files.

1.2 Invocation

The next consideration was to determine how to invoke the external assessment. Two methods were considered.

- Using external procedures
- Using an action

The action was chosen because of its simplicity. Creating an action for a user defined type to start the assessment executable is quicker and less error prone than programming an external procedure. As we will see, this had a bearing on some of our other decisions.

1.3 Data Transfer Timing

The last consideration was to determine the optimal time to combine the data. Three possibilities were considered.

1.3.1 Immediately after the assessment finishes

This option is preferable due to issues of separate storage and synchronization issues on the back end. Since the assessment is invoked via a "Start Executable" action and not via an ALIEN router, the assessment software cannot write directly into the Blaise database using BCP while DEP is running. Therefore this option requires that Blaise pull the data from the assessment data source. Two disadvantages of pulling data at this stage led us to explore other options.

The first problem situation occurs if an interviewer completed the assessment, but then failed to return as planned to the Blaise instrument. If the system was shut down before Blaise regained control then there would be no opportunity to read data through the .BOI file. If the case was exported at this point, due to assignment of a final status code or transfer to another laptop, then without backup measures the assessment data would remain on the laptop where it was captured.

The second problem situation arises if a data collector skips the question where the assessment would ordinarily be launched, suppressing the SIGNAL described later in section 3.1.2. The data collector later may realize the mistake and return to the appropriate question to launch the assessment. Unfortunately, after proceeding past this question the first time Blaise would try to import data using the .BOI file when the Access database did not have any data available. The BOI file data import only occurs the first time; the data are refreshed when the data collector goes back to the question a second time. Although the assessment may now have been completed, Blaise would still see the external database as empty and not populate the appropriate fields.

1.3.2 At the completion of the interview

This has the advantage of not slowing down the data collector and still making the data available upon re-entry of the interview after a sudden breakoff. It can be done outside the context of the Blaise interview and can therefore have unlimited access to the Blaise database. The primary disadvantage is that we would not have the assessment data together with the Blaise data in cases of transfers of partial data.

1.3.3 Immediately before transmitting the data back to our servers

The third option is similarly done outside the context of the Blaise interview. It has the considerable advantage of ensuring that the assessment data is included with the case data even if there is a failure such as a power outage during the interview. However the fact that we may need status and data items during later parts of the interview means that this cannot be the only time data is imported.

To be certain to get the data in all circumstances, we wanted all three options available. We had to develop two methods of importing data depending on when it occurred. In one case Blaise pulls data from the assessment. In the other, an external program reads the assessment data and pushes it into the Blaise database.

2. Goals and Constraints

We summarize here the technical goals of the solution, and list some constraints of the environment in which the Blaise interview and assessment must be run.

Goals

- 2.1 Run assessments independent of Blaise. In general, assessments have their own data stores and cannot be assumed to write directly into Blaise.
- 2.2 Integrate assessment data into a Blaise database. One goal was to insert data from external assessments into the Blaise data before the case leaves the data collector's laptop.
- 2.3 Integrate as soon as possible.
- 2.4 Move data redundantly to ensure capture. We need the assessment data regardless of unusual circumstances such as shutting down at the end of the assessment but before returning control to Blaise. For this reason a further goal is to…
- 2.5 Have the ability to initiate a data move outside the context of a Blaise interview.

