Using the Blaise Component Pack for All Stages of Data Collection

Lilia Filippenko, Roger Osborn, Vorapranee (Mai) Wickelgren, & Venkat Yetukuri, RTI International, USA

1. Introduction

At RTI International we have a number of studies where the Blaise Component Pack (BCP) has been used to create alien routers to conduct external assessments during computer-assisted personal interviews (CAPI). In addition, a number of applications have been developed to process Blaise interview data through the entire survey lifecycle.

This paper presents our experience with the BCP in collecting and processing the data for a recent project. During the development stage of the project, we created a number of applications in Visual Basic 6 and .Net to address our client's requirements:

- **Screener Selection** – An alien router that runs a series of algorithms to randomize and select eligible children to participate in a CAPI interview.
- **Create Cases** – An application that spawns a new case on a laptop to conduct an additional interview immediately upon completion of the screener interview. It is also used to create additional interviews during a nightly job at RTI after receiving the completed screener interview.
- **Fingerstick Timer** – An alien router and application that helps collect three blood samples during the CAPI interview at predefined time intervals.
- **Bio-Tracking System** – A web-based application to track events associated with bloodspots and saliva collection, shipment, and receipt.
- **Generate Mandatory Report** – An application that flags cases based on responses to certain questions during the CAPI interview.

Due to the complexity of requirements and time constraints, our goal was to have all applications easily configurable and testable. The paper describes developed applications and alien routers. It also includes examples of .Net classes used to monitor calls to executables and log discovered problems or errors.

2. Screener Selection

Many studies require collection of a household roster as part of a screener interview, followed by a CAPI interview with randomly selected household members. The study in question includes a screener interview, which utilizes a large amount of preload data collected from previous waves of this longitudinal study about respondents' children. Respondents have an opportunity to update as well as verify and/or add information regarding their offspring and/or other children they have parented. Upon completion of collecting this information, a series of selection algorithms should select eligible children to participate in the CAPI interview.

These selection algorithms include sorting of rostered children into three groups and using age and/or randomization to select up to two children in a group. Although sorting and randomization could be performed in the Blaise instrument itself, it is a complicated task that requires many hours for testing. We decided to create an alien router to perform the task and simplify the testing by using debugging tools in Visual Basic.

In the Blaise instrument at the top level of the data model we have created three arrays to save information about children:

- **ChildArray[1..13]** – has information needed for selection algorithms about all reported children. This array is used by the alien router to run the selection and is populated with the data just before the call to the alien router.
- **OutChildArray[1..13]** – has additional information about eligibility and results of the selection. This array is populated by the alien router after the selection is performed and is used later for analysis. Records are sorted according to selection algorithms.
- **SelectedChildren[1..2]** – has information about selected child/children needed to spawn CAPI interviews.

