

**Proceedings of the 15th
International Blaise Users Conference**

IBUC 2013

**Washington, DC USA
September 23 – 26, 2013**

Preface

This document contains the papers and posters presented at the 15th International Blaise Users Conference held in Washington, DC, USA from September 23 – 26, 2013. The conference included four days of technical papers and presentations on the use of Blaise and related topics as well as Blaise 5 workshops conducted by the Blaise Team from Statistics Netherlands.

The Conference Program was organized and planned by the Scientific Committee, chaired by Hilde Degerdal, Statistics Norway. Members of the committee include:

- Hilde Degerdal (Statistics Norway, Chair)
- Gina-Qian Cheung (University of Michigan, USA)
- Claudia Christ (Federal Statistical Office of Germany)
- Susan Corbett (NatCen, UK)
- Mike Hart (ONS, UK)
- Lon Hofman (Statistics Netherlands)
- Eric Joyal (Statistics Canada)
- Jim O'Reilly (Westat, Inc., USA)
- Mark Pierzchala (MMP Survey Services, USA)
- Jane Shepherd (Westat, Inc., USA)
- Rob Wallace (US Census Bureau)

The US Census Bureau, as the host organization, has collected, edited, and printed these proceedings for the benefit of the conference participants and others.

IBUC 2013 was organized and hosted by the US Census Bureau in conjunction with Westat. The organizing committee consisted of:

- Karen Bagwell
- Rob Wallace
- Tom Spaulding

Special recognition is given Jane Shepherd, Gaylen DiSanto, and their coworkers at Westat who provided invaluable assistance and guidance in this endeavor.

Table of Contents

Case Management

Real-Time Case Management with Blaise	1
Legacy Michigan CATI Sample Management System - Mixed Mode CATI/WEB	8
Multimode Rental Survey In Norway	22

Internet/Blaise IS Surveys

Customizing the BlaiseIS XSLT Stylesheets and ASP Scripts.....	34
Jumping around in Blaise IS.....	39
Challenges of Migrating ABS Business and Household Surveys to Blaise Web on a Large Scale and Short Timeframe.....	49
Blaise 4.8.4 Web Form Load and Performance Testing	64

Generating Blaise Code from Metadata

Generating Blaise from DDI	96
Automatic Generation of Blaise Data Models	103
Blaise Code Generator	120
Web-based CAI System for Blaise Instruments Development	122

Blaise and Tablets

Collecting Interviewer Observation Data via a Mobile Survey: Lessons Learned.....	131
Blaise On A Windows 8 Tablet. The Caribbean Netherlands Implementation	136
Developing A Web-Smartphone-Telephone Questionnaire	145

CARI/ACASI

Using Basil for ACASI at Westat-From custom to COTS.....	161
CARI Recorder Component Application in Michigan	176
The Application of CARI in Verification Systems	182

Blaise Instrument Development

Implementing Medical Care Surveys Using DEP and Manipula	183
DEP Unchained: Using Manipula on the Current Record in Memory	197
Using Blaise for Implementing a Complex Sampling Algorithm	205
Making the most out of Manipula-Maniplus	210

Blaise 5

Experiences and Lessons Learned Converting to Blaise 5.....	224
Implementing Blaise 5 in a production environment	243

Fascinating Blaise Topics

Testing a Complex Blaise CAPI Instrument.....	259
A Questionnaire Guide to Web Accessibility	272
A New Tool for Visualizing Blaise Logic	290
Centralization and Regionalization at National Agricultural Statistics Service.....	308

Paradata with Blaise

Blaise 5 Paradata Requirements.....	318
Using audit trails to monitor interviewer behavior under CAPI mode in China Family Panel Studies.....	332
Adding Business Intelligence to Paradata: The Blaise Audit Trail.....	333

Poster Sessions

Our experience in producing of the CAWI questionnaire for the survey "Statistical Report on the careers of doctorate holders (CDH)"	350
---	-----

Real-Time Case Management with Blaise

Leonard Hart and Erin Slyne, Mathematica Policy Research

Presented at the 15th International Blaise Users Conference, September 2013—Washington, DC

To meet the demand for more efficient data collection, Mathematica Policy Research developed an Integrated Survey Management System (ISMS). As part of the ISMS, we are developing a new case management application called the SCI (Survey Contact Interface). This application seamlessly combines the Blaise data collection system with our ISMS. The two systems will share data in real time.

In this paper, we describe the structure of the SCI and the process of integrating Blaise with it. We also discuss the advantages of the SCI as well as the challenges we faced during development.

1. Background

1.1 Real-time processing

Over the years, Mathematica Policy Research has been working to develop a centralized repository for survey data by re-designing our systems so they can share data in real time. Maintaining data centrally decreases the need to duplicate and transfer data between systems and provides instant real-time access. Real-time processes reduce problems like data de-synchronization and issues with data-transfer scheduling, thereby reducing the time spent troubleshooting. Ultimately, we seek to create a flexible system that saves valuable programming resources, provides end users with immediate access to their data, and is easily adaptable so it can be used for all surveys.

1.1.1 Our history with real-time processing

Over the past 10 years, we have extensively researched Blaise's capabilities to see how well they would meet our real-time processing goals. We researched Datalink, alien-router procedures, and event handlers, evaluating each in terms of its ease of implementation and how well it interacts with our other systems.

We explored the Blaise Datalink utility and discovered that, although it has some powerful capabilities, it did not meet our requirements. We needed the capability to store selected instrument information in its own database table, while simultaneously storing the survey management data in the database used by our Survey Management System (SMS). Datalink only allows users to save instrument data in a single database, with no ability to store common data in databases that share this information with different applications. By implementing some backend triggers in the database, we were able to move data between various tables within the databases in real time; however, this process became complicated to set up and maintain, and we concluded this option was not feasible.

We also researched Blaise alien-router procedures but discovered they require a predetermined list of field parameters. Given that our goal was to build a generic process that is flexible enough to use anywhere for any instrument, and that our data models differ for our various projects, this was not a feasible option. We were able to incorporate this process generically by defining a static, ordered list of all possible fields for integer, real, string, and so on; however, not all fields would be used for every instrument, so we had to ensure there were placeholders containing empty values for these fields. Unfortunately, it became too cumbersome to maintain a large list of field parameters in both systems.

Finally, we examined Blaise’s event-handler capability. In a previous paper for the International Blaise Users Conference,¹ we discussed how we successfully implemented this feature to call a Component Object Model (COM) method that invokes an external application. The methods in the COM object execute stored procedures in a SQL server database to move data between a Blaise data set and a SQL database and vice versa. An event can be linked in the data model properties file (.bdp) to a field’s user-defined data type or to the end of a parallel block. The COM object is automatically called when the user arrives at the associated field or at the end of a parallel block. Based on the capabilities we researched, this out-of-the-box capability is flexible, reusable, easy to implement, and most closely meets our real-time processing need.

1.1.2 Analysis of our current systems

Before designing a new system or implementing Blaise’s event handler, we decided to evaluate the advantages and disadvantages of our current systems. Our SMS system already centrally maintains data for multiple instruments including contact names and addresses, respondent incentive information, as well as case and instrument level statuses. However, much of this information is also maintained in the Blaise shell portion of our survey instrument, which transfers between the instrument and the SMS nightly using an overnight process consisting of batch scripts that process data. The Blaise shell, developed by Mathematica, consists of computer-assisted telephone interviewing (CATI) contact modules, the logic for setting case statuses, and a case call history array. The shell acts as a “wrapper” around our Blaise instruments. Occasionally, we encounter problems with the data transfer process including issues with network connectivity. This can cause the transfer of data to fail when the systems get out of sync as well as cause disruptions to interviewing after completion of the process. To minimize these problems, our new system will ideally collect and maintain this information in one centralized database in real time.

1.2 Planning the new system

Based on our experience using Blaise’s event handler to make real-time updates to an SQL database, we planned to build a large scale process to maintain management and survey data in separate databases and tables. Part of the plan was to move the section of Blaise code that collects management data into the Integrated Survey Management System (ISMS) through the SCI. This includes maintaining items such as respondent contact data and instrument status logic separately from our Blaise instrument, and thereby allowing us to remove this code from the Blaise shell. We still needed to maintain a few fields to be used by the call scheduler including telephone numbers, time zone definitions, and status codes. The SCI collects and maintains this information in ISMS and updates in the Blaise database in real time.

To illustrate how the ISMS will operate with a Blaise instrument (see Figure 1-1), we composed a preliminary outline. Blaise calls a generic COM object that invokes the SCI. Blaise would continue to act as the main survey data collection component with the exception of collecting the survey management level data. We incorporated the CATI contact modules and most of the demographic data collection into the SCI in order to reduce the amount of data being transferred nightly between the two systems, thereby eliminating redundancy and possible data discrepancies.

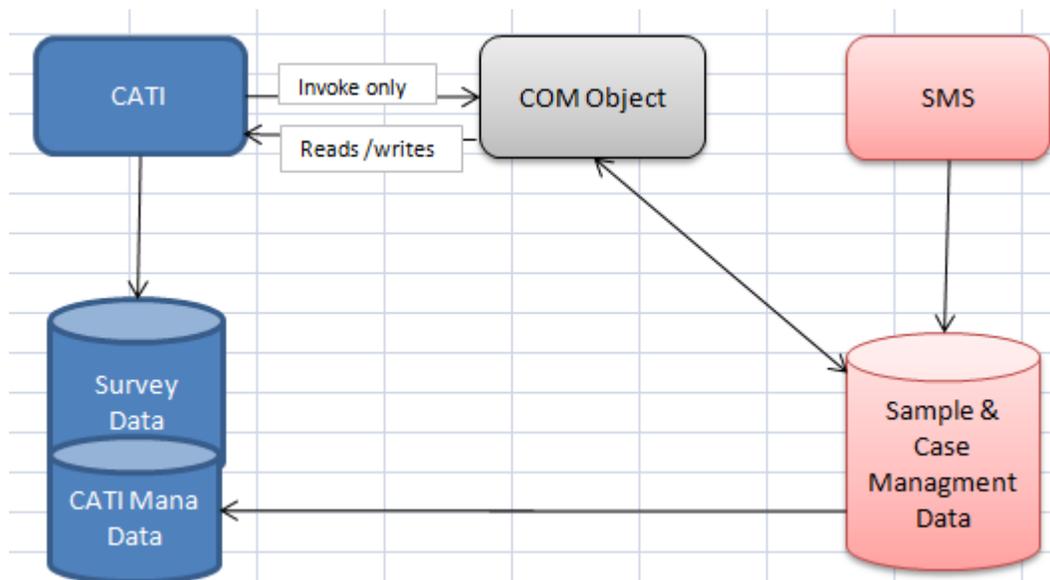
Next, we tested some of the new concepts to ensure they could work as intended. The SCI would be updating the Blaise dataset including the CATI Mana block, which is used by the call scheduler. We verified that we could modify this block externally without the Blaise system overriding its values.

¹ Hart, Leonard, Scott Reid, and Erin Slyne. “The Challenge in Balancing Data Collection Innovations, Remaining Practical, and Being Cost Effective” Blaise Users Group website. Available at <http://www.blaiseusers.org/2012/papers/01a.pdf>. Accessed June 14, 2013.

We also wanted to prove that we could seamlessly transition between the SCI screens and the Blaise Data Entry Program (DEP). After developing and testing a small instrument, we confirmed we could easily switch between the different application screens.

We built a small prototype that executed one specific path of logic and combined the proof of concepts described above. This prototype showcased the continuous transition between the two systems and proved that data could be read and written to the Blaise dataset using the Blaise Application Programming interface (API). As we developed and tested our prototype, we became confident that we could build this system but understood we might encounter unexpected challenges. However, we felt these challenges were not large enough to deter us from our ambitious goal of creating a centralized repository of all data that is real time.

Figure 1-1. The ISMS



2. Designing the SCI

After we proved these concepts, our next step was to design the ISMS application. We organized the design documentation into two parts: (1) specifications that included screen layouts and detailed logic for setting a case or call status and (2) requirements that consisted of the business rules and case flow process diagrams.

2.1 Defining the SCI and Blaise specifications

We used our Blaise shell as the foundation for designing the SCI because it serves as the backbone for all of our Blaise survey instruments. Unfortunately, the specifications for the Blaise shell were not detailed enough for this document and we were tasked with updating the written specifications to meet the new processes. This process, although time consuming, was advantageous to non-Blaise programmers because it provided a thorough understanding of how the Blaise shell worked before starting to program. The programmers also spent time exploring the Blaise DEP interface to get familiar with how it operates from a user's perspective and for graphic user interface (GUI) development research.

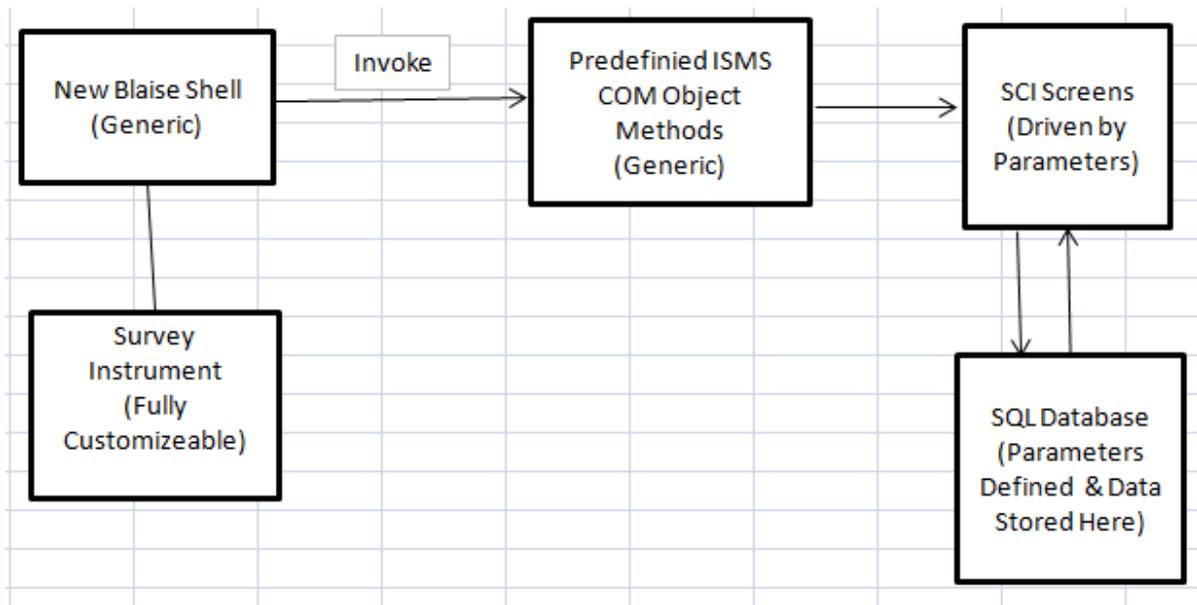
Because we wanted data collection to operate seamlessly between both systems, the SCI would use the same CATI contact screens and logic for setting case and call statuses as our current Blaise shell.

Aesthetically, we wanted to make the SCI screens look and operate as identically as possible to the CATI screens in the Blaise DEP. This included a keyboard driven interface and an info and field pane-type screen design.

2.2 Defining the SCI and Blaise requirements

To meet our goal of developing a generic, parameter-driven system that would require minimal customization, our requirements needed to be specific. We developed a detailed diagram outlining which parts would be kept generic, which would require parameters, and where those parameters would be defined (see Figure 2-1).

Figure 2-1. A Generic, Parameter driven system



As previously mentioned, the Blaise instrument would use events to invoke COM object methods using the Blaise API. The COM object methods would also update the Blaise dataset so that the user would be routed to the correct Blaise field upon returning from the SCI. The CATI Mana fields would also need to be updated to control the case delivery. Exposing the entire Blaise instrument in this way could potentially risk data integrity. For example, a programmer could accidentally assign an incorrect field value or inadvertently alter the data. To help avoid mistakes, we defined a list of all COM object method calls that Blaise could make at specific points and the fields the COM object could update. Any changes to these generic methods would have to go through tight code review before deployment.

Each method invoked from Blaise calls a set of SCI screens. However each project often has different requirements including project specific question text and the number of fields. Since the SCI can easily read and write to the SQL database, we specified parameters in the SCI and defined their values in SQL tables, rather than making text updates for each project. For the non-text modifications, we built extensible modules using Windows' Managed Extensibility Framework (MEF). Each SCI page calls out to a module when it first loads.

Because the majority of the new system is controlled by the SCI, we decided to pare down the current Blaise shell to a minimum number of Blaise management fields that need to be shared with the SCI while retaining only those used to maintain the call scheduler. We also defined the points of exit from the Blaise

instrument where the COM object was to be invoked and which fields it would update. Possible points of exit include interview break offs, setting up appointments for callbacks, completed surveys, or collecting new information about the respondent. For each point of exit, the call would be to a specific, predefined COM object method with an associated predefined set of business rules.

2.3 Bridging the ISMS and Blaise systems

The smooth transition between the two systems was critical to the success of the ISMS. From the CATI interviewer's perspective, the screens need to flow seamlessly, without disruptions or hesitations. To accomplish this, we used Windows message queuing. The COM object puts messages into the queue while the ISMS acts as a listener to read and process each message. From the Blaise side, we thoroughly tested the system to ensure all calls to the COM object methods were valid. These methods also had to assign the correct values to the Blaise fields including the CATI Mana block. Because we were unsure of the CATI Mana block field value assignments that Blaise performs internally, we ran several tests of all the various call results in our current Blaise shell with the watch window turned on before recording them in our specifications.

3. Challenges faced during development

We experienced several challenges while developing the ISMS including programming the SCI user interface, communication between .NET and Blaise programmers, bridging the gaps between the two different systems, and defining specifications for the complex Blaise shell. We were able to resolve the majority of issues by communicating using telephone, email, and face-to-face meetings. We conducted 15-minute scrum meetings to keep track of progress.

3.1 The SCI design

One challenging part of the SCI development was accurately replicating the keyboard-driven Blaise DEP interface. We wanted both applications to look as similar as possible to avoid creating any potential interviewer bias in the data collection or confusion when interviewers are switching back and forth between applications. Our current Blaise DEP application design includes an info pane where the question text is displayed and a field pane where a response is entered. The return key is pressed to allow a brand new info pane to appear. The field pane often remains on the same page so that responses to other questions are visible. Initially, the SCI did not capture the info/field pane concept, and a mouse was needed to select a response. There was no field pane box where the user could enter or edit a response with the keyboard. Eventually, the screen was developed so it operated almost identically to the DEP, with no need to use a mouse. Additionally, the designers wanted to maximize screen space in the SCI application by placing multiple questions on each page. Once the enter key is pressed after entering a response, the question is disabled and becomes re-enabled when the user presses the up arrow key.

3.2 Seamless operation

Joining the two systems so they operate seamlessly was another difficult challenge. In particular, we experienced a "focus" problem when returning to the Blaise instrument from the SCI. Instead of routing to the next unanswered field and being able to type a response, the user had to click on the screen first. This problem was solved by incorporating Windows system commands into the .NET code (see Figure 3-1).

Figure 3-1. Windows commands to fix focus problem

```
Application.Current.MainWindow.Visibility = Visibility.Visible
Application.Current.MainWindow.WindowState = System.Windows.WindowState.Maximized
Application.Current.MainWindow.Activate()
Dim proc As Process = Process.GetCurrentProcess()
Dim hWnd As IntPtr = proc.MainWindowHandle

SetForegroundWindow(hWnd)
Application.Current.MainWindow.Topmost = True
```

Set ForegroundWindow is defined as:

```
Declare Function SetForegroundWindow Lib "User32.dll" (ByVal hWnd As Integer) As Int32
```

When using the Blaise API, we also encountered a floating point unit error when trying to update several different fields in the Blaise instrument. Statistics Netherlands' resolution was to use the latest BI-API3A.dll and the 3.5 .NET framework. Because the ISMS application was written to use the 4.0 .NET framework, reverting back to 3.5 was not a feasible option, making it difficult to overcome the floating point problem. So far, we have been able to program around this issue while anticipating that it can be addressed in future releases of the Blaise API.

We also experienced some issues with executing parallel blocks using the Blaise API. With help from Statistics Netherlands, we discovered we needed to cross reference the parallel with its associated numerical index in order to ensure the correct parallel is called from the SCI. For the time being, if the parallel block changes, we will have to change the parallel index number in a configuration file. Currently, the parallel index solution works but it is not generic; we will need to revisit this during development for a future release of the SCI. Our goal is for the application to be capable of determining the proper parallel index.

3.3 Specifications

One challenge in particular was to create and edit specifications for the new system based on the complex Blaise shell. Capturing and formatting all the variable assignments with corresponding screenshots was difficult and time consuming. The .NET programmers frequently needed clarification because they did not understand the methods of specifying Blaise logic. Through increased communication we were able to work together to resolve misconceptions.

4. Conclusion and Future of ISMS

Developing the initial version of ISMS has been challenging, but we have learned several important strategies for improving future releases. First, good communication is imperative to implementing a quality system. A project can consist of competent programmers that successfully create separate modules, but if they do not collaborate, the desired outcome will not be achieved. Good communication involves selecting the most effective mode to accomplish a particular task. For example, email is more effective for lists and reminders, while face-to-face and telephone conversations are good for talking through problems. Meetings are an effective way for multiple people to disperse and gather information. As mentioned in Section 3, we used scrum through most of the development stages. During these 15-minute meetings, key development staff would give a brief update of their progress and implementation

plans for the immediate future. This kept everyone focused on his or her task at hand without having to go into great detail.

Second, we learned that the procedure for defining requirements was as important as the content. The SCI and Blaise specifications were combined into one document rather than maintained separately. In hindsight, this was a mistake, but at the time we thought this was the best solution. We thought we could modify our current documentation instead of creating another document. Unfortunately, this added confusion to the process of understanding what items belong in the old system versus what should be included in the new system.

Third, we discovered that the experience of developing the SCI has allowed its programmers to learn different technologies and applications. At times, they had problems communicating with other programmers with different expertise. During development, some struggled to understand how other parts of the system would work. This experience has broadened their knowledge of other systems and improved communication for future projects.

Finally, during planning and development stages, we discovered that due to cost and complexity, we needed to postpone several features we wanted to incorporate into the first version of the SCI. To avoid complication, we decided to build the SCI for CATI data collection only but kept in mind that computer-assisted web interviewing (CAWI) and computer-assisted personal interviewing (CAPI) would eventually be incorporated. As we encountered items during the development process that could affect multiple data collection modes, we either addressed them immediately or noted them for future releases. We also noted the capabilities we wished to add to the next version and possible advancements to research for feasibility review.

For future versions of ISMS, our primary goal will always be a centralized repository of data for all applications used to conduct surveys. After successfully implementing several CATI surveys, we plan to start modifying the SCI for CAWI surveys to accommodate our multi-mode surveys that utilize CATI and CAWI.

As of writing this paper, Mathematica is developing its first production CATI survey using the ISMS with the SCI. At the next Blaise conference, we hope to report on our successes with implementing the system, including any additional lessons learned.