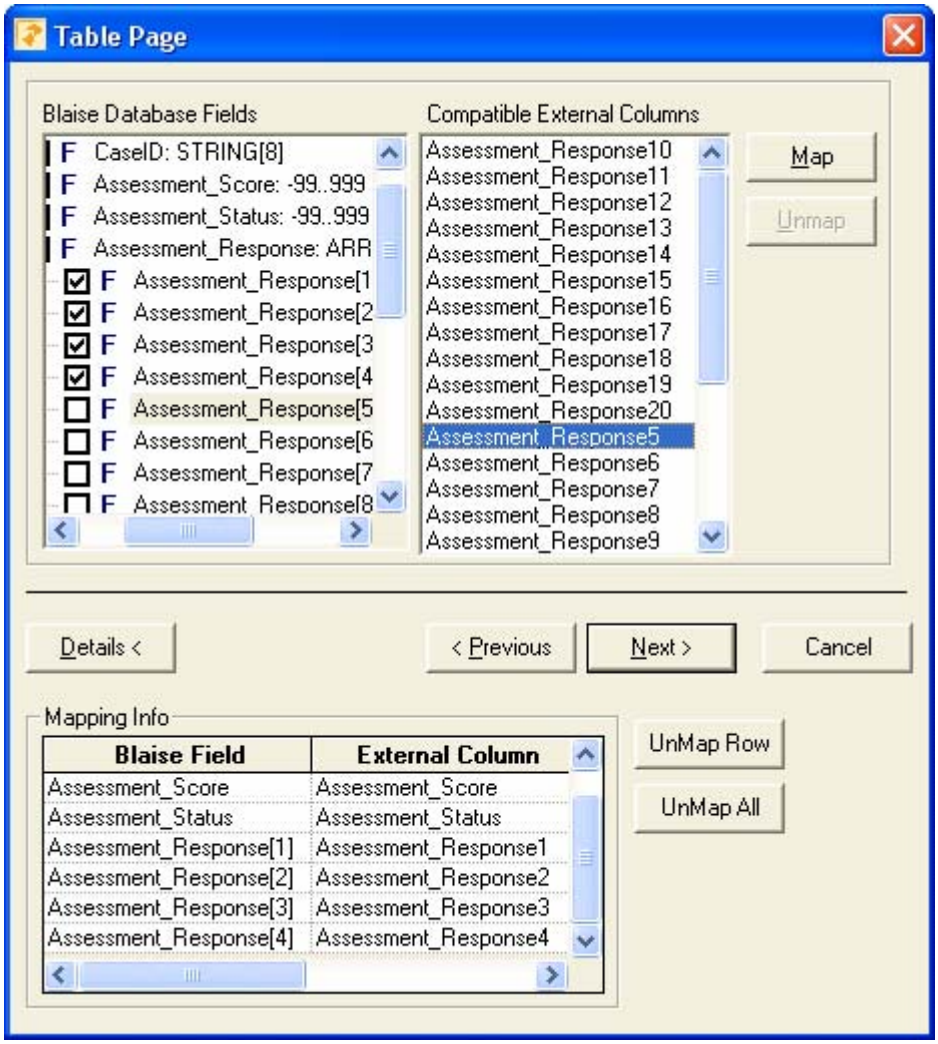
Constraints

- 2.6 Our solution had to support transfer of cases among intermittently connected laptops. Transferred cases can potentially contain partial data including assessment data.
- 2.7 When we export cases from a data collector's laptop, we prepare each case individually.

Goals 2.1 and 2.2 necessitated creating a means of moving data.
Goal 2.3 motivated us to ask the Blaise instrument to load the data immediately upon returning from the assessment.
To satisfy goals 2.4 and 2.5 we needed an additional way of moving the data, initiated from outside Blaise.

3. Data transfer initiated from within Blaise

We use a .BOI file to define the schema of the data we want to read, and how those fields map to fields in the Blaise database. While the BOI file may be automatically generated by a wizard, we found it more practical to define the BOI by hand and map external fields to Blaise fields or arrays.



3.1 Structure of the Blaise Application

We impose a structure on our Blaise application to support our use of the external data source.

3.1.1 Main Instrument File

The main instrument file (.bla) contains a reference to the meta file associated with the BOI file. The USES statement defines where to look for the meta definition.

```
{ Define external meta data files }
USES
      MyDataModel
```

Meta information for the BOI file is defined in MyDataModel.bla.

```
DATAMODEL MyDataModel
PRIMARY
  CaseID
FIELDS
  CaseID : STRING[8]
  Assessment_Score : INTEGER[3]
  Assessment_Status : INTEGER[3]
  Assessment_Response : ARRAY [1..20] OF INTEGER[3]
ENDMODEL.
```

3.1.2 Outer Block

In the main instrument, the data fields for the assessment are contained in two blocks, with one nested inside the other.

The outer block contains fields serving four purposes:

- preliminary questions leading up to starting the assessment
- a field used as a completion indicator (Yes/No)
- fields defining parameters needed to start the assessment
- a field referring to the block which will contain assessment data (ReadDataBlock)

When the data collector is going to exit the outer block, a SIGNAL is defined to check that the assessment is completed. If the completion indicator is not set to "Yes", the data collector chooses whether to suppress the signal or to start the assessment. After successful completion of the assessment, the indicator is set to "Yes". As described in section 3.1.4, the completion indicator is used to prevent future reads of the Access database and to ensure that we always KEEP (in the Blaise sense) the imported data.