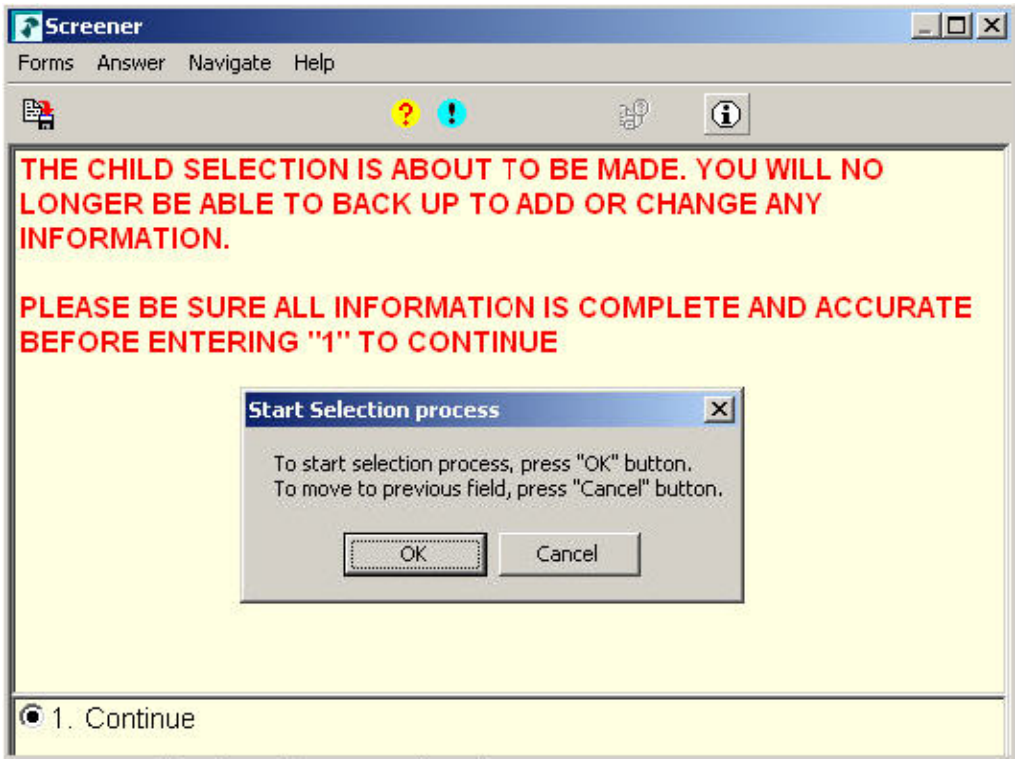

Figure 1. Call to Screener Selection Alien Router.

A Field Interviewer (FI) is asked to verify the child roster because the selection algorithms are executed only once to prevent changing the random number used to select a child, as shown on Figure 1.

After the FI clicks the OK button, the alien router reads ChildArray and adds records to three different arrays depending on a child relationship to a respondent. Records are sorted by age in each group and the selection algorithms are executed. As a last step in the alien router, information about selected children and sorted records is written to the Blaise database in SelectedChildren and OutChildArray, respectively. The FI receives a message that the selection is made and the Blaise interview continues.

3. Create Cases

3.1 Overview of Spawning Process

At RTI International a Case Management System (CMS) is used on FI laptops for many CAPI interviews. The CMS application allows Field Interviewers to update case status, enter comments, launch Blaise instruments, and synchronize the status of cases with a centralized SQL server database. A web-based Integrated Field Management System (IFMS) is used to assign and transfer cases between FI laptops and RTI. A web-based Control System (CS), on RTI's internal secure network, enables authorized staff to monitor the flow of data from the start of data collection through the creation of data files for analysis and delivery.

Sometimes, immediately upon completion of an interview, it is desired to launch a new, different interview for the same respondent. When this occurs, the new case is created on the FI laptop and is later automatically uploaded to a master Blaise database and IFMS during transmission.

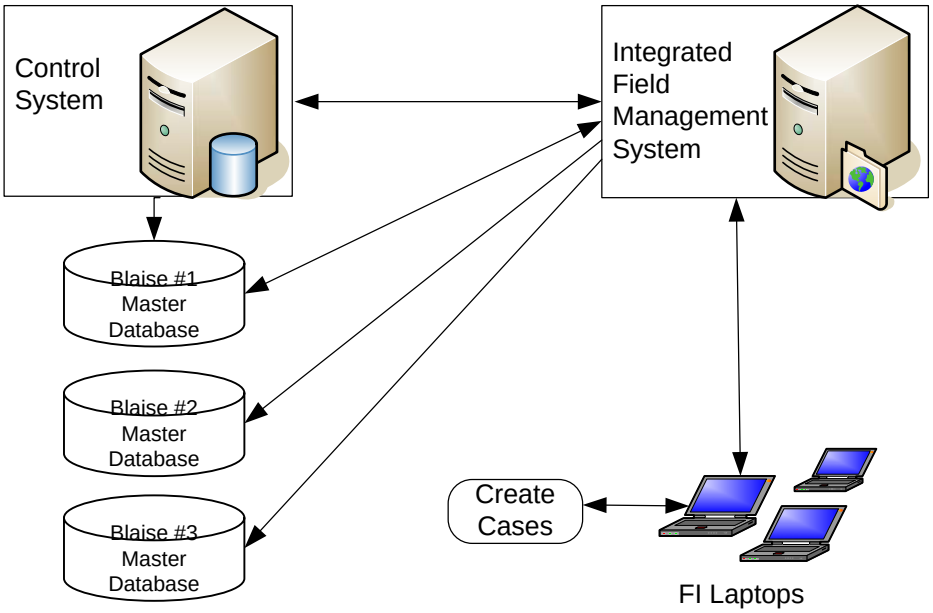

Figure 2. Data flow between FI laptop and RTI.