Legacy Michigan CATI Sample Management System – Mixed Mode CATI/Web

David Dybicki and Peter Sparks, University of Michigan

This paper examines three distinct areas for implementing a mixed-mode survey: SMS, Blaise IS, and routines needed for mixed mode. The project used for implementing the mixed mode has been referred to generically.

1. SMS

SMS is short for Sample Management System, and was developed by a small team at the University of Michigan over a short period of time and went into full use somewhere in the fall of 2001.

1.1 History

SMS was designed using paper prototyping. Paper prototyping was not just a buzz word but proved useful for the design process. It gave the end user control over the final outcome of the system, in terms of look and feel and actions. This dramatically sped up the process of development.

The first project was deployed in parallel using the old paper handling method, and the “new” case delivery system. The project had monthly data collection requirements and proved optimal for comparing results.

Since the system is programmed in Maniplus, it made changes simple and the code was portable, maintainable, and could be tailored to most project needs. SMS has the ability to work with any file type that is supported by BOI files, and has worked with SQL server as well as native BDBs.

The design separated the SMS-related fields (“INHERIT CATI”) from the survey-specific datamodel. This design allowed for complete sharing of the survey datamodel between modes (CATI & SMS/CAPI & SurveyTrak¹).

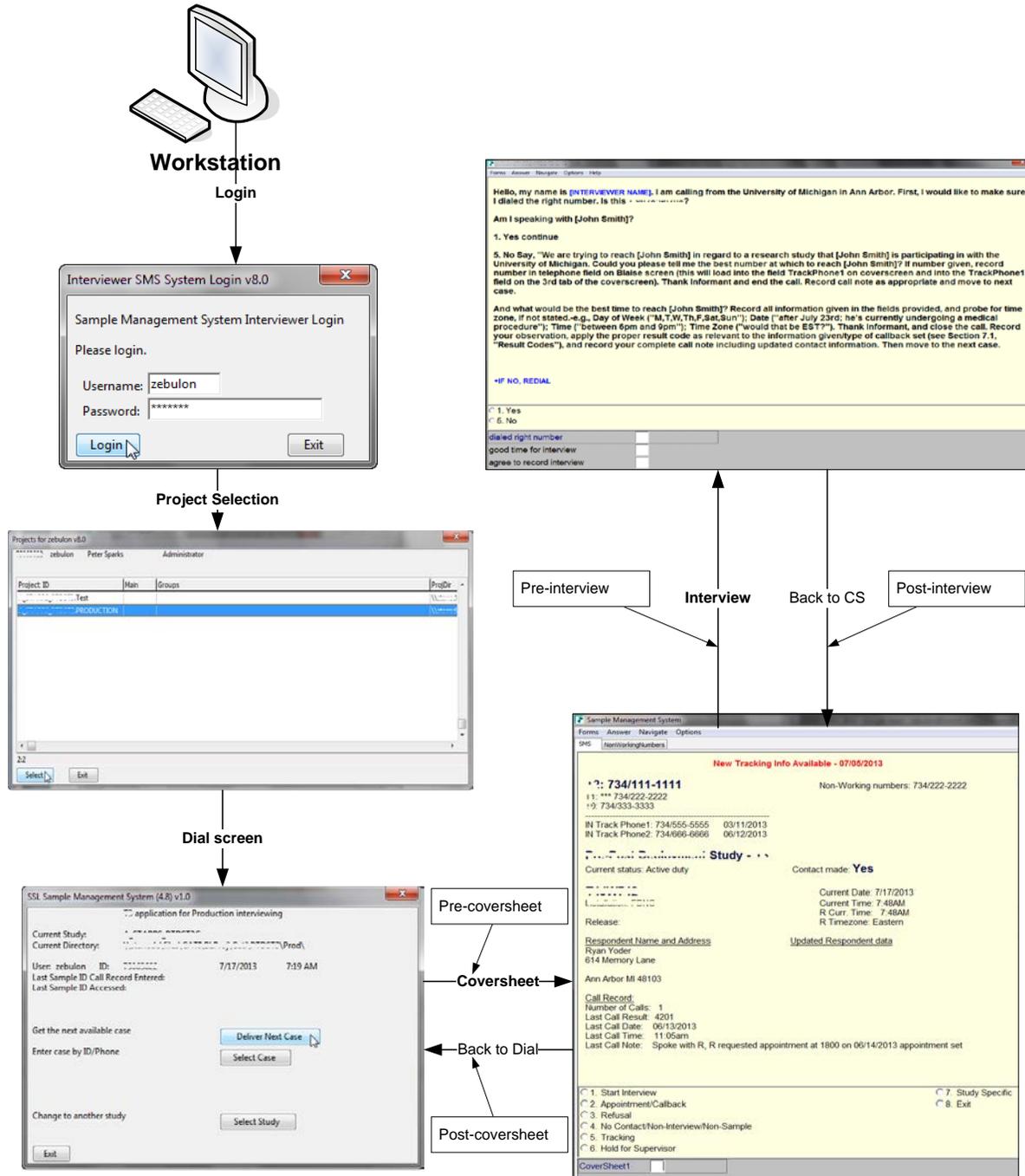
1.2 Implementation

SMS was developed around the Blaise call scheduler “out of the box” using Maniplus. Three major scripts are run: SMSPicker, SMSCati and SMSSuper.

1.2.1 Interviewing process

The diagram below shows the interviewing process: logging in via SMSPicker and selecting a project, then interviewing using SMSCati. SMSCati runs the call scheduler to retrieve the next available case according to the parameters specified in the CATI specifications. A very minimal interface allows the interviewer to receive the next available case via the call scheduler, to select a case via phone/primary identifier, or to select a project. The coverscreen replaces the “make dial” screen.

CATI Interviewer Process



An important design feature is the ability to run additional Manipula/Maniplus scripts between SMS, the coverscreen datamodel² and the study datamodel³. This is key to implementing a mixed mode survey between CATI and web. This implementation of SMS and mixed mode uses the post-coversheet (data is written to the web management database) and the pre-coversheet and post-interview (sets section flags in the CATI and web interviews, respectively).

1.2.2 Management mode

The supervisor mode, SMSSuper, works with sample (release, review, edit, assign), with users (groups, logins, projects), with call records and result codes⁴, call history, readback & interview mode. In the interview mode the mixed mode flags (see 3.9.2 below) are also checked and set.

1.2.3 System audit trail

Every significant action of the supervisor or interviewer, such as logging in to the system, retrieving a case, changing a user's role, etc., is recorded in trace files. This helps resolve the problem of incorrect/missing data in the case management.

1.2.4 CATI audit trails

Standard ADT files are stored for each interview. They are used for interview timings, and to possibly recover an interview in case that case becomes corrupt as well as to debug complex logic flows. It can be used to help determine interviewer behavior for problem screens.

1.2.5 CARI recordings

The recording of audio has been set to infinite to capture all the dialog between the respondent and the interviewer for quality control. Screen captures of the question are also stored. Sometimes, the dialog cuts out abruptly when the interviewer presses Enter too quickly and leaves the question. We have found the CARI recordings to be very useful.

1.3 Coverscreen uniformity

Though it would seem very restrictive to maintain only one structure for all SMS projects, each coverscreen is actually very flexible. All projects maintain the same base coverscreen structure. This is a datamodel that contains the INHERIT CATI command, as well as standard fields for use within the system. Study-specific questions are added using auxfields, and any study-specific data, such as name, phone number, address, and age, are stored in additional arrays: UserValue[], KeyStats[], IwerObs[], ContactObs[].

This coverscreen also contains result codes for the interview, such as a "Ring, no answer" or an initial refusal. This means that any reports, utilities, or routines created to work for one study will work for all studies in SMS.

1.4 Other functions

Extensions to SMS include programs that create reports for managing the sample, track interviewer progress and hours-per-interview. The task scheduler is used for creating new day batches automatically, running Hospital to rebuild databases to keep them healthy, and executing Manipula scripts that are specific to the project, such as setting sub-priorities⁵, moving sample between modes, and so forth. There are built-in filters in SMSSuper that allow supervisors to quickly locate cases and work with them.

1.5 New data model releases

Data model changes solely concern the study data model and not the coverscreen. As previously mentioned, the coverscreen data structure is the same between all SMS projects. Most migrations are standard: backup the data, use a Blaise to Blaise Manipula script (via the wizard), and then replace the production data model and data with the migrated data. This same script is also used for SurveyTrak field projects. Project specific Manipula

scripts, such as the pre-coversheet, post-coversheet, and post-interview routines are also have to be prepared again.

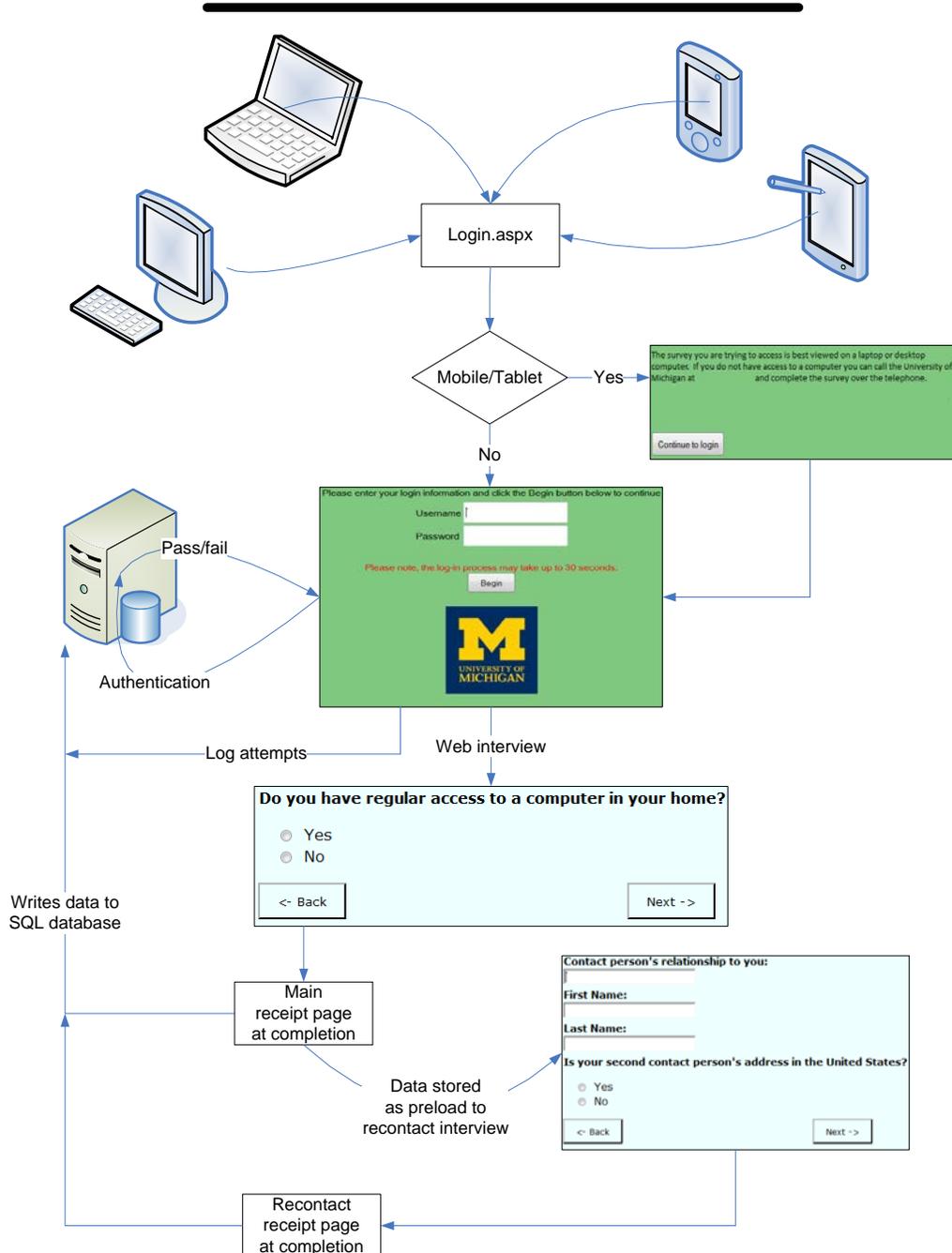
1.6 Legacy limitations

Like all legacy systems, there are a number of items in SMS that are could be improved. The interface is functional but looks and feels old. An out-of-date VB6 DLL is used by the system to manage a shared text file to achieve fast reads in a few files. However, this should be changed by storing the data in a relational database. Project maintenance can be time-consuming because of constant monitoring to ensure daily/nightly batch files run correctly. Reports are functional but need to be updated or replaced. There are many files and scripts, and although it provides flexibility it is also more difficult to maintain and catch errors. And, because of this, SMS is fragile and one failure, such as a locked or missing file, can bring a project to a halt.

2. Web

The process for the respondent using entering a Blaise IS interview is illustrated below.

Web Respondent Process



2.1 Survey process

The respondent enters the survey at the login page via some device (i.e. PC, laptop, cell phone). The login is a .Net web application, and it first checks the user's device. If it is a mobile device then a

warning page is shown stating the survey has been formatted for desktop browsers. The login page queries a backend SQL database with username & password, and if all is well the database returns the ID of the interview, different than the username, that is used to start the Blaise IS interview.

Note: if the interview has been marked as complete (a status from either SMS or web), or the case has been locked in web, then the web management system sends an invalid ID back to the login page.

2.2 Paradata

Similar to the function of ADTs, the web mode uses client side paradata (journaling) to capture information about the user's actions on a web page when answering. This data is stored in a database on the data server for timing analysis. It also can be used to help recover the data within an interview. It can be used to help determine respondent behavior for problem pages.

The recontact interview is not journaled. The recontact interview contains sensitive information, such as respondent contact information and payment. The purpose of separating the main interview and the recontact interview is to maintain a separation of respondent identifying information from survey data.

2.3 Web management database

The recontact interview is always conducted at the end of the main interview. At the end of each of the interviews, the Blaise IS receipt page is run. The pages have been modified to not display any information, but rather write key variables and other data back to the web management database.

Note: none of the flag settings occur at this point; all of those actions are taken with SMS.

2.4 Main and Recontact

2.5 New datamodel releases

A more complex process than migrating SMS is used. The web mode is made inactive via the Internet Server Manager, the web surveys and data are backed up, the new survey is deployed, the data is migrated back to the production area, and the web mode is again made active. The downtime for the web survey is planned and short, and occurs during pre-determined light usage times.

3. Mixed Mode

We use the definition of a survey that has been implemented for CATI (interviewer-assisted, Blaise 4.8, using the DEP.exe) and web (respondent self interviewing, Blaise IS 4.8, browser). The survey is automatically or programmatically sharing sample between the two modes and the case can be switched between modes repeatedly.

3.1 Blaise IS management

Key to managing the sample in web is the management database. It is used for storing key data values, such as respondent name, address, phone, CATI/web mode, contact information, and key variables from the survey. The management system also generates email invitations, reminders, text messaging, and form letters. Respondent incentive payments are also handled through this system. Data from SMS is pulled into the system every five minutes. Data from the web interview is written to the database at the end of each interview's completion.

3.2 Case locking

After careful deliberation, it was decided to handle case locking only within the CATI mode; all cases loaded in the web are always available. The sample is duplicated between CATI and web, and in CATI the sample is released by replicate for data collection. This means that a respondent could theoretically take a web survey at the same time as a CATI survey, but it is unlikely.

This design allows respondents, who are initially assigned a web interview, to call a toll-free number and to request an interviewer-assisted interview.

3.3 Implementation of the survey in CATI and web

The question order and code frames were kept as similar as possible in order to maintain consistency, even though there is not a one-to-one migration of data between the modes. CATI mode has fewer questions than web, and the types for some questions are also different but the field names remain the same.

3.4 Grouping and Formatting

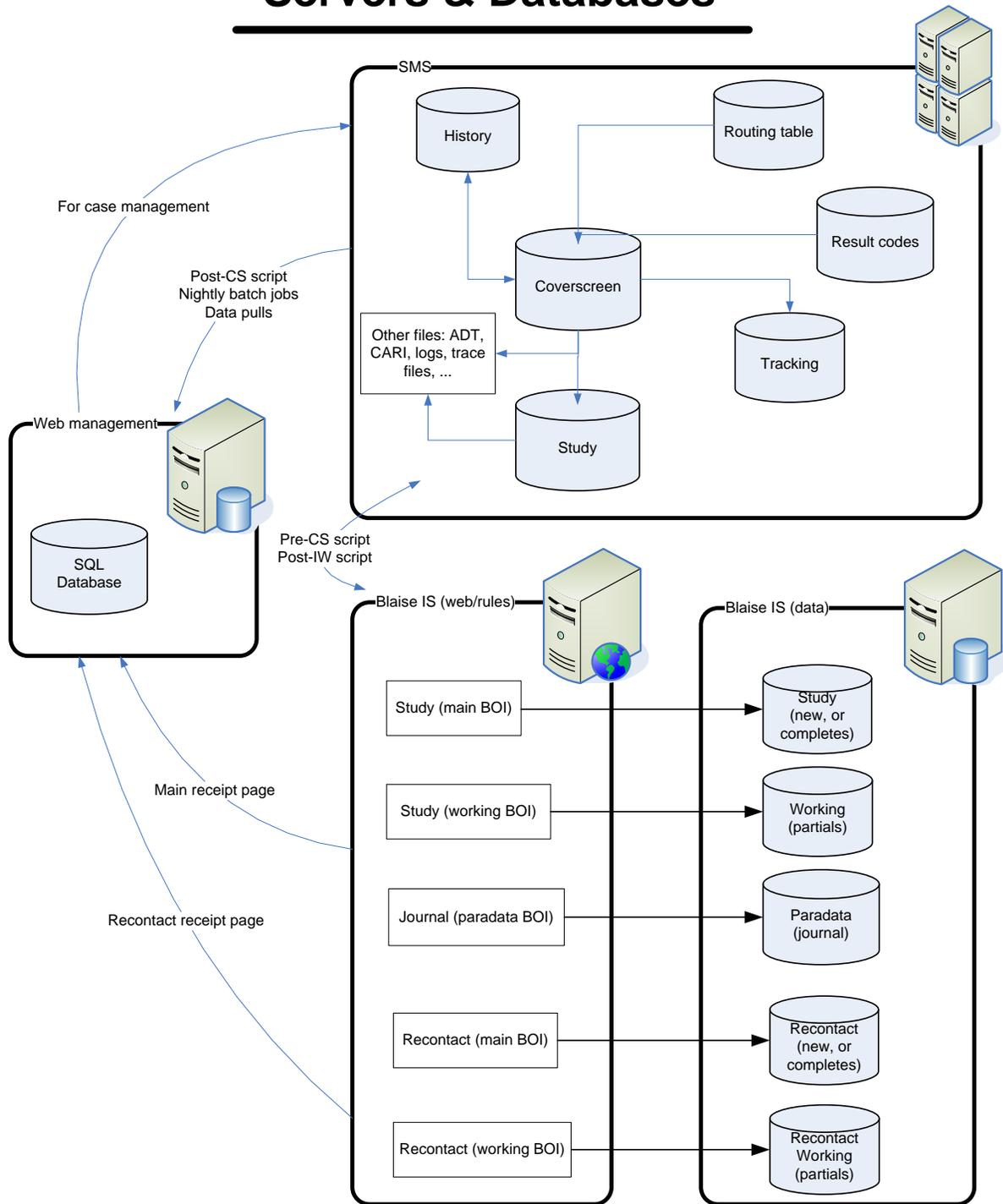
While web mode allows for many questions on the same page using grids, CATI standards are one question per screen. Hence, lead-in questions, optional text to read, interviewer instructions, question help, DK/RF, and comments have a very different look and feel between the modes. The table below compares some of these differences.

Criteria	SMS (CATI)	Blaise IS (web)
History search	Shows call history for the case	None
Help	Context sensitive	None
DK/RF	Enabled	Disabled
Comments	Enabled	Disabled
Navigation	Page up and down, arrow keys, home, end, enter key	Page up and down, arrow keys, space bar to select, Prev and Next buttons.
Questions per page	One question per page, entry in form pane.	Multiple questions per page, entry on page.
Series of questions	Lead in questions parenthesized in followups	Questions series presented in grids
Mode	lwer assisted interview	Respondent self-interview
Required	All questions are required	Respondent may skip any question

3.5 Data storage

Both CATI and web use BDBs to store the data collected. However, there are many differences between the CATI and web. SMS stores partial interviews in the same study database, while the web stores it in two data bases (main and working). Additionally, SMS also has history, tracking, a routing table (group/individual routing) and result code databases. Blaise IS stores respondent actions and pages visited in the paradata database (journaling), while SMS stores this information in ADT and CARI files. In addition, relative BOI files are created by Blaise IS when deploying the survey package to the production web server, which in turn point to the production data server.

Servers & Databases



3.6 Testing methods

Testing the mixed-mode system involved typical testing within the SMS and Blaise IS, as well as testing all of the interactions in the mixed mode. Scenarios were provided to guide testing. Both modes used a question in the survey to select which section(s) to test, including user-edited preload.

The following programs and utilities were used.

3.6.1 CAI Testing Tools (CTT)

CTT, a utility developed by the University of Michigan, provides an interface for group testing of a survey in both CATI and Blaise IS. It creates a local copy of the Blaise DEP and the survey, and stores tester's comments and cases on a networked data base. CTT also contains reports of outstanding items per user and project, priorities, data model date/time, screen shots, etc. The testing through CTT was not routed through the .Net login page.

3.6.2 Email invitation

An email invitation is sent out by the web management system and contains a hyperlink to the survey. The tester then clicks on the link to be taken to the login page for entry into the survey, just as the respondent would do.

Initially, test lines were loaded on the production server, and then cleared before production began. However, this meant once production started the test environment was no longer available. Later, a parallel project was added that duplicated the production survey in a safe manner.

3.6.3 Blaise Internet Server Manager

Programmers tended to use this method to check the survey before releasing it for testing via CTT. This did not use the .Net login page.

3.6.4 Control Centre

This was used by the programmer to test the CATI survey.

3.6.5 Survey Preview

For the Blaise IS survey used by the programmer, the preview mode in the Internet Specifications is a useful way to browse through all the pages of a web survey and look for obvious errors in pagination, layout, text enhancements, tables, and so forth. However, fills are not shown because there is no data associated with the case.

3.6.6 Modelib preview

This provides another way to look at all the pages of a CATI survey, with the same limitation for fills as the Blaise IS survey preview.

3.7 Security

Interviewers in SMS must log in to the server in order to work. They have limited access to folders within the server, related only to the surveys they are assigned. They then log in to SMS – utilizing their username/password for SMS (not the same as the server login). Only after this process can they start the interview.

For the web, the respondent must use the provided username/password. The authentication page queries a back-end database with this information and redirects the browser to the actual Blaise IS survey with the returned primary key. Username, password and primary key are all random character/digit combinations to minimize an attack by generating username/passwords.

3.8 Servers

There are a number of servers involved in the mixed mode design. Each has its own important role.

3.8.1 CATI

A separate production server is used that contains SMS, datamodels, and databases. The CATI data is stored separately than the web databases.

3.8.2 Web

Two servers are involved: web/rules server and a data server. The web server contains no data, but contains relative BOI files that are created by the Blaise Internet Server Manager that point to the production data server. Both servers also have the datamodels (study, working, paradata).