3.1.3 Inner Block

The inner block contains a series of fields which hold the assessment responses and scores. Since assessments contain multiple tests, we include here status fields (not shown in the examples) which can be used to determine which tests need to be started.

The inner block contains an EXTERNALS section with the name and path of the BOI file that will be used to read result fields after completion of the assessment.

The EXTERNALS statement defines where to look for the data.

```
{Define Assessment external data file}
EXTERNALS
      MyExternalDB : MyDataModel ('MyDataModel.boi',OLEDB)
```

MyExternalDB, then, is a Blaise Field within which we can access the data items in our external data source as though they were members of an object. It also provides SEARCH and READ methods we will take advantage of in section 3.2.

3.1.4 Invoking the Assessment

A user defined type is created with action "Start Executable" for a field that is used as an entry point to invoke the assessment. To allow for re-entry of the interview without data loss, the completion indicator field is used as a gate to determine whether to enter the assessment or to keep the existing data.

```
Assessment_Complete.KEEP
{ Conditionally invoke the external assessment }
IF Assessment_Complete = yes THEN
      ReadDataBlock.KEEP
ELSEIF Assessment_Complete = no THEN
      StartAssessment   { User must click button }
      ReadDataBlock
ENDIF
```

3.2 Data Transfer from Access to Blaise using a BOI file

On completion of an assessment, control passes back to Blaise. At that point the external BOI file is used to pull data from the Access database back to the Blaise database. This is accomplished by using the SEARCH method of the field MyExternalDB. The SEARCH takes the case id as a parameter, and uses it to locate a record in the external table defined by the BOI file. The SEARCH returns a Boolean value indicating success or failure.

If the SEARCH is successful, then the READ method of the field MyExternalDB is used to load the correct record so that values from the Access database may be assigned to Blaise fields. The possibility of one assessment being completed on one laptop and another assessment being completed on another laptop for the same case requires that before making assignments conditional logic must be used to ensure that the Blaise field is not already filled. The code below shows an example of how this is accomplished. Here we show the word COMPLETE in place of the value the assessment stores to indicate successful completion.

```
{ Get the new value if the current value is not COMPLETE }
IF assessment_status <> COMPLETE THEN
      assessment_status := MyExternalDB.assessment_status
ENDIF

{ Assign the values for the score fields }
IF assessment_score = EMPTY OR assessment_score = init_val THEN
      assessment_score := MyExternalDB.assessment_score
ENDIF
```

```
{ Assign the values for the test item fields }
{ May loop through an array of values here }
{ or make multiple similar assignments }
IF MyExternalDB.assessment_response <> Empty THEN
      assessment_response := MyExternalDB.assessment_response
ENDIF

{ Set completion flag }
IF (assessment_status = COMPLETE) THEN
      Assessment_Complete := yes
ELSE
      Assessment_Complete := no
ENDIF
```

After all of the assessment data items are read, conditional logic is used to set a completion indicator Assessment_Complete. As described in section 3.1.2, this indicator informs Blaise whether the data collector is ready to proceed past the assessment to other sections of the interview.

4. Data transfer initiated from outside Blaise

We use the Blaise Component Pack and .NET technologies to transfer data from the assessment's Access database to the instrument's Blaise database.

The Blaise API COM component offers the ability to access and to modify a Blaise database from a variety of programming languages. .NET applications can easily use COM by creating a reference to Blaise API Objects 2.0. Since our assessment instrument itself is a .NET Windows Forms application (in C#), we created a .NET class library. The class in this dll that is of interest here will be referred to as DataTools.

Using a .NET class library allows the creation of programming objects which could encapsulate the functionality of the flexible and powerful ADO.NET objects for reading and restructuring the Access data with functionality of the Blaise API objects which could read and manipulate the Blaise database. We created an easy to use programming interface which allows us to inspect and update a record in the Blaise database with data from the Access database simply by instantiating a DataTools object and calling a method passing the appropriate Case ID as a parameter.

4.1 DataTools Services

The DataTools class provides several services.