3.2 Create Cases Application

Typically, a Manipula setup is used by the CMS on FI laptops to spawn new cases upon the completion of an interview. However, in the described study Manipula was not well suited because of complicated requirements.

For cases where no children were selected, a new case should be created on the FI laptop to conduct an additional interview immediately upon completion of the screener interview. But for cases where one or two children were selected, cases should be created only in centralized Blaise databases to allow staff at RTI to prepare and send documents needed to conduct CAPI interviews with the respondent and children.

To meet requirements for the study, a Create Cases application was developed in Visual Basic using the BCP. It can run in one of three different modes (stored in a configuration file):

- **Production Mode on FI Laptop** – This spawns an interview immediately if desired and is used in the field.
- **Production Mode at RTI** – This spawns all CAPI interviews and produces special output files to be used by other systems (CS and IFMS) as shown in Figure 3. If cases were created, the application triggers an email to the RTI staff with information about the location of files that are ready for them to process in order to create documents for FIs.
- **Training Mode on FI Laptop** – This spawns all appropriate interviews that were supposed to be created after completion of the screening interview. It is used during training of FIs to show the whole process and allow training on the interviews that normally are not available on the laptop.

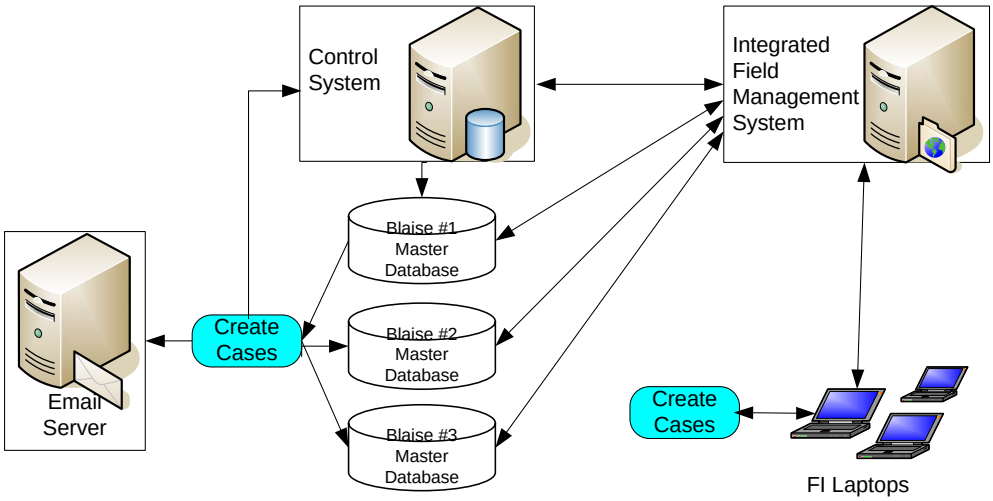

Figure 3. Data Flow between FI laptops and RTI with Create Cases application.

During the development stage of the study the data model of Blaise interviews changed many times. When a Manipula setup was used to create cases, a recompile was necessary for any data model changes. An advantage of using the BCP is that it allowed us to access information in one Blaise database and to use it to write data into another Blaise database independently of the data model versions involved in the exchange. Thus we made changes to the Create Cases application only when the fields needed to spawn new cases were changed, and that happened very rarely.

4. Fingerstick Timer

During the CAPI interview in the described study, respondents are asked to participate in a blood sample collection. Three blood samples ("fingersticks"), collected 20 minutes apart, were required. The blood samples were to be collected concurrent with the ongoing interview. Instead of stopping the Blaise interview entirely during the waiting periods between samples, a method was devised to allow an independent Windows application to run timers to remind the FI to collect each blood sample, yet still allow the Blaise interview to proceed. This "Fingerstick Timer" method involved a number of blocks defined in the Blaise instrument to facilitate usage of an alien router, a Fingerstick Timer alien router, and a Fingerstick Timer Windows application as shown in Figure 4.