3.8.3 Web management

The SQL server database is stored on a separate secured server that is not directly accessed by any of the interviewers or respondents.

3.9 Switching modes

Important part of this is that the switching of the mode is driven by the CATI side; the web side is passive. All mode switches are initiated by batch routines, iwer actions, or supervisor actions.

Since it is the CATI mode controlling the mode switch, and SMS in control, additional Manipula routines are called at strategic points during the CATI interviewing process. One Manipula script updates the web management system database with call records, contact information, and key variables. Other Manipula scripts are responsible for setting flags in the one mode (i.e., web) to lock sections completed in the other mode (i.e., CATI) so that the respondent doesn't answer the same sections twice. Note: in case of a partial section completed, that section is restarted in the new mode.

3.9.1 Interview on demand

If a case has been assigned to web, and the respondent calls in using the toll-free phone number, then the need is to start the CATI interview immediately. This is accomplished by having all sample loaded in both web and CATI at the same time. The CATI interview may not be part of the current release, but is always available by its primary key or phone number.

The interviewer then conducts the survey as normal by going through the coverscreen and then the interview. Upon a suspended or completed case, a call record is written to both SMS and the web management system.

3.9.2 Flags for sections started/completed

To reduce respondent burden, any sections that are completed in a mode are flagged within the study data model (CATI/web). The flagged sections are then kept on the route in the other mode and are not asked. Key variables that are used in other sections of the survey are copied for those sections that are locked; this maintains the flow of the survey in the other mode. For example, if Block A is completed, and Field A within the block determines which of Block B, C, or D is asked, then Field A is a key variable that has to be passed to the other mode. Other fields within the section that locked are not copied.

Thus, suppose sections A and B were completed in web, and section C was partially done in web. Then the interview was switched to CATI. The interview then would start in section C. The data for sections A & B are only in the web interview, and there would be duplication for some fields in section C between web and CATI. Resolving conflicts between the data will be handled by analysts.

The script to determine what makes a completed section does a rules check in the data model (e.g., CATI), steps through each asked field on route, and keeps track of each block that has data. It also notes the last block with data, and keeps that off the list of blocks to flag. Once the list has been determined, flags in the other mode data model (e.g., web) are set, and the rules in the data model use the flags to determine to KEEP or ASK the section. It is possible to switch modes multiple times, and each mode then will have a unique part of the survey according to what was filled in. That means, CATI could have sections A, B, F, H, while web has sections C, D, E, G, and I completed. This scenario implies CATI: A, B; then web: C, D, E; then CATI: F; then web: G; then CATI: H; then web: I. There are possible overlaps in sections B, E, F, G, H, and I.

3.9.3 Manipula scripts – batch mode

Overnight scripts also move sample between modes. This script logs its actions to a SQL database, sets appropriate flags for holding/releasing the sample, and assigns randomized appointments according to a complex algorithm.

3.9.4 R logs onto web survey at will (mode start in CATI/web)

The respondent can complete the web survey at any time. The flags for the sections should always be up-to-date from CATI. The respondent can break off and resume the web survey. However, if the interview is switched back to CATI, the flags would then be set again before the CATI interview is initiated.

4. Conclusions

Any large project takes effort, and one with a mixed mode involved takes even more. The project as a whole has been successful.

4.1 Lessons learned

4.1.1 Mixed mode switch

The switching between modes was more difficult than expected in terms of how to manage the interview data between modes and keeping case status updated in the different systems (SMS and the web management system). The breakthrough was the realization that one system, SMS, needed to be in control of the status of all cases. That led to creating Manipula scripts that set flags for locking completed sections, and for updating the status in the systems.

4.1.2 Autoindex + BOI files

The OLEDB manager wizards do a very good job of generating a data model and a matching BOI file. However, when the SQL table contains an autoindex column as a primary key, inserting records from Blaise is problematic. The solution was to use OLEDB manager and manually select the fields to create in the BOI file, excluding the autoindex column. Inserts from Blaise then work seamlessly.

4.1.3 Testing

As has been said many times before, the more time that is planned in testing yields greater confidence, data quality and fewer changes during production. As much as was desired, more time for testing was desired.

A parallel test project had not been set up initially on the production server, but instead test sample was loaded and removed once production started. If it had remained, then potentially test data could have been introduced into production sample. This meant that true testing of new releases in a production environment could not happen until a parallel test project had been established.

4.1.4 Manipulus flag setting routines

We used the Manipula meta information methods and recursion in the scripts to be able to accurately set flags in the CATI/web surveys. The point is that a survey taken in one mode had to update the other mode's survey flags.

4.1.5 Priorities

It was difficult to balance work between programming the surveys in CATI & web and working on systems development. Typically, the survey is more important – what good is it to collect bad data? In hindsight, it would have been better to work on these parts in tandem since there were system requirements that affected the data model design.

4.1.6 SMS

Setting up the project for mixed mode also required modification of the CATI sample management system. SMS had to separate from other normal CATI projects because supervisors who conducted an interview from the management program, SMSSuper, also ran the pre-coversheet and post-interview scripts to set flags.

Setting up new users in this system took extra effort because of drive mappings, dedicated computers with equipment for CARI, installed DLLs, and checking to see that all parts of the mode switch functioned.

4.1.7 Web receipt page

The receipt pages are responsible for transferring data from the main interview to a recontact interview as preload. Creating preload on-the-fly between two Blaise IS surveys will allow us to create complex new Blaise IS surveys in the future. The same receipt pages also stored key variables to the web management system via stored procedure calls written in ASP.

4.1.8 Development

Using prepare directives to produce testing & production datamodels made it easy to ensure all parts of the survey were compatible. Keeping the builds of Blaise versions current between servers was also key in reducing headaches.

4.1.9 BOI data to SQL

We discovered that user entered data that contained apostrophes, DK, or RF caused errors when being written to an existing SQL table. The single quote within a string field was interpreted by SQL as ending the string, and the rest of the string was being interpreted as SQL commands. This is the essence of a SQL injection attack. This was solved by escaping all single quotes to two single quotes.

DK and RF also were transformed by Blaise into a field of 9..9's and 9..8's. We rewrote the Manipula routine to customize the values so they fit into the data column.

4.1.10 Slow access

We found invalid references within Manipula scripts to BOI files caused a Windows seek & wait, and really slowed the execution of the script to a crawl. This happened when we moved from the testing server to the production server. Correcting the server reference solved this problem.

4.1.11 Testing documentation

Programmers and testers actually used two different specification documents. The programmer document lagged behind the testing document, and as a result true bugs were reported that did not match current programming document. In the future, one common document should be used for both.

4.2 Future directions

Updates to the SMS are necessary in order to make it easier to work with mixed mode surveys. These include adding a "mode" variable to the coverscreen, moving interviewer and contact observation variables to an external database, removing inefficiencies, maintenance, and security problems with the current system, such as shared text files.

We plan to create a "gold standard" project to use when starting a new mixed mode survey, and also have a library of groups for complex pages to help standardize the web surveys.

4.3 Insights

We want to share these insights in developing a mixed mode system:

- Spend more time reading through all the Blaise help related to mixed mode, Blaise IS, and data transferring routines before programming starts. We found that we were "going back to school" throughout the project and finding better ways of performing tasks, and sometimes rewrote code to take advantage of our new knowledge.
- With limited time constraints, we recommend producing parts of the management system as feasible. It still is very important to have a well-formed survey for data collection, but the systems have to be developed at the same time in an iterative approach.
- There were only two programmers, with temporary assistance from a third, in programming the CATI and web surveys and the SMS/Blaise IS management systems. One other programmer developed the SQL web management system. It would have been extremely helpful if there had been a programmer for each of the parts: one for CATI, one for web, one for SMS/Blaise IS management systems, and one for the SQL web management system.
- And finally, for all those non-programmers: Meetings are most useful when they are short and concise as this will allow more time for the actual work than talking about the work.

5. Footnotes

¹ SurveyTrak is the University of Michigan case management system for distributed sample, i.e., field data collection.

² Coverscreen datamodel: see 1.3

³ Study datamodel: The actual survey itself without any management piece embedded in the survey. There are standard fields, such as Complete, that are required for SMS and SurveyTrak.

⁴ Result codes are call dispositions, such as an answering machine with a message left is code 1402.

⁵ Daybatch sub-priority: a more recent feature of Blaise CATI, is a sub priority code within a result code to influence the delivery of a case.

Multimode Rental Survey In Norway.

Jan Haslund, Statistics Norway

1 Introduction

In Statistic Norway we use a case management system called SIV. This is developed in Java and uses an Oracle database for storing data. In this system we don't store questionnaire data. The data is stored in a Blaise database. This database is used for CATI interviewing. When an interview attempt is finished we need to update the Oracle database with statuses (interview or non response), new addresses, new names, new telephone numbers etc. To do this we write to a BOI file that points to an Oracle table every time we finish an interview and return to the dialscreen. We use the ondialend event in CATI to do this. We write the status of the telephone attempt (interview, non-response) and if there is any change in the name, address, telephone number. If there is a change, SIV use the Blaise API to retrieve the new information. If someone changes the information in SIV it will be synchronised to the Blaise database using the Blaise API.

Statistics Norway collects data every year on housing and rent in private households throughout Norway. The purpose of the Housing and rent survey is to compile official statistics on rent levels in Norway for different types of dwellings and geographical areas. The questions concern monthly rents and dwelling characteristics. The size of the gross sample in the Housing and rent survey is 22 000 addresses. From the answers on the Housing and rent survey we select 2500 dwellings for The Rental survey.

The Rental survey is a monthly survey with twelve waves starting in January. The first wave is a telephone interview. In this interview we collect data about the rent for January and who owns the dwelling. These questions are for finding new tenants if the person we have interviewed moves from the dwelling and a new tenant move in. With this information we can ask the owner of the dwelling who lives there. This information we use for the whole year. We also ask for an e-mail address we can use to reach the tenant.

Wave two to twelve is web and CATI. We start the first Monday in the month with web interviewing. We send an e-mail to the respondents to whom we have e-mail address and a SMS to whom we have cellphone number but no e-mail address. The e-mail includes a link to the questionnaire. Thursday we follow up the respondents who have not answered on line. This time we send e-mail and SMS to all. After a week, the next Monday, CATI interviewing starts for the rest of the respondents. The web questionnaire is still open. Every morning we get the web answers and synchronize them to the CATI database and to SIV. The CATI interviewing goes on for one week. When interviewing has finished we make the final files. From the start of data collection to the end of the wave it takes 2 weeks. We have about 900 answers on web each month and about 850 CATI.

For the web questionnaire we used Blaise IS. We know that some people get e-mail on cellphones and tablets and that the Blaise IS questionnaires are not always easy to use on these devices. At the Blaise Conference in London 2012 we saw a presentation of C-Moto from CentERdata. They showed some nice looking screenshots from cellphones and tablets. Due to this we decided to make a C-Moto version of the Blaise IS questionnaire and in May 2013 we put it in production. The first month we used it for the respondents to whom we send SMS. We saw that only a few of them answered on-line so we thought we could test it on them. In June we sent two links in the e-mails. In the future we may only use the C-Moto version because it works well on all devices - also laptops.



Figure 1. Question in Blaise IS

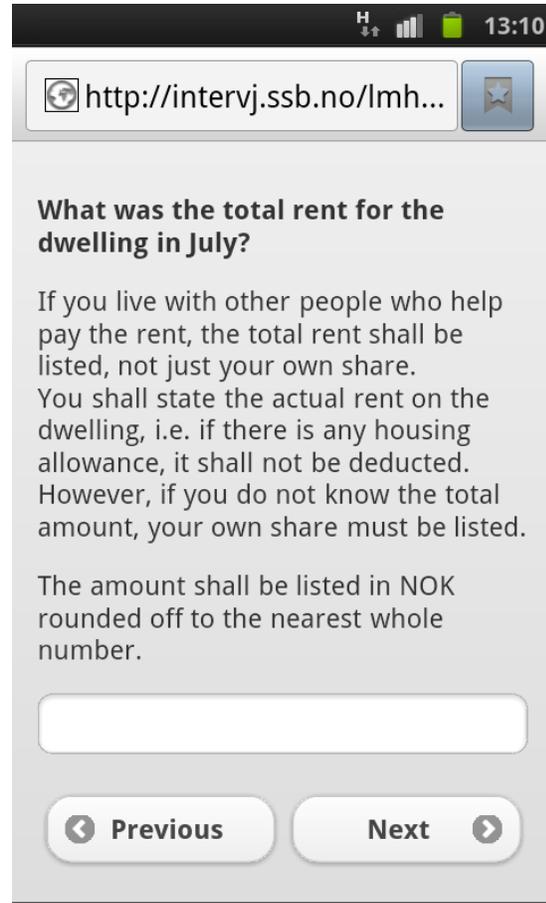


Figure 2 Question in C-Moto

Here are screenshots from a cellphone. As you can see on the Blaise IS questionnaire the text is small and difficult to read. The buttons are small and difficult to click on. The C-Moto version is easier to read and the buttons are bigger so you can easily click them.

2 The questionnaire

The questionnaire is short. First we ask if the respondent still lives on the address.

- If he or she does not, we ask if he or she knows who is living there now, and collect the name and telephone number for the new tenant. Then we try to interview the new tenant with CATI.
- If he or she still lives in the dwelling we inquire about the rent. If the rent has changed (more than 20%) we ask why the rent has changed. Further we ask if the rent includes parking space, heating and electricity; if he or she rents the dwelling with furniture; if he or she has an agreement with the landlord to carry out certain services, e.g. clearing snow or babysitting in addition to paying the rent?

At the end we ask if he or she will be living at the address next month.

- If he or she does not, we ask if he or she knows who will live there next month and collect the name and telephone number for the new tenant.
- If he or she would live there we ask if the e-mail address and phone number are correct and change them if there are any changes.

More than 95% of the respondents only answer how much they pay in rent and how we can contact them next month.

3 Tasks during the survey period

There is a lot of tasks to do before, during and after the survey period. The tasks are the same every month.

- Create questionnaire in SIV
- Prepare sample
 - Get answers from last month
 - Prepare data files for loading into SIV (Oracle database), Blaise WEB
 - Select respondents for e-mail, SMS or CATI
 - All respondents are loaded into all files (SIV (Oracle database), Blaise WEB)
- Change WEB questionnaire with data model name, wave etc. for both the Blaise IS and the C-Moto version of the questionnaire
- Change Bis file and make new Bip file
- Install questionnaire on the web server
- Load sample into SIV and export it (We use this to get a unique id for the respondents)
- Change and prepare CATI questionnaire
- Load the sample into the CATI questionnaire
- Mark the cases we shall not phone (refused more than 2 times, respondents with no telephone number, housing that is not rental properties anymore)
- Synchronize from Blaise to SIV
- Send e-mail to the respondents
- Send SMS to the respondents

- Daily
 - Retrieve answers from the WEB server
 - Put the answers into the CATI questionnaire (only status fields) to make them completed.
We do not want to call when we do CATI
 - Synchronize form from Blaise to SIV
 - Make reports
- Send reminders to those who have not responded
- Create files to client

4 How we do it

As you can see, there is a lot of tasks that have to be done on a regular basis. To help us remember what to do and when to do it we made a checklist using AutoIt.

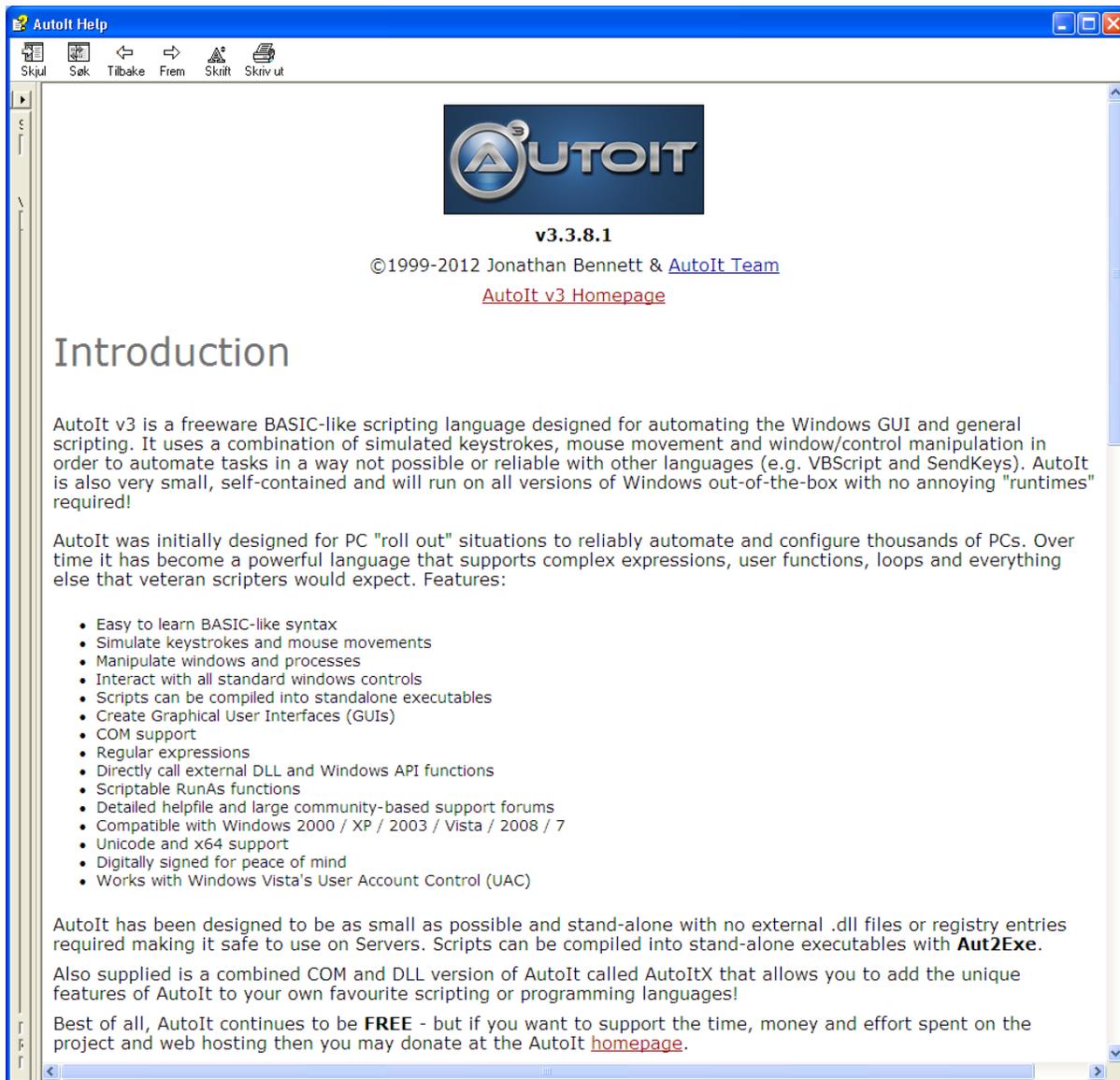


Figure 3 AutoIt introduction

Here is an example of the checklist.

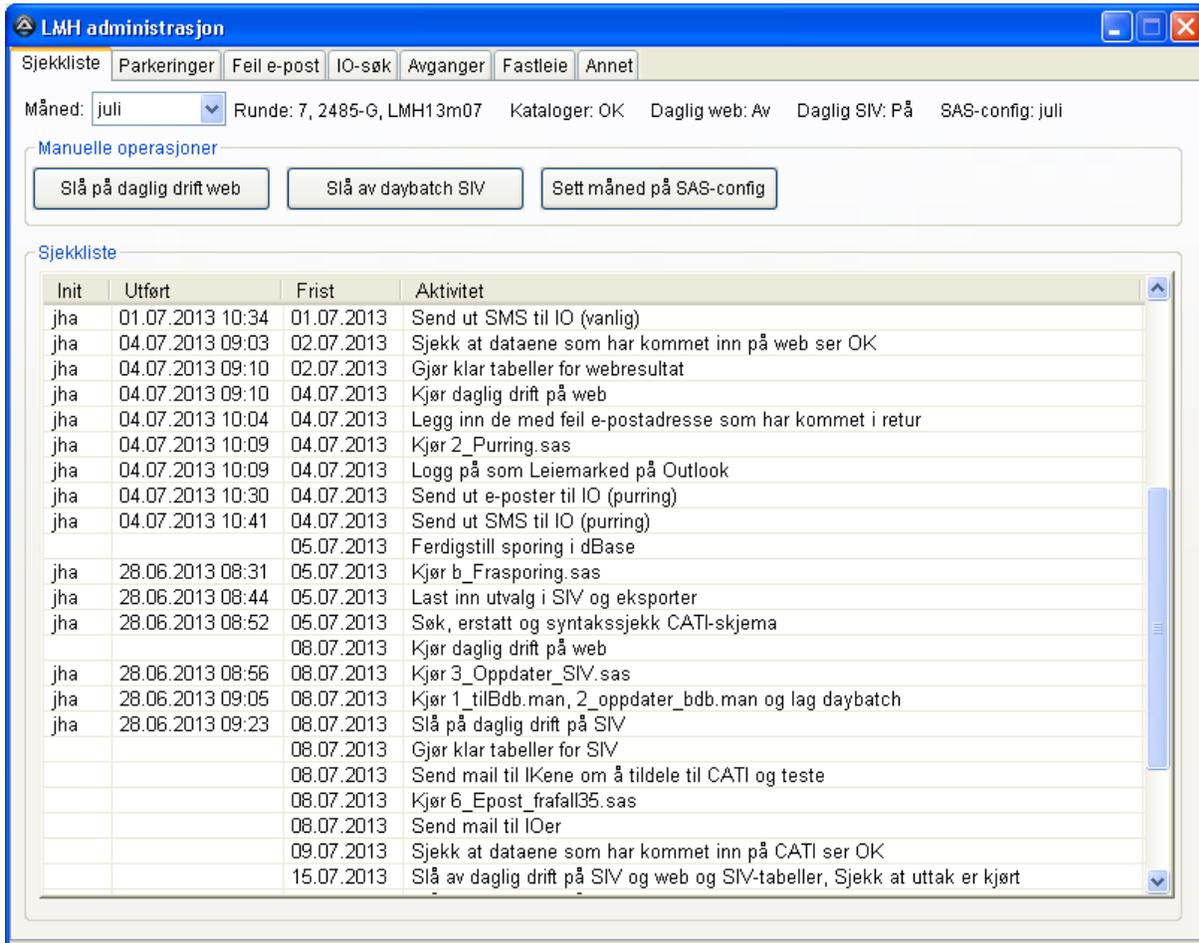


Figure 4 main window of the checklist

The Checklist contains activity, the deadline for the activity, who has done it and when it was done. In this way we make sure that everything is done at the right time. The program read a tab divided ascii file so it is easy to change it, make new lines or delete lines. Because the activity is logged with date, time and who did the activity it is easy for another person to step in and do the next activity. That mean it is easier to share the work between several people. We also have a sort of documentation of what we have done. If we double-click on an activity we get a dialog box where we can choose to do the activity automatically or manually. We can also delete the registration of who has done the activity and when it has been done. We can also close the dialog box.