- Clear data from a case
- Restructure the Access data
- Copy data from Access to Blaise

4.1.1 Clear assessment data from a case

Rules for saving data only require that we have a way to clear data for a given case. If a respondent were to break off in the middle of a test, they would need to start back at the beginning, with no responses saved from the previous run. In addition, since we consider incomplete test data invalid, it was important to not allow it to be copied back to the Blaise database.

4.1.2 Restructuring Access Data

To facilitate mapping data items in Access and Blaise for both push and pull methods, we want the transferable data items in Access to match a block of data inside the Blaise Database. The DataTools library uses ADO.NET objects to restructure normalized Access assessment data into a flat table with one record per case so that it will easily match the Blaise block where values need to be assigned.. It is this same denormalized table to which the BOI file attaches in the pull method described in section 3.

4.1.3 Copy data from Access to Blaise

4.1.3.1 Read Assessment Data from Access

Once the Access data is made ready for transfer to Blaise, ADO.NET objects from the System.Data.OleDb namespace are used to establish a connection and read data. The main objects used are OleDbConnection, OleDbDataAdapter, and OleDbCommand. One way to do this is shown in the VB.NET example below. We assume caseID and MyConnectionString variables are available.

```
Imports System.Data
Imports System.Data.OleDb

' Establish a connection to the db
Dim connection As New OleDbConnection
connection.ConnectionString = MyConnectionString
connection.Open()

' Configure a command object to select an individual case
Dim command As New OleDbCommand
command.CommandText = "select * from MyDataTable where caseid='" &
caseID & "'"
command.Connection = connection

' Configure an adapter to extract the data
Dim adapter As New OleDbDataAdapter
adapter.SelectCommand = command
```

Note from the CommandText that the data are selected such that the table will only contain one row of data corresponding to a case which needs to be updated in the Blaise database.

The DataTools object now fills a table in an ADO.NET DataSet object.

```
' Fill a table in a dataset with the record of the specified caseid
Dim data As New DataSet("ds")
adapter.Fill(data, "MyDataTable")
connection.Close()
```

Now a table called "MyDataTable" has been created within the DataSet object. It contains the one row of data which needs to be fed back to the Blaise database.

4.1.3.2 Write Assessment Data to Blaise

The reference to the Blaise API Objects 2.0 provides access to the objects BlAPI4A2.DatabaseManager and BlAPI4A2.Database. These provide the main functionality of establishing a connection to and modifying a Blaise database.

We now establish a connection to the Blaise database using the DatabaseManager and Database objects. This requires that the Blaise instrument be closed. In the real library, the Database object properties .DictionaryFileName and .DataFileName are assigned from properties of the DataTools object which are set by the constructor of the object, which takes as a parameter a path to an XML configuration where these settings are kept. For simplicity we show them assigned hard-coded values.

```
' Establish a connection to the Blaise database
Dim manager As BlAPI4A2.DatabaseManager
Dim BlzDB As BlAPI4A2.Database
manager = New BlAPI4A2.DatabaseManager
BlzDB = manager.OpenDatabase("")
BlzDB.StorageFormat = BlAPI4A2.BlStorageFormat.blsfBlaise
BlzDB.AccessMode = BlAPI4A2.BlAccessMode.blamShared
BlzDB.DictionaryFileName = MyBlaiseBMIFileName
BlzDB.DataFileName = MyBlaiseBDBFileName
BlzDB.Connected = True
```

We set the key to the appropriate case ID, then jump to the correct record in the database.

```
BlzDB.KeyValue = caseID
BlzDB.ReadRecord()
```

Now, data from the table in the ADO.NET DataSet can be transferred to Blaise using the following syntax. This example shows how the field "Response" inside the Blaise block "BlockB", which is inside the Blaise block "BlockA" can have its value set to "1".

```
BlzDB.Field("BlockA.BlockB.Response").Text = "1"
```

We store the Field names in an ArrayList and loop through them. Field names in "MyDataTable" must match those in the destination Blaise block. Field name mapping, if needed, could be achieved with an additional array dimension.

```
' Loop through all of the items and copy data for the selected caseid
from the MyDataTable table
For i = 0 To _itemList.Count – 1
    fieldname = Convert.ToString(_itemList(i))
    If BlzDB.Field(fieldName).Text = "" Then

        BlzDB.Field(fieldName).Text = _
   Convert.ToString(data.Tables("MyDataTable").Rows(0).Item(fieldName))

    End If
Next
```

Once the appropriate items in the Blaise database have been assigned, then the record must be updated.