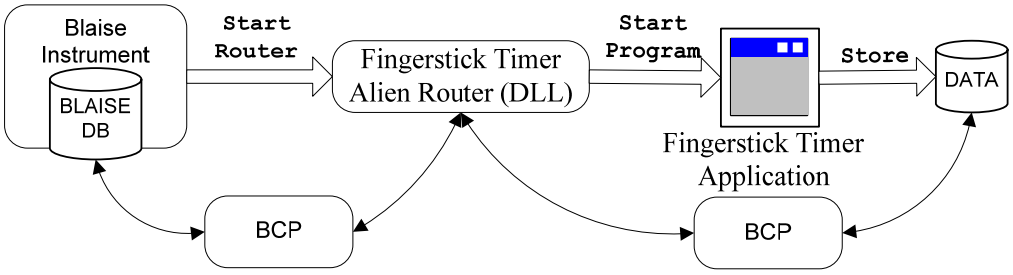

Figure 4. Fingerstick Timer Invocation and Data Flow.

The fundamental tasks of the Blaise instrument are to:

- launch the Fingerstick Timer alien router at some point during the interview, which in turn launches the Fingerstick Timer Windows application
- at some point later in the interview, launch the Fingerstick Timer alien router again to read the timer values collected by the Fingerstick Timer Windows application
- store the timer values into the Blaise database

The fundamental tasks of the Fingerstick Timer alien router are to:

- launch the Fingerstick Timer Windows application, passing the current Blaise case ID as a parameter
- read the comma-separated value (CSV) file output by the Fingerstick Timer Windows application and write the values into corresponding Blaise database fields
- log run-time information to a log file

The fundamental tasks of the Fingerstick Timer Windows application are to:

- display a pop-up window three times, 20 minutes apart, showing that it is time to collect a blood sample
- enable Breakoff of the blood sample collection (in case respondent refuses)
- record several date and time data points (exact date/time each 20 minute timer is started, exact date/time the pop-up window was presented, exact date/time each fingerstick was started, exact date/time each fingerstick was completed) and store these values in a separate CSV file for each respondent.

At some point during the Blaise interview, the respondent is asked to participate in blood sample collection, and if they consent the Blaise code calls the Fingerstick alien router, which in turn opens an instance of the Fingerstick Timer Windows application. The initial Fingerstick Timer application screen is shown in Figure 5.

The interviewer clicks on the "Begin Fingerstick" button when the fingerstick collection is started. The screen then changes to Figure 6. This screen counts up, in seconds, starting at zero. When the fingerstick collection is complete, the interviewer clicks on the "End Fingerstick" button and the Fingerstick Timer application starts a 20 minute timer then automatically minimizes itself so the Blaise interview is now showing on the screen. When the 20 minute timer expires, the Fingerstick Timer application pops up the window shown in Figure 5 again and the process repeats until three fingersticks have been collected. As each date/time value is captured, it is stored in a CSV file.

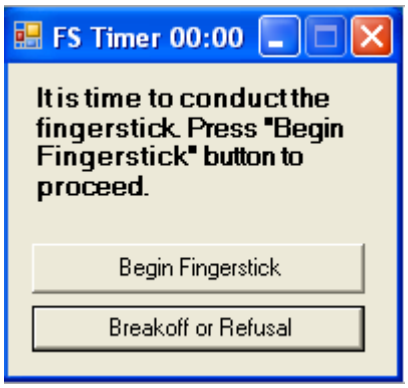

Figure 5. Initial screen

Figure 6. Time is counted up

At some later point in the Blaise interview, the Blaise code then calls the Fingerstick alien router again to get the results of the Fingerstick Timer application. First, the router ensures that the timer application is not still in progress, and if not it reads the CSV file and writes the values back into fields in the Blaise code.

Examples of .Net classes used to monitor calls to executables and log discovered problems or errors for the Fingerstick Timer implementation are provided in the Code Listing Appendix.