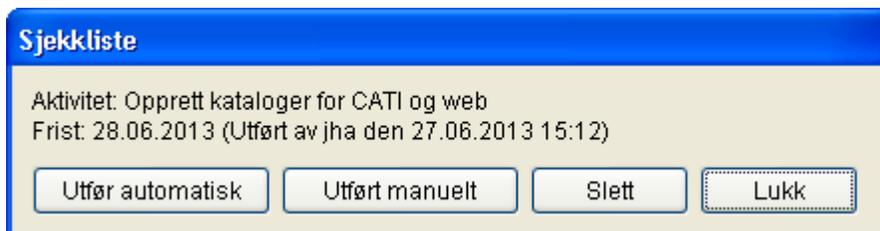


Figure 5 Activity in the checklist

For most of the activities we have made programs in SAS or Manipula. From AutoIt we can start other programs. If we click the button for doing the activity automatically it runs a SAS or manipula program. We can also manipulate text files in AutoIt. Using this we can generate a new data model and new manipula programs when we start a new wave. We copy the Blaise source files from the previous month to a new folder and rename the bla file. We read the new source files line by line and do some changes such as the name of the questionnaire, input and output files in Manipula wave etc. automatically. Most usually we change the month number from LMH13m06 to LMH13m07; or the wave which is the same as the month number. When the source files are changed we prepare the data model and the Manipula scripts.

```

; Perform search and replace on a manipula (.man) file and syntax checking it afterwards
; Parameters:
; $path = Path
; $file = Name of the manipula file (.man)
func updateManfile($path, $file)
    local $lines, $i
    _FileReadToArray($path & $file, $lines)
    if(@error) then return -1
    _ArrayDelete($lines, 0)
    for $i = 0 to UBound($lines) - 1
        $lines[$i] = StringRegExpReplace($lines[$i], "LMH\d{2}m\d{2}", $monthCurr)
    next
    _FileWriteFromArray($path & $file, $lines)
    RunWait($BlaiseParser & " " & $path & $file, $path)
endfunc

```

Figure 6 change and prepare a Manipula setup

In connection with the SAS programs we use, we have a configuration file with the variables that change from wave to wave. We create the configuration file from AutoIt.

We also have AutoIt scripts for sending e-mail and SMS, we just select the right file, which is created automatically from the SAS programs. From the name of the file we select, the script selects the right e-mail template for the SMS. Both the e-mail and SMS have a link with user-id and password. The SMS has a link to the C-Moto version of the questionnaire. The e-mail has a link to the Blaise IS version and a link to the C-Moto version.

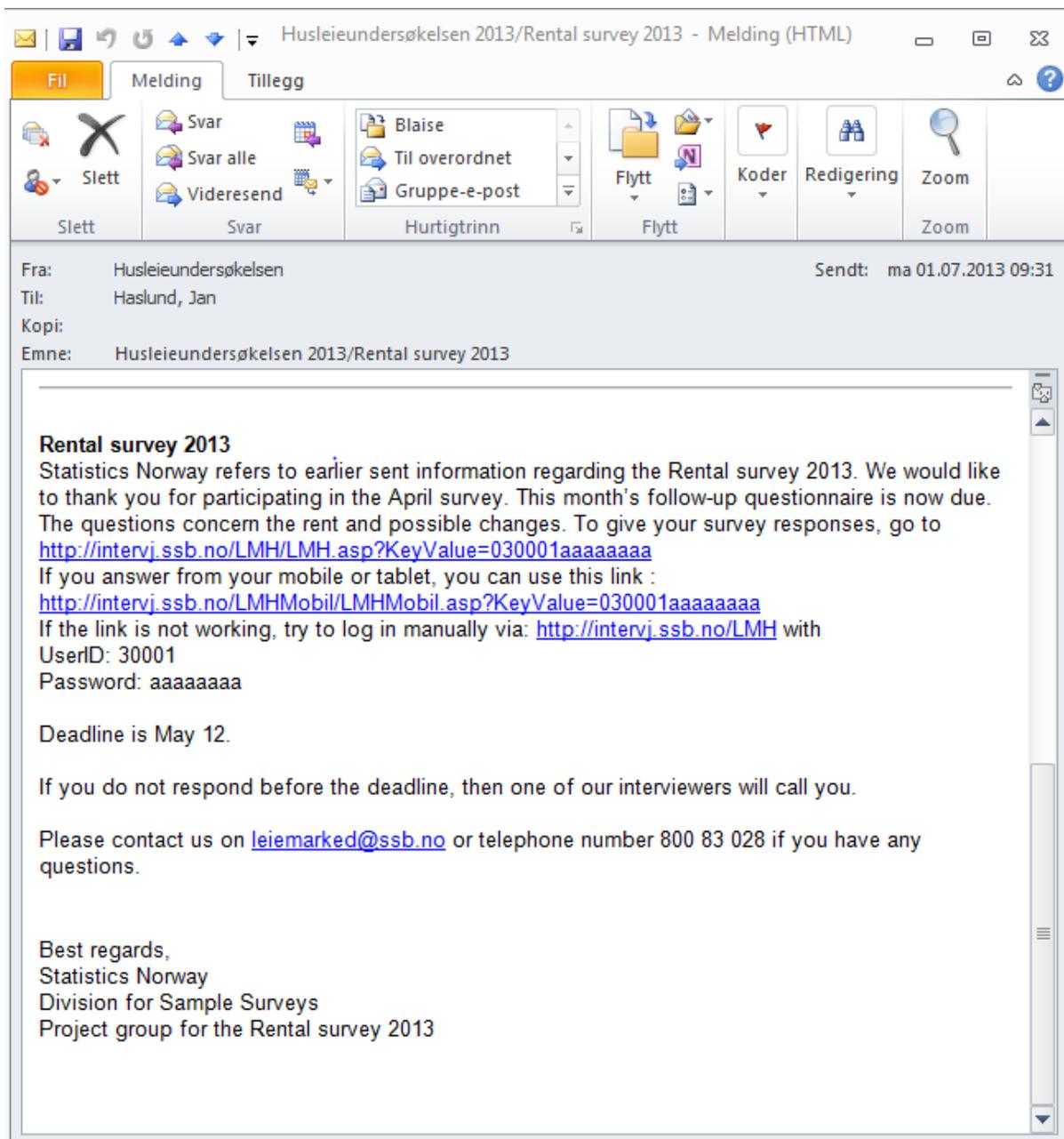


Figure 7 E-mail to respondents

Every morning we have to synchronize the web answers to both the CATI database and to SIV. We also read all questionnaire data into SAS datasets. From the SAS dataset we make an ascii-file of the new answers. This file is read into the Blaise CATI file by a Manipula script. The Script put some statuses to the Blaise file, including the date for today in the field `catimana.caticall.firstday`.

```

Blaise 4.8 Control Centre - (Untitled)
File Edit Project Run Database Tools Add-Ins Window Help

\\Kpwi-blaise01\Siv\Skjema\LMH13m07\W01\Produksjon\3_oppdater_bdb_daglig.man

USES
  Outputmeta 'LMH13m07'
  DATAMODEL InputMeta
    FIELDS
      IO_Nummer      : 0..999990
      Innled         : 1..4
      FrafGr        : 11..41
      AvgGr         : 95..98
      Status_case   : string[2]
      Intslutt      : 1..3
      Intervjuer    : string[3]
      Melding       : string[80]
    ENDMODEL

  UPDATEFILE Outputfile: Outputmeta ('LMH13m07', blaise)
  INPUTFILE Inputfile: Inputmeta ('..\..\Utvalg\oppdater_bdb.txt', ASCII)
  SETTINGS
    LINKFIELDS
      IO_number = Outputfile.IO_number

  AUXFIELDS
    nokkel : STRING[200]
    Kommando : STRING[400]
    Reslt : INTEGER
    intervjustatus : string[2]

  MANIPULATE
    if(Inputfile.search(Outputfile.IO_number)) then
      Inputfile.read
      Outputfile.Innled := Inputfile.Innled
      Outputfile.FrafGr := Inputfile.FrafGr
      Outputfile.AvgGr := Inputfile.AvgGr
      Outputfile.Status_case := Inputfile.Status_case
      Outputfile.Intslutt := Inputfile.Intslutt
      Outputfile.Fraf$pes := Inputfile.Melding

      Outputfile.Intervjuer:=Inputfile.Intervjuer
      Outputfile.CatiMana.CatiCall.NrOfCall:=1
      Outputfile.CatiMana.CatiCall.FirstDay:=SYSDATE
      Outputfile.CatiMana.CatiCall.RegCalls[1].Whomade:=Inputfile.Intervjuer
      Outputfile.CatiMana.CatiCall.RegCalls[1].Daynumber:=1
      Outputfile.CatiMana.CatiCall.RegCalls[1].DialTime:=SYSTIME
      Outputfile.CatiMana.CatiCall.RegCalls[1].NrOFDials:=1
      Outputfile.CatiMana.CatiCall.RegCalls[1].DialResult:=Completed

      Outputfile.Endring := 1
      Outputfile.WRITE
    endif
  ENDMANIPULATE

```

Figure 8 Updating CATI file script

When we have the data in the CATI database we have to synchronize it to SIV. To do this we use a Manipula script that loops through the Blaise file if it is a complete interview and it is not synchronized earlier it call another manipula script that write to a BOI file that point to an Oracle table in SIV.

```

process Retttattut
SETTINGS
  DESCRIPTION = 'BLAISE to BLAISE'

USES
  InnMeta 'LMH13m07'

UpdateFILE utfil: InnMeta ('LMH13m07', blaise)

AUXFIELDS
  nokkel : STRING[200]
  Kommando : STRING[400]
  Reslt : INTEGER
  intervjustatus : string[2]

MANIPULATE
  utfil.open
  utfil.reset
  utfil.READNEXT

  WHILE NOT(utfil.eof) DO
    IF ((CatiMana.CatiCall.FirstDay=SYSDATE) AND (Intervjuer = 'WEB')) THEN
      IF (utfil.status_case=RESPONSE) THEN
        {Lager intervjustatus}
        if (utfil.status_case= '00') then {Intervju}
          intervjustatus := '0'
        elseif (utfil.status_case= '02') then {Frafall}
          intervjustatus := utfil.GETVALUE('fracgr',UF)
        elseif (utfil.status_case= '04') then {Overfering}
          intervjustatus := utfil.GETVALUE('ovfgr',UF)
        elseif (utfil.status_case= '01') then {Avgang}
          intervjustatus := utfil.GETVALUE('avggr',UF)
        endif
        {Parametre til skriv_status2. Se skriv_status2.man for dokumentasjon}
        nokkel:= str(utfil.io_idnr)+';'+utfil.status_case+';'+intervjustatus+';'+'''+
          utfil.frafspe+';'+'''+'+utfil.intervjuer+';'+str(utfil.endring)+';'+'''+0'
        {Kjører skriv_status2 som oppdaterer blaise_event_inn}
        Kommando:='\kpw-blaise01\siv\faste\skriv_status2 /Q /P'+nokkel
        {DISPLAY(kommando,wait)}
        Reslt := CALL(Kommando)
      ENDIF
    ENDIF
    utfil.READNEXT
  ENDWHILE
  
```

Figure 9 Synchronizing to SIV

```

Blaise 4.8 Control Centre - (Untitled)
File Edit Project Run Database Tools Add-Ins Window Help

PROCESS skriv_status2 "Skriv status til blaise_event_inn"

USES
  bei      '\\kpwi-blaise01\siv\faste\blaise_event_inn'
  status   '\\kpwi-blaise01\siv\faste\status'

  {updatefile beif : bei      ('blaise_event_inn.boi', OLEDB)}

  outputfile beif : bei      ('\\kpwi-blaise01\siv\faste\blaise_event_inn.boi', OLEDB)
  outputfile statusf : status ('\\kpwi-blaise01\siv\faste\status', blaise)
  settings
    makenewfile = no

AUXFIELDS

  io_idnr      : INTEGER[6]      {I/OID nummer}
  status_case  : STRING[2]      {status_case}
  intervju_status : string[2]    {intervjustatus}
  frafspes     : string[200]    {kommentar}
  intervjuer   : string[3]      {intervjuer}
  Endra        : integer[1]     {Kontaktinfo endret}
  daybatchkode : string[2]
  id           : integer[6]     {test}
  skrivstatus  : integer[1]     {skrive status til status for AKU}
  internstatus : string[1]      {Skjemaets internstatus}
  tlf1,
  tlf2,
  tlf3        : string[9]       {skrive telefonnummer til status for AKU}

MANIPULATE

  io_idnr      := val(PARAMETER(1))
  status_case  := PARAMETER(2)
  intervju_status := parameter(3)
  frafspes     := parameter(4)
  intervjuer   := parameter(5)
  Endra        := val(PARAMETER(6))
  daybatchkode := parameter(7)
  id           := val(parameter(8)) {test}
  skrivstatus  := val(parameter(9))
  internstatus := parameter(10)
  tlf1         := parameter(11)
  tlf2         := parameter(12)
  tlf3         := parameter(13)

  {Variabler som skal inn i blaise_event_inn for alle skjema}
  beif.INTERVJU_OBJEKT_ID := io_idnr
  beif.status_case       := status_case
  beif.intervju_status   := intervju_status
  beif.kommentar         := frafspes
  beif.initialer         := intervjuer
  beif.ENDRING_KONTAKT_INFO := Endra
  beif.DAY_BATCH_KODE    := daybatchkode
  beif.BEHANDLET         := 0
  beif.id                := id {test}
  beif.intern_status     := internstatus
  beif.write

```

Figure 10 Synchronizing to SIV

5 Conclusions

Using the checklist gives us control of the tasks and secure that the tasks are done at the right time. Running programs automatically secure that we will do the same every wave.

It seems that the C-Moto version of the questionnaire works fine so we are thinking of using it for all the web respondents.

Reference

Alerk amin, Arnaud Wijnant (CenERdata, Tilburg, The Netherlands) “Blaise On-the-Go Using Blaise IS With Mobile Devices”, Proceedings of the 14th International Blaise Users Conference, April 2012

Customizing the BlaiseIS XSLT Stylesheets and ASP Scripts

Edwin de Vet (CentERdata, Tilburg, The Netherlands)

Arnaud Wijnant (CentERdata, Tilburg, The Netherlands)

1 Abstract

At CentERdata we routinely use BlaiseIS to administer questionnaires over the web to a large number of respondents. We often have to customize the BlaiseIS system to meet the requirements of our clients. These customizations almost always involve modifying the XSLT stylesheets that are used for rendering the HTML pages. This paper contains examples of how we extended the capabilities of BlaiseIS. These examples include incorporating a slider using jQuery, using the HTML tag “label” with checkboxes and radio buttons to make the text clickable, styling of tables etc. Besides modifying the existing BlaiseIS stylesheet, we created also a lightweight XSLT stylesheet from scratch. This stylesheet creates simple, lean HTML while still providing the functionality we need. We found that this stylesheet solves some performance issues we encountered with large tables. Furthermore, it served as a basis for our stylesheet for mobile devices. We also modified ASP scripts to add functionality. Examples are the key stroke logs in text format introduced in the page handler and a simple security check using the SHA1 algorithm in the interview starter.

2 Introduction

At our institute we run web questionnaires using BlaiseIS for a large number of clients. Since our migration to BlaiseIS in 2010, we gained a lot of experience in adapting the system to our needs and the needs of our customers. These modifications to BlaiseIS almost always involve adapting the XSLT style sheets and/or the asp scripts. This paper will give a broad overview of those modifications. We run Blaise 4.8.2 on our servers.

3 XSLT Style Sheets

3.1 Clickable text

We use the HTML tag “label” for text belonging to checkboxes and radio buttons. This has the advantage that respondents can select or deselect those elements by clicking on the text. Previously, we added an Id to the HTML input elements of type radio and checkbox. This Id is the same as the Id of the corresponding CategoryControl element in the XML used as source for the transformation. The template RichTextElement (in biSimpleHTMLWebPage.xsl) was modified so that it adds a label around the text with a “for” attribute that references the corresponding checkbox/radio button. This is a simplified example of the HTML:

```
<input name="qulfpala" id="qulfpala1" value="1" type="radio">
<label for="qulfpala1">Option A</label>
<input name="qulfpala" id="qulfpala2" value="2" type="radio">
<label for="qulfpala2">Option B</label>
```

3.2 Slider

To create sliders in BlaiseIS questionnaires, we used the jQuery-UI library. The following code was put in the Blaise source file:

```

<div id=""slider2""></div>
<script type=""text/javascript"">
  $(function() {
    createSlider(2);
  });
</script>

```

The XSLT style sheet was modified so that it included the jquery-1.3.2.min.js and jquery-ui-1.7.2.custom.min.js libraries, a CSS file for the slider and the JavaScript code for the createSlider function:

```

function createSlider(id) {
  var element = document.getElementById("qu" + id + "_id");
  element.style.display = 'none';
  var val = 550;
  if (element.value != '') {
    val = parseInt(element.value);
  } else {
    element.value = val;
  }
  var sliderOpts = {
    min: 100,
    max: 1000,
    value: val,
    slide: function(e, ui) {
      element.value = ui.value;
    }
  };
  $("#slider" + id).slider(sliderOpts);
}

```

This function creates a slider with values between 100 and 1000 and a default value of 550 (unless the answer is already present in the form). In the Blaise source file, the data type is integer. The createSlider function makes the open text field invisible to the user. By moving the slider, this hidden element is filled with the set value of the slider and once the form is submitted, the selected value is stored in the Blaise database.

De orde in ons land handhaven.

Helemaal niet belangrijk 1 2 3 4 5 6 7 8 9 10 Heel erg belangrijk



Figure 1: jQuery slider in a BlaiseIS questionnaire.

3.3 Table Styling

In our questionnaire we use a lot of tables/grouped questions. The XSLT style sheet was modified in such a way that the header row (<tr> HTML element) always gets the CSS class “HeaderRow”. The other

rows in the table (containing the questions) get alternating class “OddRow” and “EvenRow”. Standard CSS can be used to give each type of row its special style.

	1	2	3	4	5
Vond u het moeilijk om de vragen te beantwoorden?	<input type="radio"/>				
Vond u de vragen duidelijk?	<input type="radio"/>				
Heeft de vragenlijst u aan het denken gezet?	<input type="radio"/>				
Vond u het onderwerp interessant?	<input type="radio"/>				
Vond u het plezierig om de vragen in te vullen?	<input type="radio"/>				

Figure 2: Example of table styling using the “HeaderRow”, “OddRow” and “EvenRow” CSS classes.

3.4 Simple Style Sheet / Mobile Style Sheet

At CentERdata we also created a minimal XSLT style sheet that only has the functionality we require. This style sheet is roughly 6 times smaller than the original style sheets, renders much faster and is easier to modify due to its simplicity [1]. This style sheet formed the basis for a style sheet for mobile devices and is documented extensively elsewhere [2].

3.5 Custom questions

Adding interactive questions can enhance the respondents’ experience. For this, we created a mechanism to easily integrate web applications in a questionnaire programmed in Blaise[3]. This opens up the opportunity to make use of all the options the web has to offer for responding to a questionnaire item. Examples of this are special interfaces (like auto complete boxes and even games), interactive feedback to respondents, and the use of other resources available on the web.

3.6 Trigram search/encode question

One of the custom questions that we created is an automatic look-up and encode question. This question uses a database that contains descriptions in which a respondent can search for items and attached to that a code that will be used for the answer in the Blaise questionnaire.

Respondents that visit this question can simply start typing their answer and then select the correct answer from a list of possibilities that is very similar to a Google search suggestion box. Once the respondent has selected the answer, the application encodes the answer automatically on the background and the routing of the questionnaire can be based on this code.

Please select a school.

gree

- Greenbrae School, Greenbrae Crescent, Bridge Of Don, Aberdeen, AB23 8NJ
- Udney Green School, Udney Green, Ellon, Aberdeenshire, AB41 7RS
- Timmergreens Primary School, Emislaw Drive, Arbroath, DD11 2HJ
- Greenmill Primary School, 2 Barrhill Road, Cumnock, KA18 1PG
- Castlevew Primary School, 2d Greendykes Road, Edinburgh, EH16 4DP
- Balgreen Primary School, 171 Balgreen Road, Edinburgh, EH11 3AT
- Juniper Green Primary School, 20 Baberton Mains Wynd, Edinburgh, EH14 3EE
- Parsons Green Primary School, Meadowfield Drive, Edinburgh, EH8 7LU
- Pentland Primary School, 10 Oxgangs Green, Edinburgh, EH13 9JF
- Castlebrae Community High School, 2a Greendykes Road, Edinburgh, EH16 4DP
- Thorntree Primary School, 55 Cobinshaw Street, Greenfield, Glasgow, G32 6XL
- Greenview Learning Centre, 165 Glenhead Street, Glasgow, G22 6DJ

Figure3: Example of the use of a trigram search in a Blaise web questionnaire.

To make this application more portable, the code is written in Classic ASP and uses an Access database. This makes it possible to integrate this application in the .bis package of Blaise and run it without any extra server requirements.

4 ASP Scripts

4.1 Logging to a text file

We modified the page handler script (BiPagHan.asp) so that it logs every form submit in a plain text file. The following items are logged: name of the questionnaire, timestamp (date and time), key value of the respondent, question name and given answer. Such a log allows us to analyze in detail the time interval between given answers and whether respondents tend to go back and forth in the questionnaire.

4.2 Security check

In our institute, respondents normally start a BlaiseIS questionnaire by clicking on a link or posting a form. This link (or form) normally contains the ID of the respondent (KeyValue) as either a GET or POST parameter. This parameter can be manipulated by the respondent. Our simple and robust solution is the addition of a hash value of the KeyValue concatenated with a salt string (normally bigger than 20 characters) in the link (or form). We use the SHA1 algorithm to create this hash. In the BiInterviewerStarter.asp, we added code that recalculates this hash value. Respondents only get a valid interview session when the recalculated hash is identical to the submitted hash. [1]

5 References

[1] Arnaud Wijnant and Edwin de Vet, Performance and Security Enhancements on the Blaise IS standard stylesheet; Proceedings of the 14th International Blaise Users Conference IBUC 2012, p232-236.

[2] Alerk Amin and Arnaud Wijnant, Blaise-On-The-Go: Using Blaise IS with mobile devices; Proceedings of the 14th International Blaise Users Conference IBUC 2012, p237-248 and www.centerdata.nl/link/cmoto.

[3] Arnaud Wijnant and Maurice Martens, C3B: Exploiting the numerous possibilities web technology offers to elevate questions; Proceedings of the 13th International Blaise Users Conference, Baltimore, Maryland USA, October 19-21, 2010, Vol. 13, p.94-105

Jumping around in Blaise IS

Maurice Martens, CentERdata

1. Introduction

In recent years CentERdata has developed several Life History Calendars (LHC). A LHC is an interactive scaled calendar representation on which annotated life events are visually marked by dates or years of occurrences. For CAPI projects these calendars have been developed using the Blaise API combined with a Visual Basic (VB) shell in order to represent the calendar interface. For a web version we scripted it using PHP and JavaScript, independent of Blaise.