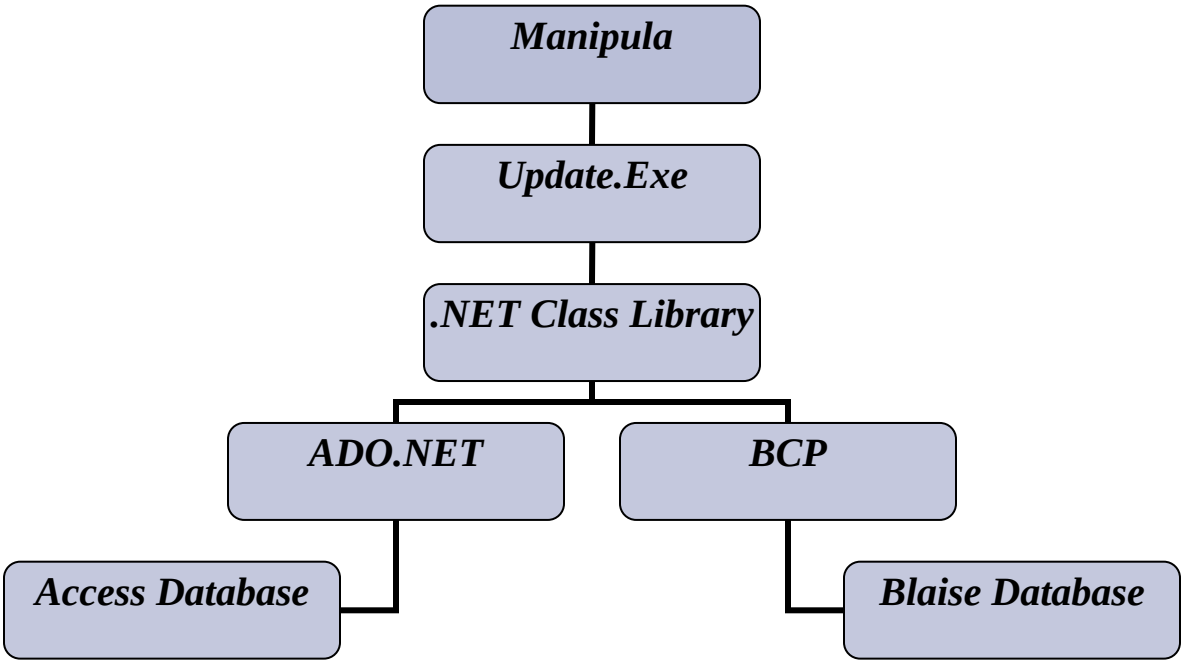
```
' Save the changes
BlzDB.UpdateRecord()
```

Finally, the connection to the Blaise database can be closed.

```
' Disconnect from the Blaise database
BlzDB.Connected = False
```

4.2 DataTools Context

We use Manipula both to invoke our interview instrument and to initiate case export. To satisfy our data transfer needs, the Manipula activates DataTools both following an interview and prior to an export. It does this by calling a simple exe which instantiates DataTools objects and invokes their methods. These relationships are shown in the diagram below.



5. Conclusion and Future Directions

Critical issues surrounding integration of assessment or other data collecting software with Blaise are whether to integrate assessment data and Blaise data, and if so, when to integrate.

When it is desired to move assessment data from an OLEDB database into the Blaise database, the Blaise Component Pack is useful whether pulling data from Blaise, or pushing data into the Blaise database from outside. For the former, BOI files can be employed to simplify the data transfer. The latter is needed to deal with some field data collection realities. In this case, the combination of ADO.NET and BCP objects provides a powerful means of data manipulation.

One possible improvement would be the insertion a generic invocation and data transfer layer between Blaise and the assessment. This layer might consist of an ALIEN router dll which would have access to the Database object exposed by BCP. When invoked, the dll would in turn invoke an assessment. Upon completion of the assessment, it would write data and status information back into the Blaise database via the Database object before returning control to DEP. This could eliminate the need for including data reading code in the Blaise instrument.