5. Bio-Tracking System (BTS)

5.1 Importing Field Data into BTS

When completed CAPI interviews are received at RTI, collected information is imported from the Blaise database during the nightly job at RTI into BTS by a.Net utility program, Load BTS. The program uses the BCP to open the Blaise database to access data collected in the Blaise interview and in the Fingerstick Timer application. This application reads more than one hundred fields per completed case and exports them into temporary tables in a SQL Server database as shown in the example below.

```
//Open Blaise Database
     BlAPI4A2.Database db = dbMgr.OpenDatabase(SourceDB);
    db.AccessMode = BlAccessMode.blamShared;
    db.Connected = true;
...
//Export Fingerstick Timer data
Time1 = db.get_Field("FngTmrResults.FngTmrResultsData[1].FSTimerEnd").Text;
Time2 = db.get_Field("FngTmrResults.FngTmrResultsData[2].FSTimerEnd").Text;
Time3 = db.get_Field("FngTmrResults.FngTmrResultsData[3].FSTimerEnd").Text;
...
//Export refusal answers
T_Sref1 = FormatData(db.get_Field("T_S.T_Sref[1]"));
T_Sref2 = FormatData(db.get_Field("T_S.T_Sref[2]"));
T_Sref3 = FormatData(db.get_Field("T_S.T_Sref[3]"));
...
```

The FormatData() function is used to export the value from the Blaise database and examine “Don’t know” and “Refusal” responses.

```
static public String FormatData(BlAPI4A2.Field obj)
    {
       string p;
       if (obj.Status == BlFieldStatus.blfsDontKnow)
          p = "9";
       else if (obj.Status == BlFieldStatus.blfsRefusal)
          p = "8";
       else
          p = FilterNulls(obj.Value);
       return p.ToString();
    }
```

After the completion of data importing steps, this program invokes an algorithm to identify the cases that are required to be loaded into BTS. The cases are separated based on sample type (blood spot or saliva). For blood spot, up to 3 samples on different color coded cards are collected with one or more fingersticks per card. For saliva a single sample is collected. A corresponding record is added to BTS for each collected sample.

5.2 Using Field Data in BTS

The application allows authorized staff at RTI to enter shipment details such as FedEx number, date shipped, time refrigerated, etc. Lab personnel who process blood spots and DNA look up the shipments by FedEx number or a special biospecimen ID number (BioID). The preloaded field data is used to verify information from FIs and simplify the process of entering additional information as shown in Figure 7.

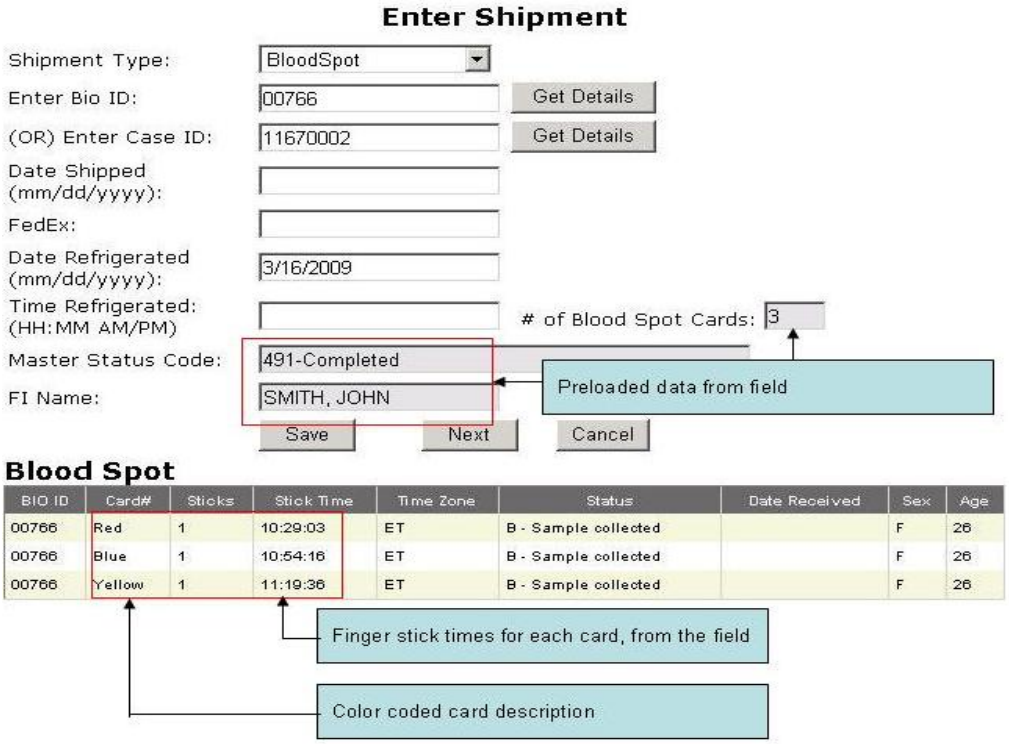

Figure 7. Enter Shipment Screen

When laboratory personnel enter results on the form, preloaded data from the field such as fingerstick time can be compared against the real time that was shown on the sample. This provides additional verification, improves accuracy, and confirms that the labs are entering the receipts against the proper sample.