Recently CentERdata was asked to support a study by providing such a LHC. This study was conducted by doing two hour face to face interviews with respondents with a criminal background. As the development team of CentERdata, we considered reusing the VB version, but since the laptops that were used for conducting the interviews had an internet connection available, and since preventing any loss of data was very important for this hard to reach group, we decided to try to implement a web survey that runs the CAPI over the web using Blaise IS.

The questionnaire with a LHC has some specific properties. It should always be possible to jump directly back to earlier questions. The calendar is a representation of events in a person's life, it should always be visible. The questionnaire consists of a series of large loops over events that persons would have had in their lives. This generated a large number of fields, pages and tables, which caused the questionnaire to slow down the questionnaire immensely. To solve this problem and to speed the questionnaire up we implemented some locks in the code to make sure the forward route checking would not load the complete questionnaire in memory.

Initially we tried to generate the html code of the calendar in Blaise, but due to memory constraints we ended up creating an independent service (using PHP) to generate the html code that displayed the calendar which was loaded inline using asynchronous calls. JavaScript would then trigger client side visualizations and actions performed on the presented calendar.

Based on these experiences a more generic tool was developed that can be included in any Blaise IS questionnaire. This tool builds a history of the questionnaire fields visited. It allows you to jump freely through Blaise IS questionnaires. For each field a status can be set and remarks can be made, giving us the basic features of an online testing app for Blaise IS questionnaires.

2. Online life history calendar

A Life History Calendar refers to a survey method in which events of a respondent's life are visualized in a table structure. CentERdata was asked to develop a Life History Calendar for a study in the Netherlands that wanted to have a retrospective interview with people who have a criminal record.

We assumed these (former) criminals were reluctant to participate, which raised the value of the interviews. Therefore we had to make sure a completed interview could never be lost so we avoided storing data on the laptops during the interviews. We decided to store the answers on an external server over a secured line. After some investigations we decided that mobile internet coverage was good enough in the Netherlands to support this technique. In this decision it was unfortunately overlooked that some institutionalized respondents are not allowed to have internet access at all.

We grouped life events into thematic groups: partners, children, accommodations, jobs, prison, health and other. The interview routing would by default walk through the questionnaire sequentially, group after group. E.g. the first group asks about children- for each of the children when they were born, what their names are, if they are still alive, if not, when they died. When all children are discussed the questionnaire continues looping through all notable romantic relations the respondent has had and so on. All the known events are represented in the calendar by separated lines. The vertical axis of the calendar showed the lifespan of the interviewee.

The calendar does not only give a graphical overview of life events, but can also be used to navigate through the questionnaire. The idea is that respondents, triggered by the visual representation, mentally link life events to each other. Rather than exactly knowing what year an event took place a respondent is more likely to have remembered that an event occurred at the same time as or earlier or later than another event. This makes it more likely that respondents will find an earlier answer was not correct. Using the links provided by the calendar it is possible to jump directly back to these questions.

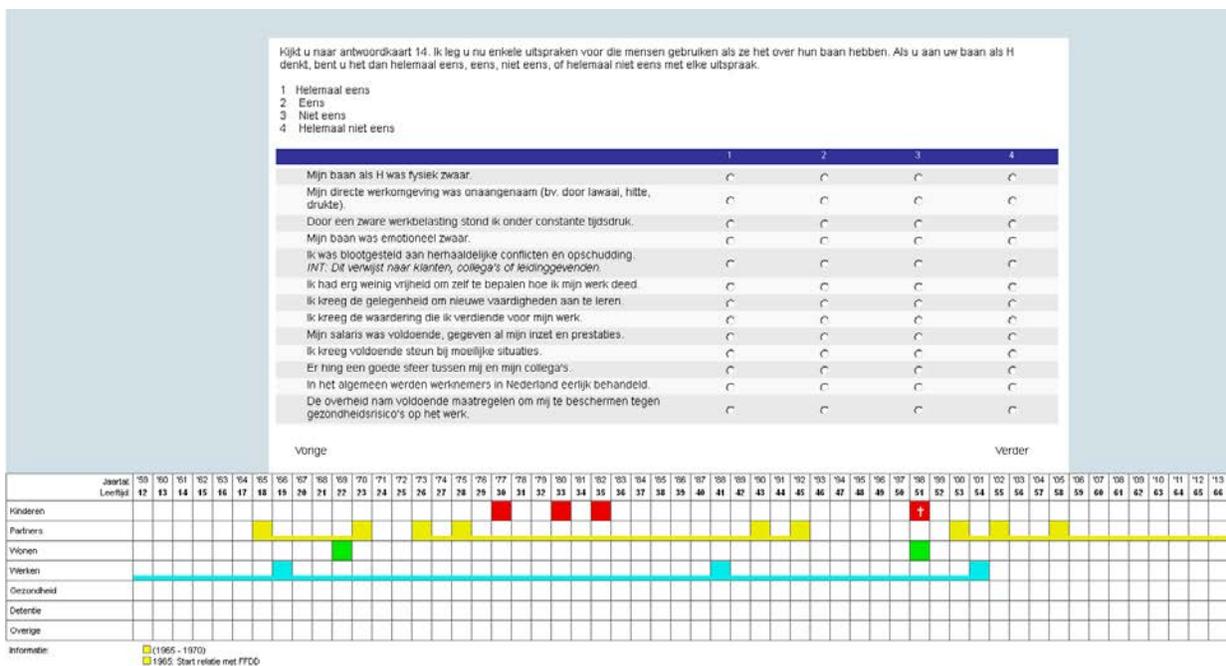


Figure 1: Screenshot of the online LHC

The colored blocks in the calendar represent certain events. When such a block is hovered with the mouse the bottom part of the calendar ('Information' field) displays detailed information on these hovered events. When a colored block in the calendar is clicked the displayed details of the events are locked into place and continue to stay visible. The detailed information contains links that will navigate the user (back) to the questions that set the values for the event.

- piRangeLabel: the label that will show on mouse over on the range
- piSymbol: a symbol that will show in the event tab
- piYear: the year in which the event took place
- piShowRange: whether the range will be visible
- piEndRange: whether the range has an ending
- piLink: the full path and name of the question

The procedure will set a call to a function of a calendar class named push that forwards the JSON description to the service.

```

PROCEDURE SaveEvent
PARAMETERS
  IMPORT piSectionIndex: INTEGER
  IMPORT piRangeIndex: INTEGER
  IMPORT piEventIndex: INTEGER
  IMPORT piLabel: STRING
  IMPORT piRangeLabel: STRING
  IMPORT piSymbol: STRING
  IMPORT piYear: INTEGER
  IMPORT piShowRange: INTEGER
  IMPORT piEndRange: INTEGER
  IMPORT piLink: STRING
  EXPORT peCalendar: THTMLCODE
RULES
  peCalendar := '<script language="JavaScript">
    calendar.push ( {
      "SaveEvent": {
        "Key1": '+STR(nohouse)+' ,
        "Key2": '+STR(nomem)+' ,
        "SectionIndex": '+STR(piSectionIndex)+' ,
        "RangeIndex": '+STR(piRangeIndex)+' ,
        "EventIndex": '+STR(piEventIndex)+' ,
        "Label": "' +piLabel+' " ,
        "EventRangeLabel": "' +piRangeLabel+' " ,
        "Symbol": "' +piSymbol+' " ,
        "Year": '+STR(piYear)+' ,
        "ShowRange": "' +STR(piShowRange)+' " ,
        "EndRange": "' +STR(piEndRange)+' " ,
        "Link": "' +piLink+' " ,
      }
    } );</script>'
ENDPROCEDURE

```

When a new event has to be triggered from the Blaise questionnaire this procedure is called and the result is loaded in a string Calendar, which is can be called in a question text as a fill.

If, for example, we have two questions:

```

RC024_kidyob (RC024)
  "@# ^FL_RC024_1 In what year was ^FL_RC024_2 child born?<br>

```

```

        <i>INT: If year is unknown please .</i>@#" ":
        TYear
RC025_kidname (RC025)
        "@#What is the childs first name?<br>
        ^CALENDAR@#" : TKidname

```

And the following routing:

```

Txt_FL_RC024(index, NrOfKids, FL_RC024_1, FL_RC024_2) RC024_kidyob

RC025_kidname

IF RC025_kidname = RESPONSE THEN
    RC025_kidname := UPPERCASE(RC025_kidname)
ENDIF

```

We would add a call to the SaveEvent procedure in the rules:

```

SaveEvent(1, piIndex, 1, RC025_kidname+' born', RC025_kidname , '',
RC024_kidyob, 0, 0, 'Sec_RC.New_Children['+str(piIndex)+'].RC024_kidyob',
CALENDAR)

```

This will make sure the correct JSON structure will be included in the next question text that has a ^CALENDAR fill included.

The calendar is an html table constructed out of independent div containers that have a mouse over event and a mouse click event defined. These divs are generated by an external service. For example a div with its attributes would look like this:

```

<div
  id="calendar_item_4_2_1"
  class="calendar_item calendar_item_style_cga_4 calendar_range_4_2"
  style="height: 24px; margin-top: 0px; z-index: 43; width: 24pxpx;"
  onmouseout="calendar_information_fixed_show();
  calendar_highlight_hovered(false, '4', '2');
  calendar_information_hover_clear();"
  onmouseover="calendar_information_fixed_hide();
  calendar_highlight_hovered(true, '4', '2'); calendar_information_hover_set(
  '4', '2', '1', 1966, '4_2_1',
  'CALENDAR_ITEM');" onClick="calendar_information_fixed_set();">
</div>

```

An **onmouseover**-event calls three functions:

- `calendar_information_fixed_hide()`; clears whatever detailed information is currently shown
- `calendar_highlight_hovered(true, '4', '2')`; highlights the hovered event(s) in the calendar
- `calendar_information_hover_set('4', '2', '1', 1966, '4_2_1', 'CALENDAR_ITEM')`; displays the detailed information

An **onclick**-event freezes the current set that is selected. This allows us to move the mouse pointer towards any other event without, accidentally, overwriting the chosen displayed details.

```
<a onclick="javascript: FieldFocused(64, 'Sec_RE.RE002_edfinage', '');"  
href="#"> Education </a> (1952)
```

The FieldFocused() function will be called and moves the active question to Sec_RE.RE002_edfinage. To get the FieldFocused function to work, the biPagEvt library should be called from the Style Sheet.

```
<script type="text/javascript" src="libraries\biPagEvt.js"></script>
```

We now established a way of using JavaScript to send JSON strings in Blaise IS. The service that generated the calendar-html was developed in PHP. Unfortunately it was not possible to install the service on the same server and port as Blaise IS. Because of the security design of many browsers it is not possible to call scripts installed on external servers. This issue can be solved by calling a script on the same server that forwards the request. We bridged this problem by using some ASP code that aligned with the Blaise ASP code. The JavaScript code called a service written in ASP (getCalendar.asp), which forwarded the request to a service written in PHP on a different server.

getCalendar.asp:

```
<%  
Dim objXMLHTTP  
Dim contents  
Dim x  
Dim URL  
  
contents = ""  
For x = 1 to Request.Form.Count  
    contents = contents & "&" & Request.Form.Key(x) & "=" &  
Request.Form.Item(x)  
  
Next  
  
URL = "http://someserver/lhc_ajax.php"  
Set objXMLHTTP = CreateObject("Microsoft.XMLHTTP")  
objXMLHTTP.Open "POST", URL, false  
objXMLHTTP.setRequestHeader "Content-Type", "application/x-www-form-  
urlencoded"  
objXMLHTTP.Send contents  
Response.Write objXMLHTTP.responseText  
Set objXMLHTTP = Nothing  
%>
```

4. Speeding things up

When testing the questionnaire some problems were encountered; due to the length of the questionnaire and the number of loops and tables loading time exploded. Waiting several seconds up to half a minute for a new page to show up is simply not workable. In the initial testing phase we found it impossible to work with this. Testing online Blaise questionnaires is in itself a difficult task, because you can never be sure how things will display until the questionnaire is uploaded to a server and viewed with several browsers. With the added problem of the slow loading time, this questionnaire was impossible to test. We implemented a trick to speed things up.

The questionnaire consists of several sections. At the end of each section we set a Timestamp field:

```
TimeStampEnd / "END TIMESTAMP SECTION": TIMETYPE
```

In each section this field is called using the rules:

```

TimeStampEnd.keep
IF (TimeStampEnd = EMPTY) THEN
    TimeStampEnd := SYSTIME
ENDIF

```

This TimeStampEnd is used to set a new section on route only when a previous section is finished.

```

IF (Sec_W.TimeStampEnd <> empty) THEN
    Sec_X
ENDIF
IF (Sec_X.TimeStampEnd <> empty) THEN
    Sec_Y
ENDIF

```

...

This reduced the time it took to generate a page to a reasonable level.

5. Towards a testing environment

The experiences from the development of the online Life History Calendar in Blaise IS triggered some new ideas. Since we found out how to navigate more dynamically, we could use this to create other tools that could help shorten our testing time. In our development of larger online questionnaires we found that especially testing is very frustrating. The layout editor will not always show what a browser will generate. The first challenge is how to let the JavaScript know what fieldname is currently visible. In the html source of a page the name or path is never mentioned. Luckily the Blaise Menu Editor can be used to load this dynamics information in the source code. Using the Blaise Menu Editor we introduce a new control for the language(s) the questionnaire is in.

```

Caption: '<div onload="setPageLog(''+$KEYVALUE+''',
'''+$DATE+''', '''+$TIME+''', '''+$DATAROOT+''', '''+$FIELDNAME+''',
'''+$QUESTIONTEXT+''')">'

```

The type of caption is switched from 'Literal text' to 'Expression'.

Make sure the 'Visible' property is set to True.

The type of the control is set to 'Label'.

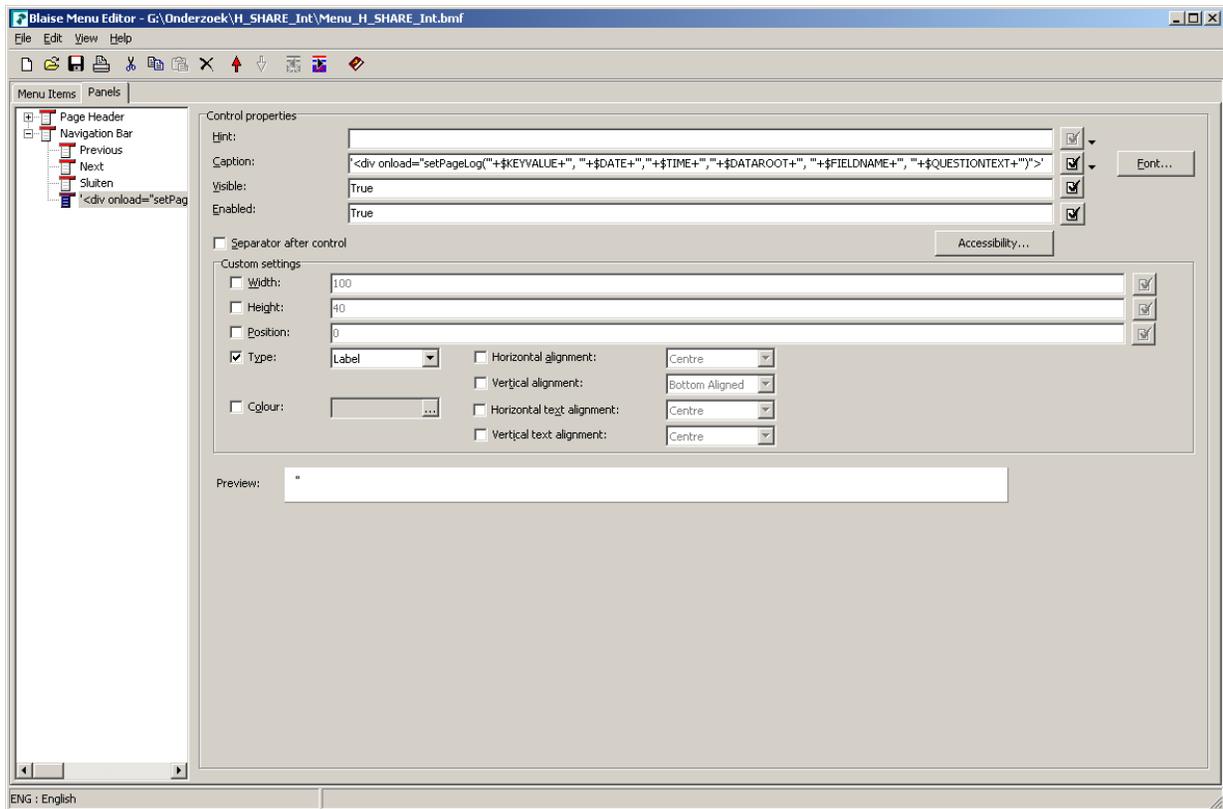


Figure 3: Use the Blaise Menu Editor to feed questionnaire information to the JavaScript

Now each time a new page is loaded in the browser, a JavaScript call will be made to a function `setPageLog`, sending the key, date, time, dataroot, fieldname, and questiontext.

In our system an Ajax call is made to a service `setFieldInfo.php`. Its result will be loaded into a div 'testenvironment'

```
function setPageLog(keyvalue, date, time, dataroot, fieldname, questiontext)
{
    var myurl = "setFieldInfo.asp"
    var params =
"dataroot="+dataroot+"&keyvalue="+keyvalue+"&fieldname="+fieldname+"&date="+date+
"&time="+time+"&questiontext="+questiontext;

    xmlhttp.open("POST", myurl, true);
    xmlhttp.setRequestHeader("charset", "utf-8");
    xmlhttp.setRequestHeader("Content-type", "application/x-www-form-
urlencoded"); xmlhttp.setRequestHeader("Content-length", params.length);
    xmlhttp.setRequestHeader("Connection", "close");
    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
            if (xmlhttp.status==200) {
                setAndExecute('testenvironment',
xmlhttp.responseText);
            }
            document.body.style.cursor = "default";
        }
    }
    xmlhttp.send(params);
}
```

The *testenvironment* div is included in the BlaisePage template in the style sheet of the Interview Page, in our example biHTMLWebpage.xml

```
<div id="testenvironment " />
```

In biHTMLWebpage.xml a reference is made to the blaise_tester.js script and to the stylesheet that the testing environment uses:

```
<script type="text/javascript" src="blaise_tester.js" />
```

```
<link rel="stylesheet" type="text/css" href="style/tester.css" />
```

Make sure these new files are available in the BIS

Some ASP files are added to handle the asynchronous JavaScript calls:

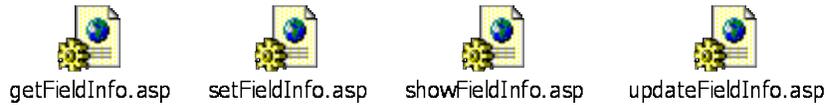


Figure 4: Services as ASP files

Similar to the LHC, we load the testing environment information inline.

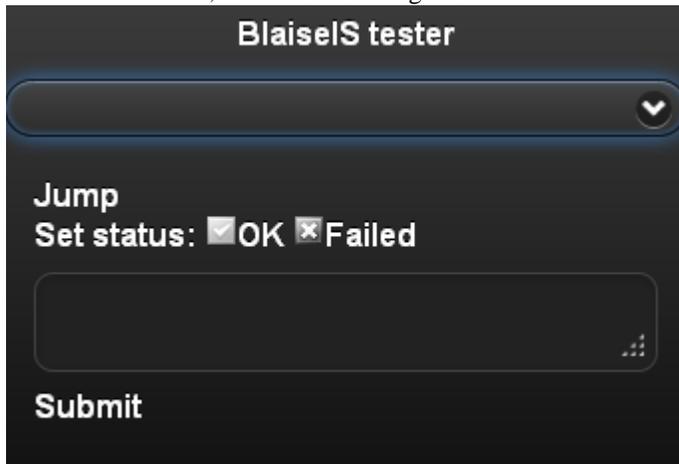


Figure 5: Blaise IS test app



Figure 6: Blaise IS test app with fields found sofar

This app can float and keep track of the route and comments made during the questionnaire test. It seems easily enough to include it into other questionnaires using the method explained in this section. Since it has been recently developed, we did not have the chance to use this in a production environment. The functionality to insert web-apps or other JavaScript applications into online questionnaires strengthens the possibilities Blaise IS has to offer.

6. Conclusion

More and more new ways of complex data measurement are designed and more and more these measurements move online. JavaScript can push the possibilities Blaise IS has, further down this path. Developing questionnaires for Blaise IS is however not very convenient, it sometimes feels like hacking the system to get things working. We hope that Blaise 5 will support the latest techniques better and has easy ways of supporting JavaScript calls and support quick reference testing.

Challenges of Migrating ABS Business and Household Surveys to Blaise Web on a Large Scale and Short Timeframe

Author(s): Adrian Bugg, Kathy Buck, Anthony Davies, Annette Hants, Monica Kempster

Presenter(s): Helen Robson

Organization: Australian Bureau of Statistics (ABS)

1. Abstract

In 2012, the ABS began a challenging program of transforming its data collection activities to include eCollection as an option for most of its collections. The program began rolling out in December 2012 and covers both household and business surveys. The main part of the migration program is due to conclude at the end of 2013. This paper will look at some of the challenges, including building capability in staff, drivers for the change, things that worked well, learnings and future plans.

2. Background

The Australian Bureau of Statistics (ABS) is Australia's national statistical agency. By providing trusted statistics and statistical leadership, the ABS supports public debate and helps Australians to make informed decisions in an increasingly complex world. The ABS provides statistics on a wide range of economic, social, population and environmental matters for government, business and the community.

The ABS continues to be regarded as a world leader amongst national statistical agencies. However, we face a number of challenges including increasing demands for more timely and diverse statistical data on our economy, society and environment. Additionally, we operate in a fast changing information landscape and we are constrained by a tight financial situation. The ABS is facing increasing collection costs and complexity as well as provider resistance. As Australia's official statistical agency, our ability to effectively respond to these challenges is key to our ongoing success.

The ABS needs to ensure that high quality official statistics are readily available to governments and the community when key decisions about the future of our nation are made. In response to this, the ABS has embarked on a significant journey of change – the ABS 2017 Program - that will transform the way we collect, manage and deliver information and statistics. Without this change, our ability to achieve our mission into the future is at risk; we will not be able to continue to fund our existing work program, nor respond effectively to the changing needs of our users.

The ABS commenced planning its business and information management transformation program in 2010. This program represents the most strategic initiative to update statistical business processes and information management infrastructure within the ABS since the 1970s.

In early 2012, the ABS 2017 Group was formed to lead the transformation of the way that the ABS collects, collates, manages, uses, reuses and disseminates statistical information. The transformation program has three key goals:

- to reduce the cost of doing business by streamlining operations to reduce the time it takes for information to move through each stage of the statistical production process. By significantly

reducing the cost of collection and processing, resources can be reinvested in higher value customer products and services.

- to grow the business through new statistical products and services. For example an enhanced capability to quickly bring together data from a range of sources (e.g. administrative, transactional, survey) will help to better shed light on complex economic, social and environmental problems.
- to deliver the first large scale digital Census (2016) on time, budget and to a high quality. This represents the most significant change to an Australian Population Census in 100 years.