A number of BTS reports provide information about data collected from the field to monitor FI performance, case status, sample shipment and receipts, sample quality, etc. For example, the Cumulative Bio-Specimen Status report by FI shown below uses more than 60 fields passed from the completed CAPI interview to calculate the total number of completed eligible interviews, the total number of respondents who consented, those who refused, the total number of samples collected by case, and lastly those cases where blood spots were collected but have yet to be shipped to the lab. The report also shows quantity by case to indicate if the FI collected all three blood spots or one/two blood spots per case.

	Overall - by Case										Quality - by Cards								Quantity - by Case			
FI_Name	Completed eligible interviews		Total consented		Refused		Total Collected		Collected not shipped		Adequate		Marginal		Inadequate		Unusable		All 3 blood spots		1 or 2 blood spots	
	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%
1. FI Name1																						
BLOOD SPOT	13	100.00	12	92.31	1	7.69	12	92.31	2	15.38	14	50.00	5	17.86	9	32.14	0		9	90.00	0	0.00
SALIVA	17	100.00	15	88.24	0	0.00	15	88.24	2	11.76	3	100.00	0		1		0	0.00	0	0.00	0	0.00
2. FI Name2																						
BLOOD SPOT	11	100.00	7	63.64	4	36.36	7	63.64	0	0.00	14	66.67	4	19.05	3	14.29	0		7	100.00	0	0.00
SALIVA	14	100.00	9	64.29	5	35.71	9	64.29	0	0.00	0		0		0		0		0	0.00	0	0.00
3. FI Name3																						
BLOOD SPOT	3	100.00	2	66.67	1	33.33	2	66.67	0	0.00	5	83.33	1	16.67	0	0.00	0		2	100.00	0	0.00
SALIVA	3	100.00	3	100.00	0	0.00	3	100.00	0	0.00	0		0		0		0		0	0.00	0	0.00
4. FI Name5																						
BLOOD SPOT	5	100.00	4	80.00	1	20.00	4	80.00	0	0.00	8	100.00	0	0.00	0	0.00	0		2	50.00	2	50.00
SALIVA	5	100.00	4	80.00	1	20.00	4	80.00	0	0.00	0		0		0		0		0	0.00	0	0.00
5. FI Name5																						
BLOOD SPOT	25	100.00	24	96.00	1	4.00	24	96.00	0	0.00	49	68.06	11	15.28	12	16.67	0		24	100.00	0	0.00
SALIVA	29	100.00	28	96.55	0	0.00	28	96.55	0	0.00	1	100.00	0		0		0	0.00	0	0.00	0	0.00

Figure 8. Cumulative Bio-Specimen Status report by FI

6. Generate Mandatory Report

Some studies at RTI require an alert be sent to staff so they will take appropriate action when certain interview answers of interest occur. This application is used to identify and report such events. Certain questions within several of the interview measures have been identified as indicators of concern. When particular responses are given to these questions in the CAPI interview, they trigger a series of follow-up questions. Once completed and

sent to RTI, the case is checked and run through the application. If the event is triggered, an email report is spawned and sent to the authorized staff alerting them to the need for further review.

Using the BCP, this Visual Basic application makes it possible to meet the requirement for checking 71 events and 538 variables, setting a flag in the Blaise database upon completion and sending an email report within 24 hours of receipt of the data.

Since there are many variables to be checked and to be reported, the application is written in a way that the code can be used with many variables. Events with the same structure but different question text are grouped in arrays.

```
'Define block names that have maltreatment flag
...
strFld(4) = "TCMC.TCM52"
...
strText(4) = "child ever physically abused"
```

When the application checks events, the base variable name of the event, FldName, is created from values defined in array strFld. The variable name of the follow-up questions is created by appending suffixes to the base name. The value from the Blaise database is assigned to a variable strTemp. The event descriptions are assigned to array strText.

```
'Check all elements in arrays
For i = 1 To 6
For j = 1 To 4
      'Define variable name of the event
    FldName = strFld(i) & "Arr[" & CStr(j) & "]."
    PrintName = "THL" & " - " & Mid(strFld(i), 6) & CStr(j)
    PrintName2 = "THL" & " - " & Mid(strFld(i), 6)
        If j = 1 Then
           FldName2 = strFld(i)
           PrintName2 = "THL" & " - " & Mid(strFld(i), 6)
           strLine = Space(4) & PrintName2 & " - " & _
              strText(i) & " - " & mBlaiseDB.Fields.Item(FldName2).Value & " (" & _
              mBlaiseDB.Fields.Item(FldName2).Text & ")"
           Print #intFile, strLine
        End If
     'Get value of the follow-up questions
        If mBlaiseDB.Fields.Item(FldName & "a").Status = blfsDontKnow Then
           strTemp = cDK
        ElseIf mBlaiseDB.Field(FldName & "a").Status = blfsRefusal Then
           strTemp = cRF
        ElseIf mBlaiseDB.Field(FldName & "a").Value > 0 Then
           strTemp = mBlaiseDB.Fields.Item(FldName & "a").Text
        End If
        If Len(strTemp) > 0 Then
           strLine = Space(4) & PrintName & "a" & " - " & _
                 "age when this happened - " & strTemp
           Print #intFile, strLine
        End If
          …
         Next
Nex
```

Below is the example of a mandatory report. The report displays an abbreviation of a section's name (THL), a variable name (TCM52, TCM521a, etc.), a variable description and the value of the answer.

```
THL - TCM52 - child ever physically abused - 1 (Yes)
THL - TCM521a - age when this happened -  11
```

THL - TCM521b1 - perpetrator - acquaintance - female - adult
THL - TCM521g - number times this person did this - 2

The configuration file is a file containing initial settings for an application. Using a configuration file helps make testing more efficient. There are options that specify whether a flag would be set, whom email should be sent to and the location of the input Blaise database. Transition from testing mode to production mode goes effortlessly with this configuration file.

7. Overnight Process

To accomplish successful exchange of the data between the different systems and the programs described earlier, an automated nightly job was set up. It was essential to execute a number of different programs in a predefined sequence to have IFMS, CS, and BTS systems updated correctly.

As a first step, cases from the field are loaded into Blaise master databases. Then, the three described programs, Create Cases, Load BTS, and Mandatory Report, are called sequentially to process the data as shown in Figure 9.

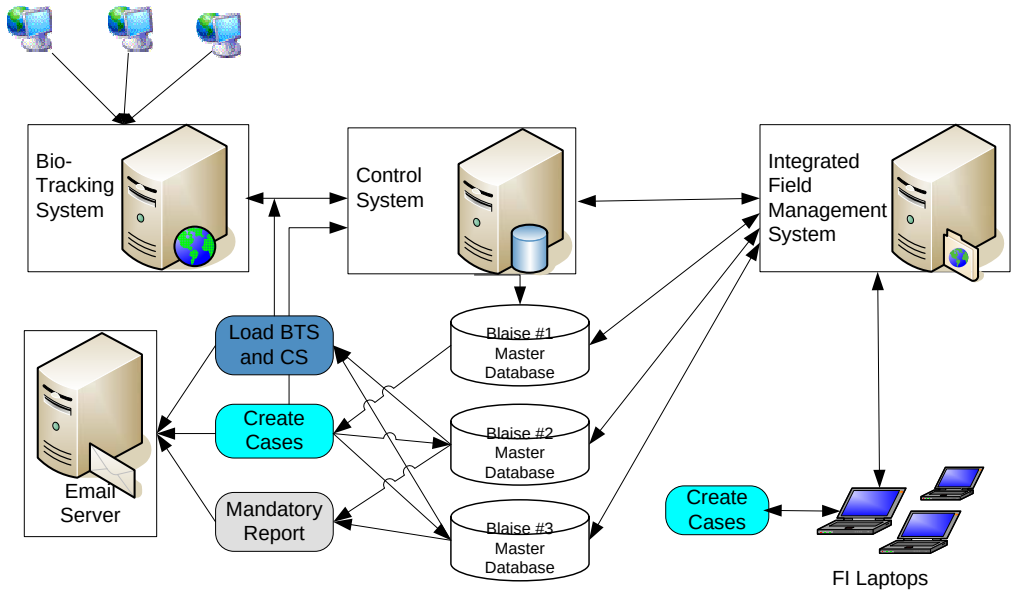

Figure 9. Data Flow between FI laptops, RTI, and Labs.