The ABS 2017 program needed to ensure a sustainable future for the ABS. The ABS needs to be more productive, timely and flexible with the information that we collect, process and deliver to meet the expectations of user communities.

The ABS must transform its operations in order to maintain our relevance for Government and the Australian community. To meet the ABS 2017 goals, large scale innovation is needed across the ABS, but in a timely manner whilst delivering on business as usual and maintaining the reputation and trust that the ABS is renowned for.

The ABS has been moving toward a goal of increased electronic data collection for some time and we have achieved success through the 2011 eCensus and the Agricultural Census eCollection. The Longitudinal Study of Australian Children and the Household Energy Consumption Survey also successfully deployed web-based survey components in 2011–12. In 2016, when the ABS conducts its first predominately digital census, the target is a minimum 65% eForm response.

Under the ABS 2017 banner, the ABS has embarked on its short term strategic priority of developing key enabling infrastructure, including eCollection capability, within our business and household survey program and for the 2016 Census. The five-yearly Census of Population and Housing is the largest statistical collection undertaken by the ABS. Since August 2011, the ABS has collected approximately 6.1 million Census forms and 2.8 million eCensus submissions, and we have converted these responses into Australia's most important dataset, providing information on our population, where we live, and our key characteristics.

3. Collection Costs and Complexity

The ABS has recently examined response rates being achieved for ABS surveys and the level of effort being expended on intensive follow up to achieve these response rates. There is evidence of provider resistance to supplying survey data to the ABS, and this has been steadily increasing over the past ten years. A steady decline in response rates has been detected for some key surveys. Analysis suggests that this decline is driven by external factors outside of the control of the ABS.

Many business surveys have in the past required providers to complete a paper form. As these surveys are now migrating to eCollection, it is hoped, and experience so far has shown, that providers are more willing to participate using eCollection. This mode is less time consuming and burdensome for providers, avoiding the need for completion and mail back of a paper form. Response rates for these surveys will continue to be carefully monitored to establish if eCollection has increased participation and resultant response rates.

4. The Journey

The project to deliver ABS eCollection capability for household surveys commenced in 2010 when it was envisaged that the system developed for the 2010/11 Agricultural Census would be enhanced to accommodate the additional requirements of these collections. The Monthly Population Survey (MPS) would be the first household survey to adopt eCollection.

Following the successful deployment of the Agricultural Census eCollection System (ACES) system, work commenced in 2011 to build the necessary capability. By early 2012 it became evident that, without a significant redevelopment requiring the injection of additional funds, ACES would not deliver an acceptable solution for MPS.

In February 2012, a Request for Expressions of Interest (RFEOI) was lodged seeking information from Industry on six core capabilities, one of which was eCollection. A total of 21 companies responded to some or all of the six capabilities presented in the RFEOI, however, there was no obvious or clear solution provided to any of the six capabilities. Statistics New Zealand participated in this process through membership of the ABS2017 EOI Steering Committee, and in the post evaluation discussions.

The ABS already had considerable existing investments in Blaise as a data collection platform for personal and telephone interviewing, as well as editing and processing for business and household collections. Blaise 4 was already in widespread use within the ABS for Computer Assisted Telephone Interviewing (CATI), Computer Assisted Personal Interviewing (CAPI) and editing purposes.

Blaise was known to be able to handle the complex household survey instruments. MPS and Special Social Surveys (SSS) instruments already used a form of Blaise and it had the added benefit of being used by other international statistical agencies.

An evaluation was subsequently undertaken into the potential for the Internet version of Blaise (Blaise IS) to form the basis of the solution for the online data collection of household surveys, specifically the MPS. Blaise IS offered a ready-made solution that would integrate well with existing survey collections. This solution needed to be available for deployment for the MPS from the December 2012 cycle. The evaluation of Blaise IS recommended proceeding with a Blaise based eCollection capability. Testing has since confirmed that Blaise IS is a more suitable platform for current and future survey collection requirements.

Following a visit by a Statistics Netherlands Blaise IS expert, an evaluation was conducted by the ABS to understand how it could be deployed and what work might be involved in integrating it with other ABS processing systems. The evaluation confirmed the potential of Blaise IS as a cost effective, functionally suitable, solution in the short- to medium-term.

As a result of these evaluations, the February 2012 Household eCollection Program Board endorsed a proposal to proceed to use Blaise IS for household surveys, specifically the development of online collection capability. Following this decision, the ABS commenced the accelerated development of an eCollection solution in May 2012 using Blaise IS for household and business surveys.

The full transformation program included development of the following:

- eCollection for ABS household and business collections
- Administrative data - receipt, transform and load functionality to support the Enterprise Data Warehouse project
- Mobile devices and applications for use by field staff and providers
- Workload and Workforce Management
- Provider/collection interaction and management systems (including a provider/user portal).

Following adjustments to funding and discussions by senior management, the program was rescoped to focus on translation of ABS household and business surveys to eCollection.

4.1. Plans for migration of the Monthly Population Survey to online collection

The MPS has been conducted by the ABS since 1960 to provide regular information about the population and labour force of Australia. Key economic indicators such as the unemployment rate are produced from MPS data. Approximately 35,000 households around Australia are included in the MPS each month. Households are included in the MPS for eight consecutive months. Using a rolling sample model, dwellings are replaced every eight months, therefore seven-eighths of the month to month sample is the same as the previous month.

The MPS is made up of the Labour Force Survey (LFS) and supplementary survey topics such as education, the environment, conditions of employment and child care arrangements. The LFS component can be answered by any adult member of the household. The Multi-Purpose Household Survey (MPHS) is an additional one-off supplementary survey containing a mix of topics. One randomly selected person in a proportion of MPS households is selected for the MPHS each month. The MPHS is only asked of outgoing MPS rotation group households, who are in their final month of MPS.

The MPS eCollection capability was tested in a full dress rehearsal which was conducted in October and November 2012.

Following the dress rehearsal the ABS initially offered eCollection in December 2012, to one "champion" rotation group. This rotation group continued in the MPS for its full 8 month cycle, finishing in July 2013.

To effectively manage risk and measure the statistical impact of migrating to eCollection for the MPS, a roll-out and statistical impact measurement strategy was developed. This strategy measures the "offer of eCollection" effect to the MPS results, rather than measuring the actual "mode effect" of the eCollection instrument in isolation.

The measurement strategy has two main aims:

- to identify during rollout if there is any catastrophic statistical impact, defined as being a treatment effect greater than three Labour Force Survey (LFS) Standard Errors; and
- to measure (if possible) the size of the statistical impacts post the transition to eCollection, to be able to quantify them for LFS users.

Under the measurement strategy, eCollection has been offered to 50% of incoming MPS rotation group from May 2013 and extends the eCollection offer to 100% as each rotation group refreshes. This option fulfils both the ability to identify if there is a catastrophic impact and, if large enough, to quantify it for users. Full eCollection implementation would occur by April 2014 at the completion of the measurement period.

4.2. Migration of Monthly Population Survey to online collection

Phase 1

Phase 1 of the Migration of Household Surveys to eCollection involved the development of a web form for the MPS, using Blaise IS. A system was developed that integrates Blaise IS into the ABS environment and offered electronic enumeration to MPS respondents.

Infrastructure and processes were put into place to integrate the data collected via web reporting (Blaise IS) with the data and support mechanisms for existing collection methods for MPS, i.e. CATI and CAPI (managed through the Computer Assisted Interviewing Workload Management System (CAIWMS)).

The full functionality of the system was available in time for the December 2012 MPS, after the DR conducted in October and November 2012.

Development of eCollection brought it to a stage where it was considered suitable to maintain the collection of data and provide stability to the process required for an effective measurement strategy while balancing the potential risk introduced by continual changes. Changes were minimised to allow for a stable collection via eForm month to month during the implementation and roll out to more sample.

From May 2013, it was agreed that no further eForm changes were to be made under the MPS Phase 1 Program for eCollection until completion of the measurement strategy. Further system development continued to support infrastructure requirements for eCollection in general.

Outstanding issues identified during Phase 1 were brought forward to be addressed as part of MPS Phase 2 migration.

Phase 2

Phase 2 of the eCollection program is currently being developed. Online collection Phase 2 is proposed for delivery after the measurement strategy has concluded. This phase will include the following developments:

- Any corporate initiatives that improve coding functionality and the application of standard coding frames for industry, occupation, country of birth or other significant coding frames applicable to supplementary surveys and MPHS
- Changes identified but not included in the changes made for the May 2013 instrument
- Changes to the structure of eForms to meet accessibility requirements as per Web Content Accessibility Guidelines (WCAG) 2.0
- Series of questions to accurately identify scope and coverage exclusions
- Any improvements suggested by LFS data analysis
- Review of the Household Form operation
- Review of in-form edits

- Any improvements identified to improve respondent experience and reduce overall completion time
- Incorporation of additional data in Blaise database structure to support ongoing mode analysis
- Improved approach materials and review of survey supporting information
- Improvements to the Household Contact Details Form
- Pre-approach strategies to increase uptake of eCollection.

4.3. Migration of Business Surveys to eCollection

The migration strategy for business surveys proposed a roll-out schedule for migration to eCollection that grouped surveys into six batches.

Batch 1 – subannual surveys, large sample volume, with a form that is relatively simple to create with the Blaise web form functionality that has been established by December 2012 (i.e. the functionality required for the Labour Force Survey form). Batch 1 will deliver large savings for small investment.

Batch 2 – subannual surveys, with forms that will require some additional functionality in Blaise, or have some other element of complexity. This group also includes some surveys with relatively simple forms but with smaller volumes than Batch 1 surveys.

Batch 3 – subannual surveys with forms that have a heavy use of explanatory material. Some further thought needs to be given to how these forms are presented in web format.

Batch 4 – the remaining subannual surveys. These surveys have been assessed as a relatively low priority for migration, either due to small sample size, or else the availability of spreadsheet-based electronic reporting.

Batch 5 – annual surveys. This batch aims to make substantial progress towards migrating annual surveys without being too ambitious, preference is again given to collections with large sample sizes. This batch also lists surveys where web forms have already been introduced.

Batch 6 – The remaining annual/irregular surveys (on hold)

There are a relatively small number of business surveys that encountered obstacles to the use of eCollection and will not be migrated. The reasons for this include incompatibilities in infrastructure, specialised data collection requirements or cost factors.

4.4. Progress of Migrating Surveys to eCollection

As discussed earlier, the ABS has focused on implementing eCollection capability for the Monthly Population Survey, most business surveys and preparation of an eCollection solution for the Census test in August 2013.

Significant achievements have included:

- Progressive implementation of MPS commencing from December 2012
- Implementation of quarterly business collections and other Batch 1-4 surveys – See Table 1
- Preparation for MPHS and MPS supplementary topics
- Readiness for annual collections
- Preparedness for Census test in August 2013
- Blaise workshop for international collaboration.

By the end of 2013, the ABS will have implemented eForms for most quarterly and annual business collections, including surveys that result in Major Economic Indicators (MEIs), the MPS (including the MPHS and MPS supplementary topics).

Further to this, we will have established an eCollection capability for household and business collections (based on Blaise) that will have been evaluated for Census. Subject to this evaluation, Blaise will be further developed as the corporate capability or used in conjunction with an enhanced version of the application used for the 2011 Census. In progressing this work, the ABS has established productive working relationships with Statistics Netherlands and has been sharing knowledge and expertise through the International Statistical Network.

In the process of transforming our data collection activities, all relevant areas of the ABS have been working closely together, from subject matter areas through to technical experts. This has assisted in meeting our goal to integrate our activities and systems and reduce duplication of effort. This collaborative effort and commitment from all relevant areas has been vital to success.

All services that support eCollection were designed to provide a corporate capability, for use across business and household surveys. This corporate capability will be used by all household and business collections and the 2016 Census.

Tables 1 and 2 contain lists of the business surveys now using eCollection as the primary mode of data collection and those expected to adopt eCollection in 2013-14 and 2014-15.

Table 1: Business collections now using eCollection as the primary mode of data collection (as of July 2013)

Collection	Cycle Sample size	First use of Blaise eCollection
Internet Activity Survey (IAS)	Half yearly 100 to 600	December 2012
Business Indicators Survey (QBIS)	Quarterly 16,000	March 2013
New Capital Expenditure (CAPEX)	Quarterly 8,000	March 2013
Survey of Tourist Accommodation (STA)	Quarterly 4,500	March 2013
Retail Trade Margins Index (RTMI)	Quarterly 150	March 2013
Engineering Construction Survey (ECS)	Quarterly 2,000	March 2013
Average Weekly Earnings (AWE)	Half yearly 5,500	May 2013
Coverage Survey for International Trade in Services (SITS coverage)	Quarterly 800	May 2013
Survey of Employment and Earnings (SEE)	Annual 2,000	June 2013
Rural Environment and Agricultural Commodity Survey (REACS)	Annual 35,000	June 2013

Table 2: Business collections expected to adopt eCollection in 2013-14 and 2014-15

Collection	Cycle Sample size	First use of Blaise eCollection
Annual Integrated Collection (AIC)	Annual 40,000	August 2013
Agricultural Land and Water Ownership Survey (ALWOS)	Annual 11,000	August 2013
Survey of Venture Capital and Later Stage Private Equity (VC)	Annual 250	August 2013
Private Health Establishments Collection (PHEC)	Annual 600	October 2013
Survey of Motor Vehicle Use (SMVU)	Every second year / 3 segments 16,000	October 2013
Freight Movements Survey	One-off 16,000	October 2013.
Building Activity Survey (BACS)	Quarterly 11,000	December 2013
Job Vacancy Survey (JVS)	Quarterly 5,500	November 2013
Vineyards	Annual 10,000	May 2014
Survey of Employee Earnings and Hours (EEH)	Every second year 63,000	May 2014
Land Management Practices Surveys (LAMPS)	Annual 50,000	July 2014
Business Characteristics Survey (BCS)	Annual 17,000	October 2014
Cultural Funding by government	Every second year 7,000	January 2015
Research and Experimental Development (Businesses)	Every second year 7,000	January 2015

4.5. eCollection Take Up Rates

pleasing take up rates to date indicate that Australian households and businesses are happy to respond via the web. Businesses are particularly keen to use eCollection for our surveys, with an average of 66% of all providers offered eCollection now completing their survey on-line.

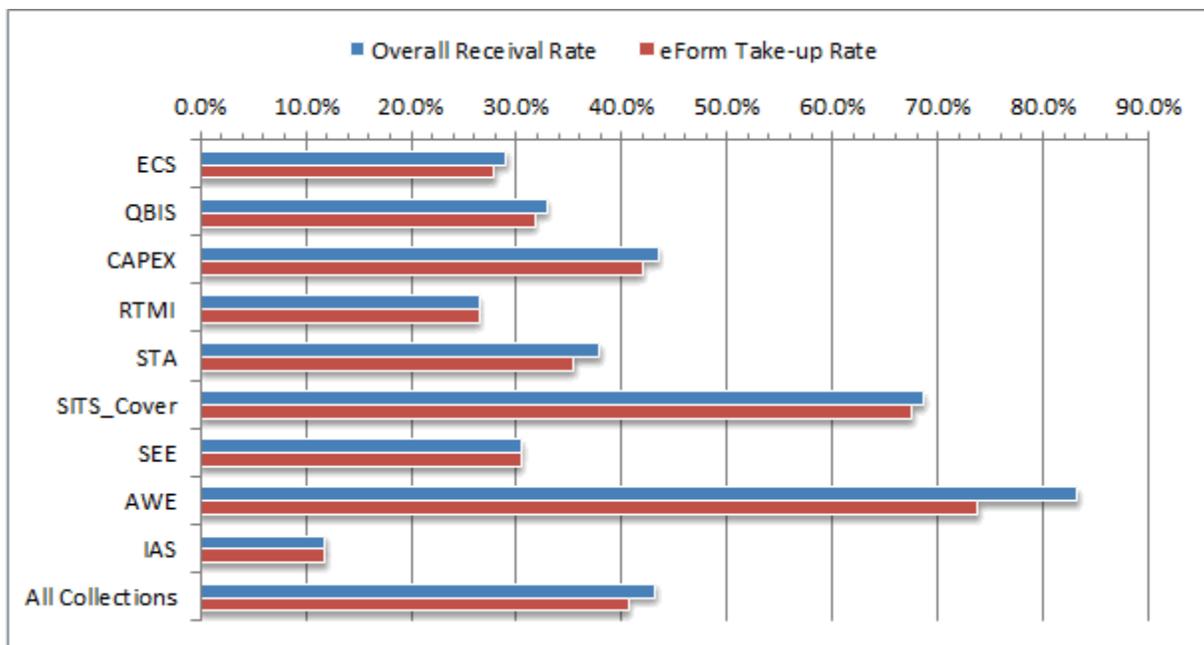
On the household side, implementation of eCollection for the MPS is well underway, with approximately 19% of the sample now offered eCollection.

The opt-out strategy used for business surveys involves a letter being sent to providers, which offers an online option only for survey completion. This strategy is resulting in higher take up rates for business surveys compared to household surveys, as the onus is on the provider to contact the ABS if they are unable to do the survey online.

On the household side for MPS, eCollection take up rates are lower as the offer of eCollection letter encourages respondents to complete the survey online, but also provides another option. Respondents are encouraged to complete the registration process and the survey online by a particular due date. If this date is not met, they are advised that an ABS Interviewer will visit the address to conduct an interview. Households enumerated by an Interviewer are asked after the first month's interview if they would prefer future interviews to be conducted online.

Tables 3 and 4 report the latest take up rates for eCollection for Business and Household Surveys.

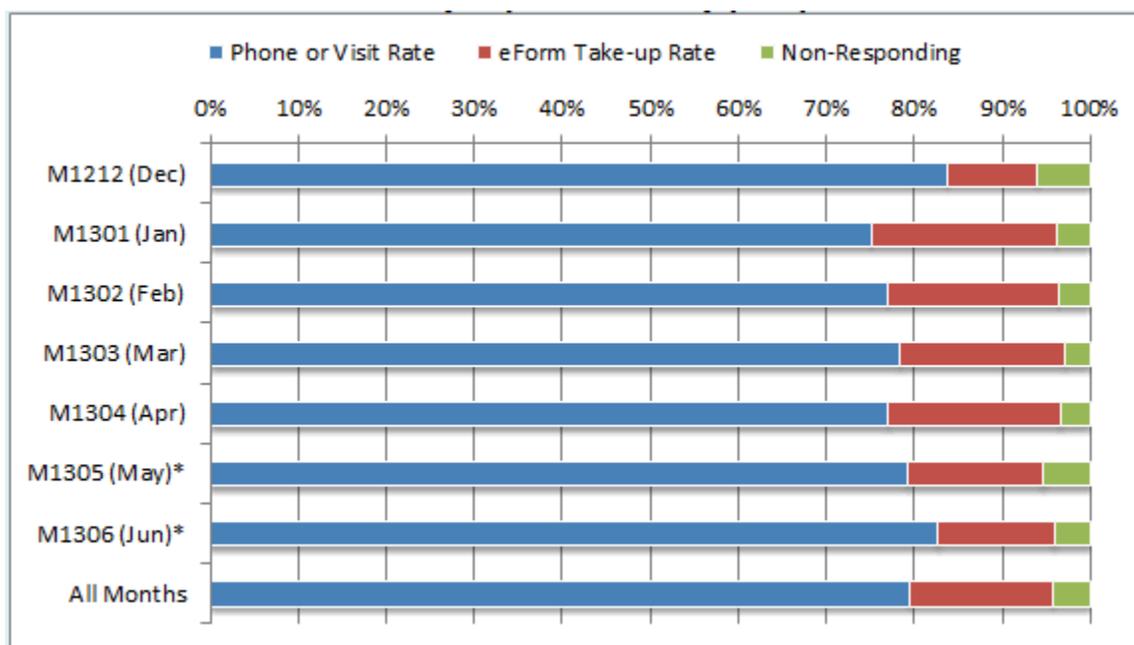
Table 3. eCollection Take Up Rates for Business Surveys in July 2013



* Current as of 15 July 2013. ECS, QBIS, CAPEX, RTMI & STA are June quarterly surveys.

** eCollection take-up rate includes data entered by providers into an eCollection and data entered into an eCollection over the telephone by PCU staff.

Table 4. eCollection progress of the Monthly Population Survey (MPS) since December 2012



ECollection take up rates on commencement for MPS (December 2012) came in at the expected rate of 10%. This rate increased in subsequent months and is currently averaging 18%. Management Information Reports indicate that some households switch between modes, moving from eCollection to Telephone Interview and vice versa during the 8 months that they are in MPS.

Respondents in households offered eCollection who initially decline the offer are enumerated face to face or by telephone. If the MPS was completed by an Interviewer, the Interviewer was prompted to ask the respondent whether they would do the MPS online the following month.

During the 2011 Census of Population and Housing 33% of Australian households chose to complete the Census questionnaire online using the 2011 Electronic Census Lodgement Solution (eCensus). This was a significant increase on the 10% who chose to use the eCensus in 2006. This increase was driven by a combination of factors, including a greater focus on ‘selling’ the benefits of the eCensus ‘at the door’ by Census field staff, and strong growth in Internet use and connectivity throughout Australia in between Censuses. This growth in take-up is encouraging given the ABS’s stated aim of achieving at least 65% eCensus take-up in 2016.

5. Challenges, Opportunities and Lessons Learned

Integration issues were identified during the initial stage of eCollection implementation. These were subject to incremental fixes and enhancements to ensure a stable eCollection experience. Issues identified included:

- the need for early identification of issues to enable technical staff to identify areas of risk and complexity which might impact on eCollection development or deployment
- the need for better control of the agreed timetable for development and operations
- constant competition for resources between business as usual and eCollection development

- the need to focus on the whole collection process, not just eForm delivery
- insufficient time for end to end testing
- short timeframes required for instrument delivery
- insufficient metadata management
- instability of legacy systems
- variation of lead times between business collections and the consequent need for timeframes to be considered on a case by case basis
- failed prefill for rollover of LFS December household form data to January online form.

Overall, ABS eCollection deployment capability improved during this migration phase, demonstrated by the decreasing number of eCollection iterations needed as a result of testing.

5.1. Ambitious Release Schedule for Migration to eCollection

The original planned release of eCollection proved too ambitious in the timeframes allowed and had to be scaled back. A decision was made to proceed to eCollection for the Monthly Population Survey and the Internet Activity Survey. Originally the ABS had aimed to migrate other business surveys ie Quarterly Business Indicators Survey (QBIS) and Capital Expenditure Survey (CAPEX). The ABS has now successfully implemented all March 2013 quarterly business surveys.

In transitioning to eCollection, subject matter areas had to specify requirements to Blaise programmers twice, one set for eCollection and one for CAPI. In turn Blaise programmers had to build two instruments. No additional time could be built into survey development timetables to cater for the increase in work.

There were a large number of Service Requests (SRs) approved for action by developers, and it was difficult to work through these systematically and efficiently. Tight timeframes, changing priorities and unexpected dependencies also resulted in some SRs being overlooked or not correctly implemented. In addition, system integration issues were underestimated.

5.2. Change Management

Subject matter areas in the ABS were initially reluctant to move their surveys to eCollection. These areas are, however, now using data collected via eCollection. Once they migrated, expectations needed to be carefully managed, as many staff wanted to access features available relating to presentation and pop up displays on screen. At this stage of eCollection implementation, only limited enhancements are provided and this will be reassessed through the transformation phase.

5.3. Password Resets on Inbound Calls

The work of ABS staff responsible for handling respondent queries increased substantially as a result of offering eCollection. The main reason for the increase was due to the need to reset respondent passwords, as many respondents either had trouble logging-in to the eForm or had lost their password. Self-managed passwords are a high priority for the ABS for this reason, as it will allow respondents to quickly reset the password themselves, rather than needing to contact the ABS by telephone or email.

5.4. Limited Resources

The ABS IT environment required solutions and expertise to support the systems. Staff involved in implementation of eCollection, particularly technical staff, needed to work long hours to ensure all the required changes were made to allow for eCollection to effectively integrate with existing systems.

5.5. Integration with ABS legacy systems

There was limited time to integrate Blaise eCollection with ABS legacy systems such as the CAIWMS. Several work arounds have been put into place to allow eCollection to proceed for MPS. For example, households offered eCollection are initially allocated into an interviewer workload in the CAIWMS. If the household takes up the offer and completes the survey online, a process was put into place to remove the household from the interviewer's workload.

5.6. Authentication and Authorisation

It has been demonstrated through external load testing, that the current authentication and authorisation solution, which is a Blaise instrument that utilises external system calls, is not performing optimally, and is at least partially responsible for the reduced Blaise capacity and stability. There are plans to replace the current authentication and authorisation tool, which will not only provide a solution for the eCollection platform, but also be implemented for broader use within the organisation.

5.7. Testing

Availability of testing environments for eForms was limited. Deployment of surveys required technical staff involvement, which was frustrating for staff responsible for testing surveys. Setting up testing records was a difficult process. Timeframes allowed for testing were at times limited due to late delivery of survey instruments.

6. Where to Next

The next 12 months through 2013-14 will consolidate the translation work through provision of eCollection for the majority of business collections, the Monthly Population Survey (including Multi-Purpose Household Survey and the supplementary surveys) and provision of an eForm for the Census Major test in August 2014. The ABS aims to increase uptake of eCollection by examining and improving pre-approach and follow up strategies for households and businesses.

Beyond 2013 the focus will shift from implementation of eCollection to other components including provider management and workforce management to improve the efficiency and effectiveness of our collection operations. It is essential that the ABS build on the capabilities being developed for Census to provide the next generation of systems through which improvements and savings can be achieved. Use of mobile devices and applications will be a key aspect of further modernisation of how we utilise our field workforce and interact with providers.

Further, there will be opportunities to build on the gains already made through implementation of eCollection by transforming electronic collection instruments as distinct to the current approach of

translating existing paper forms. This transformation will drive further productivity improvements for example through incorporation of edits that reduce the amount of processing (editing) required. Savings are currently being derived through reduced collection expenditure (e.g. printing, postage and scanning) whereas further transformation will see benefits obtained from subject matter areas involved with processing.

7. Summary

In summary, the ABS has been successful in migrating a considerable number of surveys to eCollection in a short timeframe. Overall, the quality of instruments has been good but the provider experience can be improved. Our ambitious program has managed risk, but ultimately achieved the outcomes sought by adopting a minimalist approach to the initial round of eCollection.

ABS capability in building and using eCollection will continue to develop, and the cost of deploying eCollection will fall. eCollection delivery processes will be progressively incorporated in standard operations and ultimately become a component of "business as usual". This will include capability to rapidly develop and deploy new eForm requirements, as well as roll-over new cycles for ongoing collections.

Engagement is continuing with Statistics Netherlands, and the international Blaise community on the staged release of Blaise 5 and what its capacities are. In conjunction, ABS is seeking to engage with other National Statistical Organisations (NSOs) to learn what has been done with Blaise Internet versions to date, including whether it has been used for a Census deployment, and what other NSOs may be planning for future versions of Blaise Internet. Our interest includes any investigation into the use of Blaise Internet for developing smart phone and tablet applications.

The second phase of the Blaise evaluation project has commenced and includes:

- Further load and performance testing, to be undertaken by a strategic external partner;
- Further security testing on both Blaise IS and Blaise 5;
- A full evaluation of Census, household and business survey requirements against Blaise 5; and
- Testing of integration with Census 2016 back-end systems and infrastructure.

Phase 2 work will contribute to Census 2016's key decision point in November 2013, when a decision will be made on Blaise Internet for the 2016 eCensus solution.

8. References

ABS Annual Report, 2011-12 (Cat no. 1001.0)

[1001.0 - Australian Bureau of Statistics -- Annual Report, 2011-12](#)

ABS Corporate Plan , Jul 2012 (Cat no. 1005.0)

[1005.0 - ABS Corporate Plan, Jul 2012](#)

ABS Forward Work Program 2012-13 to 2015-16 (Cat no. 1006.0)

[1006.0 - Forward Work Program, 2012-13 to 2015-16](#)

Volguine O, IBUC 2013 Conference Paper: Blaise 4.8.4 Web Form Load and Performance Testing, July 2013.

Gligora, C, ABS Internal Paper: MPS eForm Report and Analysis; December 2012 and January 2013.

Griffiths, G, ABS Internal Paper: Measurement strategy for identification of statistical impact on MPS estimates arising from the introduction of Web form collection, 2011.

Griffiths, G, ABS Internal Paper: MPS eForm Statistical Impact measurement strategy, June 2012.

Dubois, C, 2011 eCensus Take-up - What the data tells us- Final Project Report, July 2012

Blaise 4.8.4 Web Form Load and Performance Testing

Author(s): Oleg Volguine, Presenter(s): Helen Robson, Lane Masterton

Organization: Australian Bureau of Statistics

1. Abstract

This is a technical paper on load and performance testing of Blaise 4.8.4. The Australian Bureau of Statistics (ABS) has recently introduced web surveys for its collections using Blaise Internet. In order to ensure the quality of provider experience the ABS has undertaken both internal and external load and performance testing. This paper details the investigations undertaken, including the methodology, approach, and test strategies utilised, as well as the results and findings that were obtained.

2. Background

ABS has made a strategic decision to use Blaise 4.8.4 (Blaise IS) for all web form development. The first household collection released on Blaise IS in December 2012 was the Monthly Population Survey (MPS). Migration of business surveys to Blaise IS commenced in December 2012 and will be completed in July 2014. Throughout the June and September quarters for 2013, the eCollection platform is expected to support up to 142,000 eForm submissions across many business and household survey collections.

The ABS has undertaken load and performance testing in order to ensure a stable and responsive online survey respondent experience. The testing for Blaise eCollect system has been carried out by ABS staff, and through an engagement with an external load and performance testing partner, to establish an understanding of Blaise IS capabilities, optimal infrastructure configuration, scalability options and how this solution supports the anticipated business outcomes for surveys going live in June 2013 and beyond.

3. Purpose of Testing

3.1. Purpose

The purpose of the test program was to evaluate the capacity of the Blaise 4.8.4 eCollect platform to process the expected production load levels for June 2013 ABS eForms. Specifically the goals of this program were designed to:

- Assess the response times of key transactions under production load, such as login (including both authentication and authorisation), completion of survey data, and survey submission.
- Evaluate the reliability of the Blaise 4.8.4 infrastructure while processing production levels of load over an extended period.
- Assess system performance for different end user network speeds, 56kbps, 64kbps, 512kbps and 2048kbps.
- Identify any bottlenecks impacting system performance and highlight options for resolution.

- Facilitate an initial round of system performance diagnosis and optimisation.
- Provide key learnings for further performance and load testing and to support future system optimisation.

3.2. Load and Performance Targets and Service Level Agreements (SLAs)

In order to ensure a stable, reliable and responsive provider experience, the following SLAs and key performance metrics were targeted:

- 5 seconds for at least 90% of all other transactions (page to page transactions).
- 15 seconds for at least 90% of login (authentication and authorisation) transactions.
- Stable and responsive system behaviour over time:

That is, no system performance degradation over time such as memory leaks or excessive use of hard disk space or system instability. The business requirement is that the system is operational 24 hours per day and 7 days a week.

4. Test Strategy

The test strategy consisted of a suite of tests designed to assess system performance, reliability and responsiveness. The full suite of tests and their purpose is described in this section.

4.1. Normal (Average) Capacity Test

The purpose of this test was to assess the system's ability to cope with the average production level load expected on a peak day. Average load was defined as the load that the system was expected to cope with on its busiest day. The average value was derived by taking the total number of transactions on the busiest day and averaging over the time that the system was expected to be used. For the Blaise eCollect platform this value was derived by looking at the historical number of paper form submissions for all surveys that were expected to be migrated to eForms by June 2013 and combining the number of submissions on the peak day for all surveys over the full enumeration period. The total was then averaged over the number of hours that the system was expected to be operational on any day which was 11.

4.2. Peak (Absolute) Capacity Test

The purpose of this test was to assess the system's ability to cope with the absolute maximum load expected on a peak day. For example, on a peak day, for a period of time the system may experience a load that is much higher than the average peak day load. For the purpose of testing the Blaise eCollect system this value was based on historical information from previous online surveys and from the 2006 and 2011 Australian Population eCensus experiences. This value was set at 1.7 times the expected average capacity load.

4.3. Endurance Test

The purpose of this test was to assess the system's ability to cope with a sustained load over a prolonged period of time, the test was set for 8 continuous hours. For this test, the system must remain stable and responsive without performance degradation over time such as increased usage of memory (memory leaks), excessive use of hard disk space, system instability or failure.

4.4. Stress Test

The purpose of this test was to assess the limits of the system's performance under a given hardware and system configuration. The test was designed to push the system's limits far beyond the expected production levels of load. For the Blaise eCollect system the levels of stress test load were set based on the results of the peak capacity tests and were double the load set for the peak (absolute) capacity test.

4.5. Data Extraction Test

The purpose of this test was to assess the system's ability to cope with the production level load expected on a peak day while data was being extracted from the live production database and loaded into the back-end ABS systems. The business requirement for this was that the system must allow for regular data extraction and loading of submitted surveys into back-end processing systems (one every hour), while continuing operating normally. The regular loading of data into ABS systems provides up-to-date management information for further follow-up with survey providers.

5. Load Modelling and Test Methodology

5.1. Modelling the Expected System Load

The modelling of the expected system load was based on estimating the maximum number of respondents expected to use the system at any particular time. The aim of this approach was to estimate the number of concurrent users and total survey submissions per hour. The transactions per hour target could then be used to derive more fine-grained transaction rates if needed, such as transactions per minute or transactions per second.

The modelling for the expected system load was based on existing metrics from the ABS paper based business survey returns and interviewer based household forms returns. For the round of load testing undertaken for June 2013, all available information on respondent return rates for all surveys that were going to be offered as eForms in June were analysed and combined to derive the total number of returns per day. The largest number of combined daily survey returns expected on any given day was 3,938. This number was then used to derive the average peak hourly rate of survey submissions, by dividing 3,938 by the number of hours in a day that respondents could use the system. For example, while the eCollect platform operates 24 hours, 7 days a week, the system is only significantly utilised for 11 hours, between the hours of 8am to 7pm. This figure was based on the Blaise eCollect survey metrics from the March 2013 quarter.

In addition to the average peak hourly rate, the absolute peak hourly rate was also used in order to account for a load that was higher than the expected average hourly load. For example, on a peak day, for a period

of time the system could experience a load that was much higher than the average peak day load. For the purpose of testing the Blaise eCollect system this value was based on historical information from previous online surveys and from the 2006 and 2011 Australian Population eCensus experiences. This value was set at 1.7 times the expected average capacity load.

The average peak and absolute peak targets were derived as:

Average Peak hourly rate: $3,938/11 = 358$ survey submissions per hour.

Absolute Peak hourly rate: $1.7 \times 358 = 607$ survey submissions per hour.

The peak hourly rates were then used to derive the expected number of concurrent users required to achieve the target number of survey submissions an hour. This approach was based on overall target submissions an hour and the average time it took to complete the survey. It was also based around the assumption that not all users would start and complete their surveys at the same time and in step with each other. For example, if the target hourly rate was 360 survey submissions then the users would have to be working through the survey at a rate of 6 surveys per minute to reach this target. Also, if the survey took 20 minutes to complete and submit, then in 20 minutes, $20 \times 6 = 120$ submissions would be expected. Therefore, the required minimum number of independent users or concurrent users required to complete their surveys at a rate of 6 submissions per minute would have to be 120. This was the number of users at any point in time throughout the peak hour who would be concurrently working to complete and submit their survey.

Each of the survey tests was then configured to repeat the necessary number of times for each concurrent user in order to achieve the target survey submissions per hour figure. In the example given above, this was 3 iterations per user in one hour.

5.2. Test Methodology

For each survey selected for testing, a specific scenario was designed in detail and documented. This scenario included the most common sequence of questions answered by most survey respondents for that survey. This was done to ensure that a reasonably realistic load was placed on the system.

The scenario containing the question and answer pattern was then recorded for each survey using HP Performance Centre Load Runner version 11. This tool is an industry standard solution for load and performance testing. This produces an automated script, used to simulate the behaviour of the end user from the point of view of their interactions with the web based eCollect system. For example, a typical scenario that may be expected of users is to login, complete a specific series of questions relevant to the sample group, and then submit the survey.

Out of twelve ABS surveys scheduled for eForm migration in June 2013, a combination of representative surveys from Economic (Business) and Population (Household) programs were selected for their characteristics in terms of providing valuable information about system behaviour under load. These characteristics included the sample population size, survey structure and the number of question fields in the survey. The number of simulated users for selected surveys was scaled upwards to account for those surveys that were not tested. The list of selected surveys for testing is detailed in the Appendix, Table 3.

The recorded scripts were then run using the HP Performance Centre load generation platform, which simulated the targeted number of concurrent users and survey submissions per hour in terms of their interactions over HTTP/HTTPS web protocols with the Blaise eCollect system.

The performance of infrastructure components of the eCollect solution such as memory, CPU, disk space and network bandwidth utilisation were monitored throughout the duration of each of the tests, and the results analysed using the analysis toolset available in HP Performance Centre solution.

6. ABS Blaise 4.8.4 eCollect Solution Architecture

The following diagram, Figure 7, shows the architectural solution overview for the Blaise eCollect platform. The diagram depicts all major solution components as well as major communication flows.

A description of each of the components is presented in the next section.

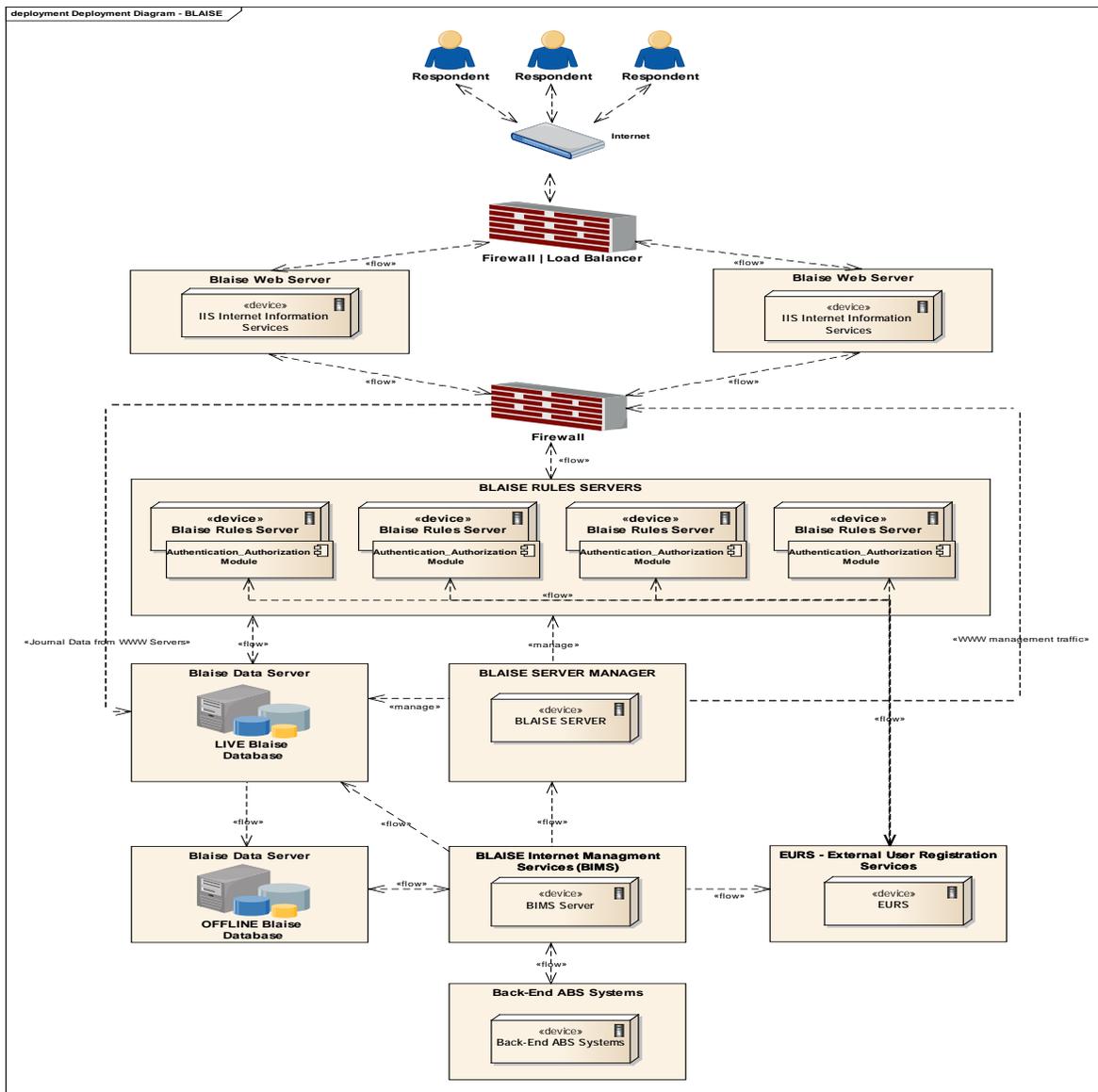


Figure 1 Blaise 4.8.4 eCollect Solution Architecture

6.1. ABS Blaise 4.8.4 eCollect Solution Components

The ABS Blaise 4.8.4 eCollect solution consists of a number of infrastructure components. It is based around the Blaise IS product and it is built using a single Blaise Park. A Blaise Park is a combination of one or more Blaise Web Servers, one or more Blaise Rules Servers, a Blaise Data Server and a Blaise Management Server. In addition the Blaise eCollect platform has a number of ABS specific components that allow for integration with ABS processing systems.

Blaise Web Servers

The Blaise Web Servers provide the presentation layer for Blaise Internet. Web Servers interact with the Blaise Rules Servers when a user navigates to the next survey page or submits a survey.

Blaise Rules Servers

The Blaise Rules Servers are responsible for executing the business logic for web surveys. For example, Rules Servers determine the next sequence of questions to be presented based on the answers received.

Blaise Management Server

The Blaise Management Server coordinates the Blaise Server Park and provides functionality for deployment and management of surveys deployed to Blaise Internet.

Blaise Data Server (Live Database)

The Blaise Data Server (Live Database) stores collected provider data for the duration of a survey.

Blaise Data Server (Offline Database)

The Offline Database stores provider response data once it has been submitted. This component is used to mitigate security risks associated with storing provider data in the Live Database.

Blaise Internet Management Services (BIMS)

The Blaise Internet Management Services (BIMS) component is an ABS built Web Service layer for eCollect that provides interfaces to survey lifecycle management. Typical operations facilitated by BIMS include, initialisation of a survey, retrieval of collection processing status and retrieval of provider response data (data extraction). BIMS integrates Blaise Internet with back-end ABS processing systems through the use of the Blaise API.

External User Registration Services (EURS)

External User Registration Services (EURS) is an ABS developed system that is used to manage provider credentials for authentication purposes.

Authentication and Authorisation Module

The Authentication and Authorisation Module is an ABS developed component. The primary purpose of this module is to provide authentication and authorisation functionality for survey respondents. It is used

in conjunction with the ABS External User Registration Services. The module is implemented as a DLL and is installed on each of the Blaise Rules Servers. This module is referenced and called through the use of Blaise ‘ALIEN’ procedure calls in Blaise instruments.

7. Test Environment

7.1. Setup

The test environment was setup as shown in the solution architecture diagram Figure 1. The server hardware components in the test setup were configured as per the details in Table 1. All servers in this environment were virtual and not physical machines. Testing included all valid network components, including firewall, routers/switches and load balancer devices.

Table 1 Test Environment Configuration

Blaise Park Component	Operating System	Software	Hardware Specification
Blaise Web Server 2 Servers	Windows Server 2008 R2	Blaise 4.8.4.1767 Microsoft Internet Information Services (IIS 7)	4 x CPUs @ 2.7Ghz Intel Xeon E5-26800 * 4GB RAM
Blaise Rules Server 4 Servers	Windows Server 2008 R2	Blaise 4.8.4.1767	2x 4 CPUs @ 2.93Ghz Intel Xeon X5570 2x 4 CPUs @ 2.7Ghz Intel Xeon E5-26800 4GB RAM
Blaise Data Server 1 Live DB Server 1 Offline DB Server	Windows Server 2008 R2	Blaise 4.8.4.1767	4 CPUs @ 2.93Ghz Intel Xeon X5570 4GB RAM 2 CPUs @ 2.93Ghz CPU Intel Xeon X5570 4GB RAM
Blaise Management Server 1 Server	Windows Server 2008 R2	Blaise 4.8.4.1767	2 CPUs @ 2.93Ghz CPU Intel Xeon X5570 2GB RAM
BIMS Server 1 Server	Windows Server 2008 R2	Blaise 4.8.4.1767 Microsoft Internet Information Services (IIS 7)	2 CPUs @ 2.93Ghz CPU Intel Xeon X5570 2GB RAM

*The number of CPUs on the Web Servers was upgraded from 2 to 4 CPUs based on results of Stress Test 1 – Section 8.4.

7.2. Load Generation

Load was applied to the Blaise eCollect system by load generators external to the ABS infrastructure environment. This was done to ensure that all ABS eCollect infrastructure components, including load balancers, firewalls and routers were tested.

7.3. Test Monitoring

All Blaise eCollect components were monitored throughout test runs by using the ABS PG3 Tool, which automatically monitors server resource usage (CPU, memory, disk and network performance).

Additionally, Windows Performance Monitor tool and verification of system logs were used to assess system behaviour under load.

8. Test Results

This section describes the tests that were conducted and the results of those tests.

This section does not detail every single test that was undertaken, as multiple iterations of tests were often undertaken, particularly if any issues were encountered during test runs. For clarity, only final and significant test results are included, in order to highlight Blaise eCollect performance characteristics and issues which were encountered. In addition to this, analysis of Blaise scalability and performance as well as issues and challenges are included in Section 10 and Section 11 respectively.

8.1. Normal (Average) Capacity Test Results

Test Parameters: 127 Concurrent Virtual users for 2 hours, target 397 submissions an hour.