If necessary, the programs send emails to appropriate staff to inform them that a required action should be taken. Lastly, the newly created cases are added to IFMS for transmitting to FI laptops.

8. Conclusion

For the described study, the following advantages of the BCP were used:

- Ease implementation of sorting and randomization
- Fast programming of complicated tasks
- Ease of debugging the applications
- Effortless learning for experienced Visual Basic and .Net programmers
- Ease of adding already tested common classes and functions

The BCP reduces programming time for creating processes to pass data from Blaise databases into SQL Server databases, Excel, and other types of output files. For complex CAPI surveys that involve interviews of selected household members, collection of biospecimens, psychological tests, and special reports, usage of the BCP helps make data collection efficient and accurate. The BCP is being used at RTI extensively to replace Manipula when possible.

9. Acknowledgements

The authors would like to acknowledge Joe Nofziger and R. Suresh of the Research Computing Division at RTI International for help with this paper and for input into how to address some of the requirements requested.

10. References

Lilia Filippenko, Joseph Nofziger, Mai Nguyen and Roger Osborn (2006): Methods of Integrating External Software into Blaise Surveys, Proceedings of the 10th International Blaise Users Conference, Arnhem, Netherlands, May 2006.

M. Rita Thissen, Sridevi Sattaluri and Lilia Filippenko, (2007): Using the Blaise® Alien Router for Audio-Recording of In-Person Interviews, Proceedings of the 11th International Blaise Users Conference, Annapolis, USA, September 2007,

Appendix: Code Listings

Code Listing 1 – Log File Class

```
class clsLog
 {
   private String LogPath;
   public clsLog(String BlaiseDBPath, String strClassName)
   {
     LogPath = BlaiseDBPath + @"\Log";
     DirectoryInfo di = new DirectoryInfo(LogPath);
     if (!di.Exists) di.Create();
     LogPath += @"\";
     LogPath += strClassName;
     LogPath += @"_Log.txt";
   }
   public void Log(string message)
   {
     StreamWriter writer = new StreamWriter(LogPath, true);
     writer.WriteLine(DateTime.Now.ToString() + ": " + message);
     writer.Flush();
     writer.Close();
   }
    public void LogException(Exception ex)
    {
      string strException = "EXCEPTION:" + Environment.NewLine + ex.Message;
      strException += Environment.NewLine + "STACK TRACE:" + Environment.NewLine + ex.StackTrace;
      if (ex.InnerException != null)
      {
        strException += Environment.NewLine + "INNER EXCEPTION:" + Environment.NewLine +
ex.InnerException.Message;
      }
      this.Log(strException);
      System.Windows.Forms.MessageBox.Show(strException);
    }
  }
```

Code Listing 2 – Alien Router Class Log File Usage

```
  public class FingerstickTimerRouter
  {
     ...
    private string BlaiseDBPath;
    private clsLog objLog;
```

```
    public void Run(BlAPI4A2.Database data, BlAPI4A2.DepState state)
    {
      if (state.AlienRouterStatus != BlAlienRouterStatus.blrsPreEdit) return;
      this._data = data;
      this._state = state;
      BlaiseDBPath = data.DataFileName.Substring(0, data.DataFileName.LastIndexOf(@"\"));

      objLog = new clsLog(BlaiseDBPath, this.ToString().Substring(this.ToString().LastIndexOf(@".") + 1));
      objLog.Log(_data.KeyValue + ": " + String.Format("Instance started on {0} {1}",
DateTime.Now.ToShortDateString(), DateTime.Now.ToShortTimeString()));

     try
     {
      ...

      }
      catch (Exception ex)
      {
        objLog.LogException(ex);
      }
      finally
      {
        objLog.Log(_data.KeyValue + ": " + "Instance ended");

      }

    }
  }
}
```