Execution: 11/05/2013 between 11:25:03 -13:52:01

Objective: Normal Load was applied to Blaise eCollect system using various network speeds and benchmark end user response times for MPS, QBIS and REACS surveys.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

Key Observations:

- The total number of survey forms submitted was 795 and it was as per transaction rate of 397 submissions an hour targeted for peak load test.
- There were no errors seen throughout the execution of run.
- Response times for 90% of the transactions for submission of survey pages at 512Kbps and 2048Kbps are within acceptable SLA of 5 secs.
- Response time for 90% Login module transactions and click survey transactions at 512Kbps and 2048Kbps is within the SLA of 15 secs.

- Response times for 90% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for 90% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- CPU utilization on Web Servers was averaging 35%, 10% on the rule servers, and less than 10% on the Database server.

8.2. Endurance Test

Test Parameters: 127 Concurrent Virtual users for 8 hours, target 397 submissions an hour.

Executed: 11/05/2013 18:40:52 - 03:49:02

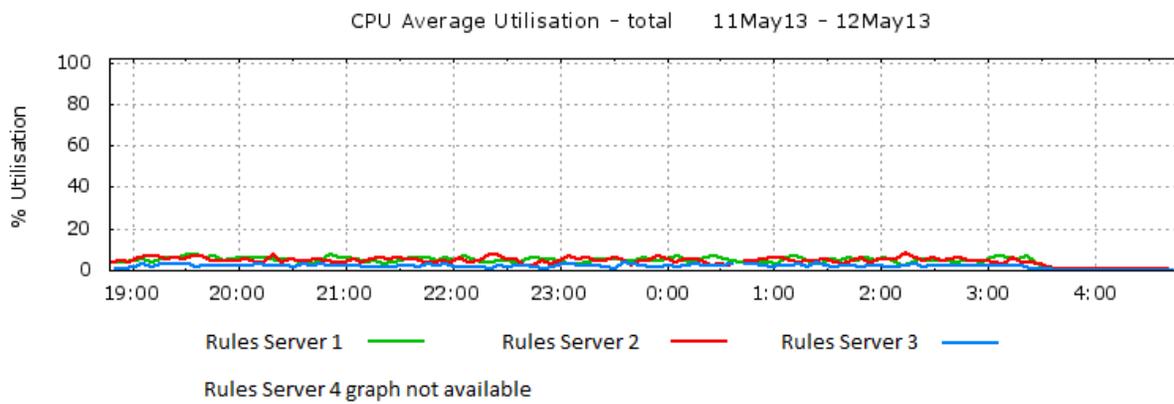
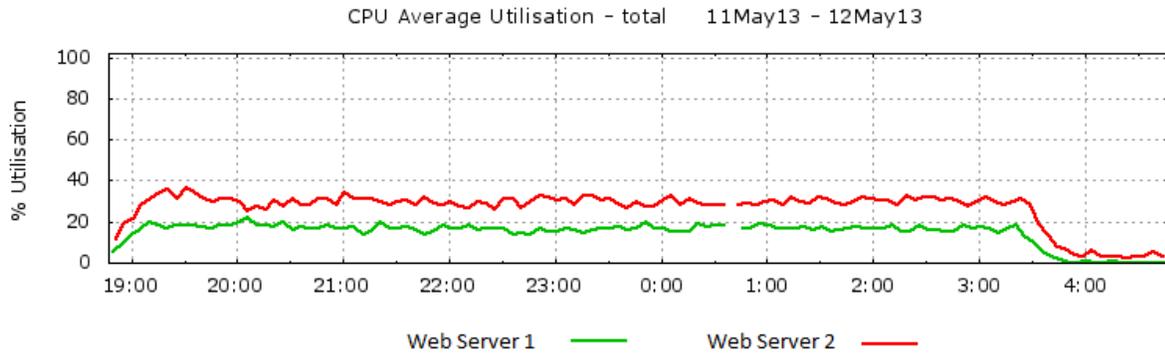
Objective: The objective of the test was to apply normal load to Blaise eCollect system for 8 hours using various network speeds and benchmark end user response times for MPS, QBIS and REACS surveys to verify if system can handle load for prolonged period.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

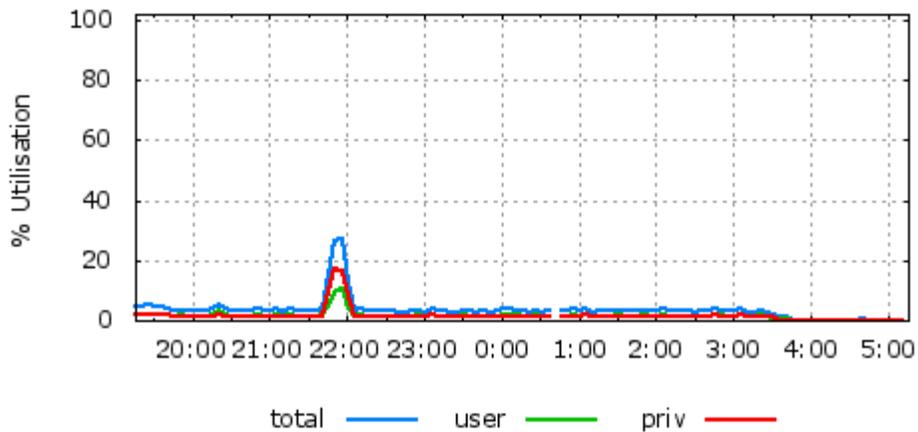
Key Observations

- The total number of survey forms submitted was 3,231 and it was as per transaction rate of 397 submissions an hour targeted for peak load test.
- No errors and transaction failures encountered throughout the duration of the run.
- There was no degradation in response times under load over the 8 hour period.
- No memory leaks were encountered during the total duration of the run.
- Response times for 90% of the transactions for submission of survey pages at 512Kbps and 2048Kbps are within acceptable SLA of 5 secs.
- Response time for 90% Login module transactions and click survey transactions at 512Kbps and 2048Kbps is within the SLA of 15 secs.
- Response times for 90% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for 90% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- CPU utilization on Web Servers was averaging 35%, 10% on the rule servers, and less than 10% on the Database server.

8.3. Endurance Test Graphs



Data Server - Live Database - CPU Average Utilisation 11May13 - 12May13



The peak at 22:00 was caused by security software updates and was not related to load testing

8.4. Stress Test 1

Test Parameters: 370 Concurrent Virtual users for 2 hours, target 1,090 submissions an hour.

Executed: 13/05/2013 between 17:42:37- 20:21:15

Objective: To apply more than 1.5 times the peak load on Blaise IS at different network speeds to verify if the system can sustain additional load without any issues for MPS, QBIS and REACS surveys.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

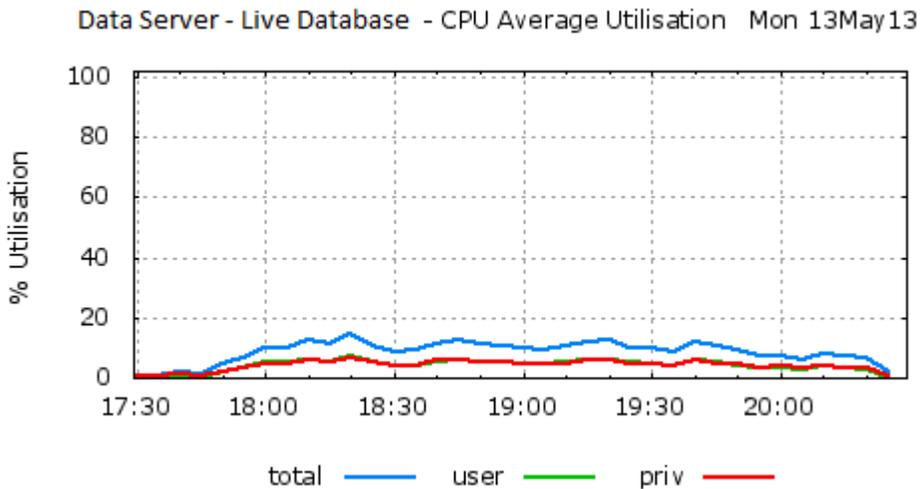
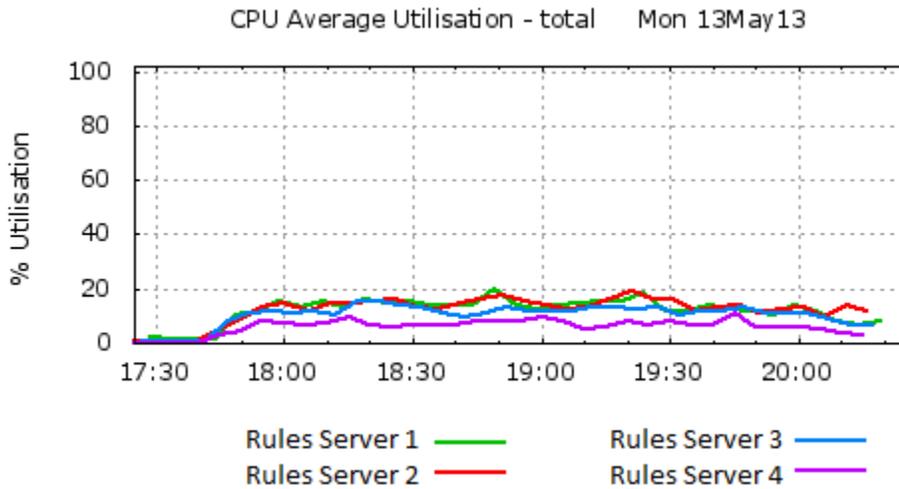
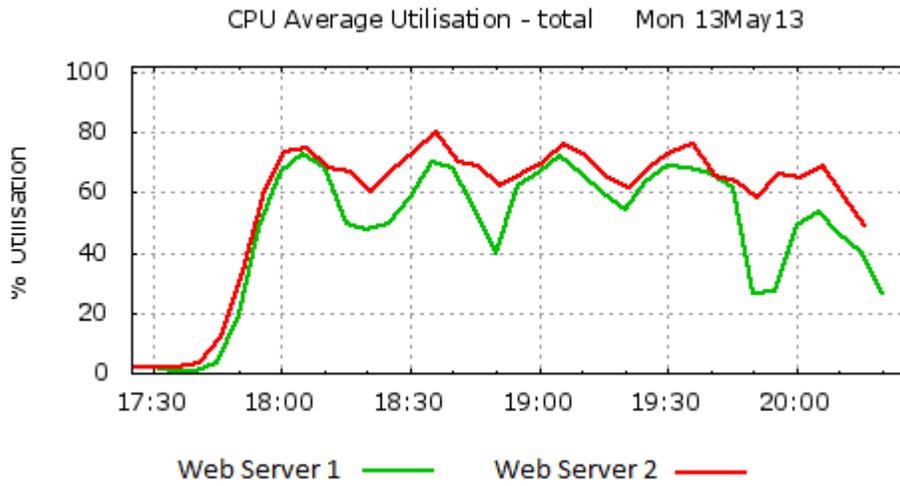
Key Observations

- The total number of survey forms submitted was 940 in one hour. This test did not achieve the target survey submissions rate.
- Total surveys submitted were in one hour.
- Many errors were detected between 19:40 - 19:52. We believe this was due to connection time-outs between the Blaise API Services3 and the Journal Database.

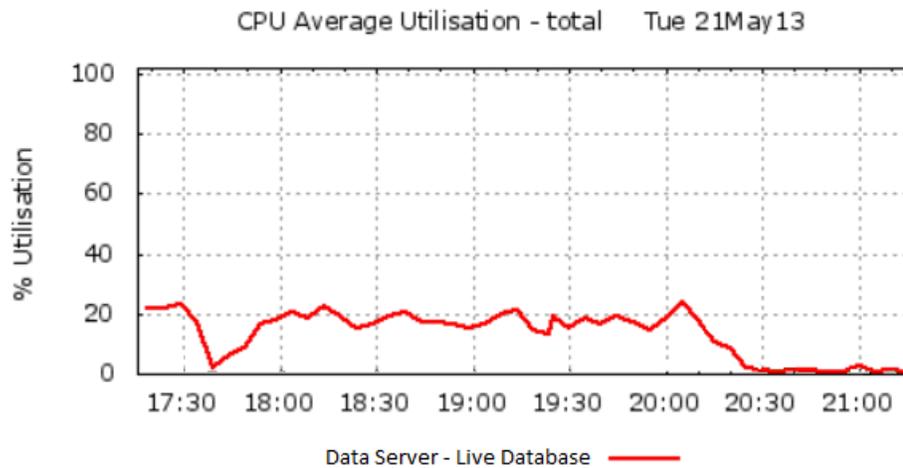
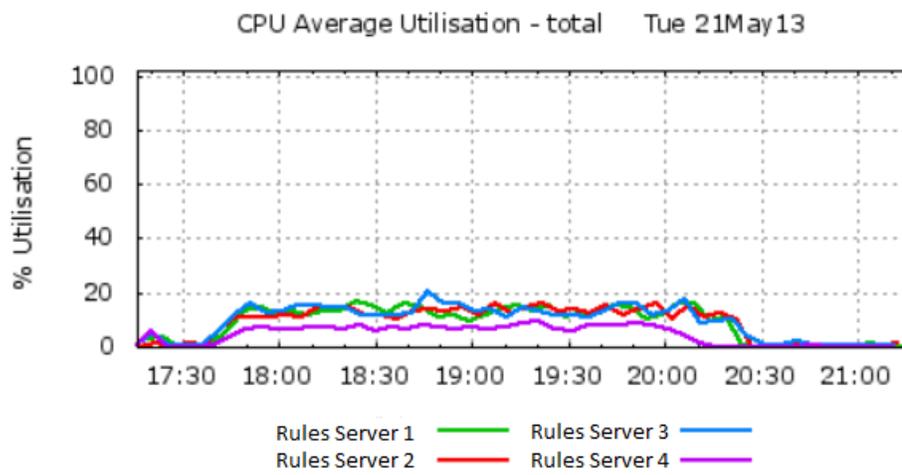
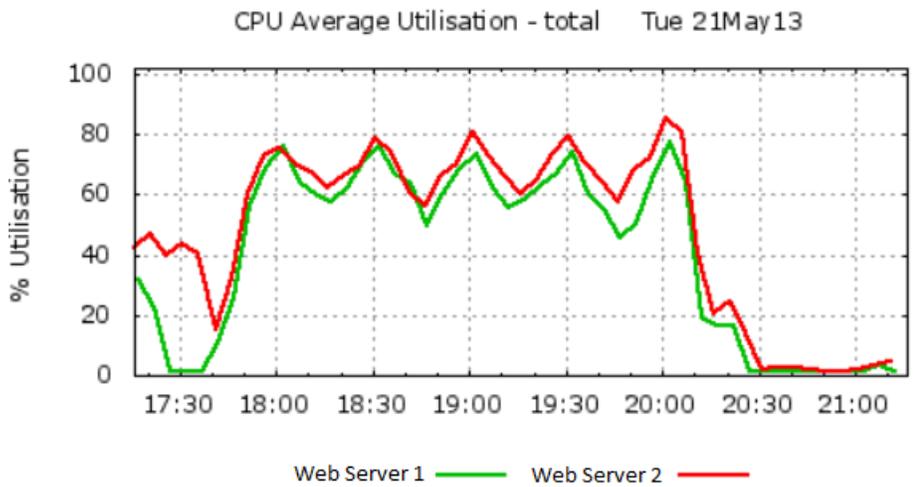
Error: BlJour3A.Journal: Could not connect to BlaiseAPIService3 (Socket Error # 10060-Connection timed out.); ErrorNumber: -2147215301.
- 1,600 TCP/IP sockets were observed in TIME_WAIT state on the Blaise Data Server. These connections were confirmed to have originated primarily from the Blaise Web Servers
- Throughput was averaging at 1.1 Mbps.
- CPU utilization on Web Servers peaked at over 80%, and was around 20% on Rules Servers and Data Server.

The cause of the errors observed in this test was identified as the large build-up of 1,600 TCP/IP sockets in TIME_WAIT state on the Blaise Data Server. The build-up of the TCP/IP sockets was due to the number of connections that the Blaise API processes on the Web Servers were making to the Data Server for Blaise journaling calls. A fix in the form of a Windows Registry setting for the TIME_WAIT value was identified through research on the internet and applied to the Blaise Data Server (MSDN 2013; IBM 2013). The stress test was subsequently re-run successfully on 21/05/2013. Further analysis and comments are available in Section 10.5.

8.5. Stress Test 1 Graphs - Failed Attempt



8.6. Stress Test 1 Graphs - Successful Attempt



8.7. Peak (Absolute) Capacity Test

Test Parameters: 221 Concurrent Virtual users for 2 hours, target 696 submissions an hour.

Executed: 04/06/2013 between 17:19:57 - 20:25:29

Objective: Capacity (absolute maximum peak day) load was applied to Blaise eCollect system using various network speeds and benchmark end user response times for MPS, QBIS, REACS, CAPEX and ECS surveys.

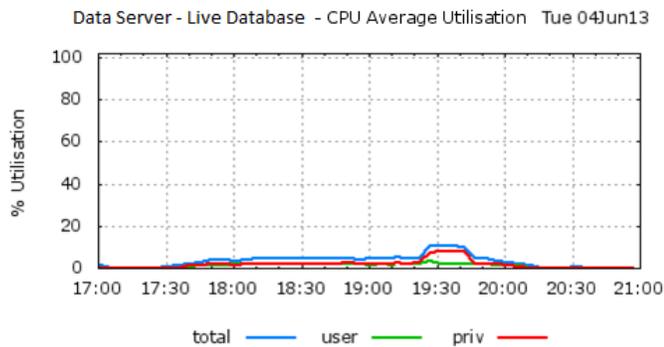
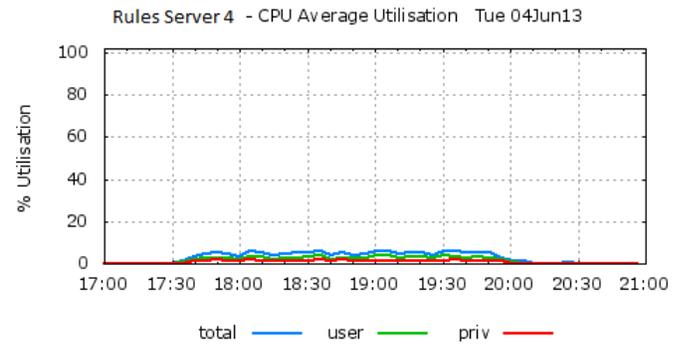
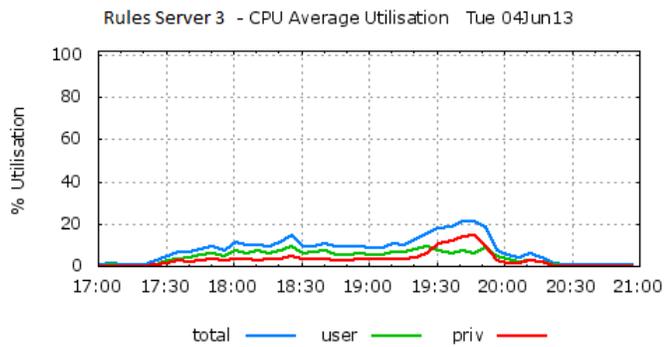
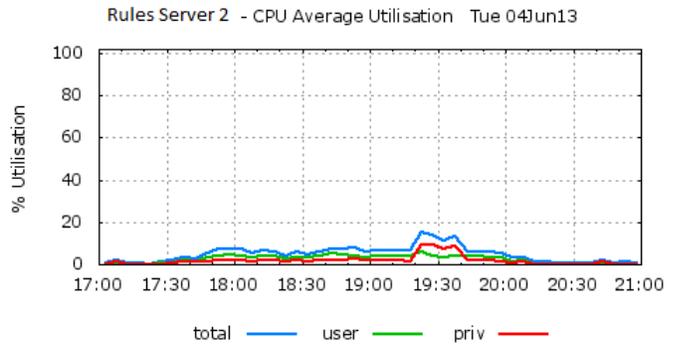
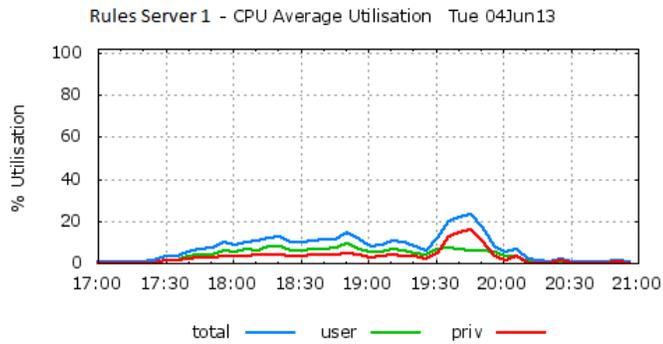
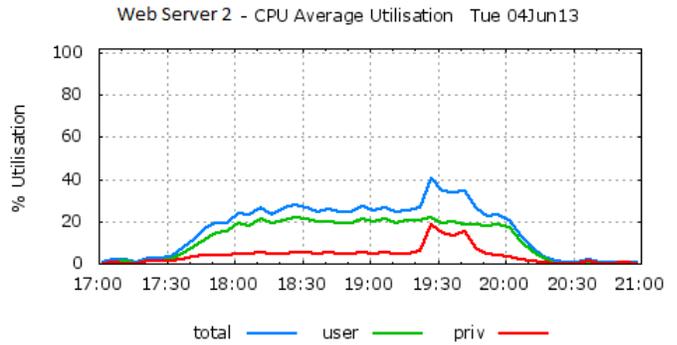
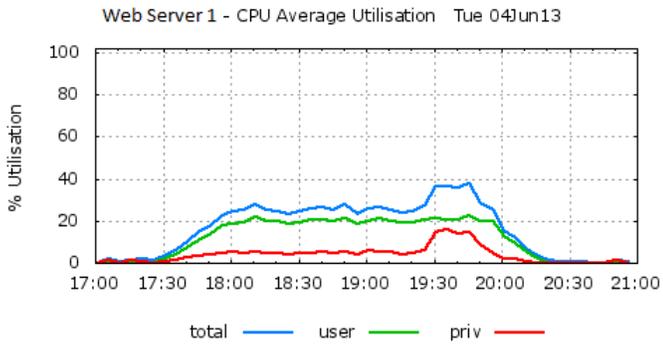
Based on results of previous stress tests the number of CPUs was increased from 2 to 4 CPUs on each of the Web Servers.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

Key Observations

- The total number of survey forms submitted was 1,540 and it was as per transaction rate of 696 submissions an hour targeted for peak load test.
- There were no errors seen throughout the execution of load test run.
- Response times for 95% of the transactions for submission of survey pages at 512Kbps and 2048Kbps are within acceptable SLA of 5 secs.
- Response time for 95% Login module transactions and click survey transactions at 512Kbps and 2048Kbps is within the SLA of 15 secs.
- Response times for 95% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for 95% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- Throughput was averaging 4.3 Mbps.
- Total transaction throughput was averaging 6.0 transactions per second.
- CPU Utilization on web servers was averaging around 30%, and as high as 40% for a period of time.
- CPU Utilization on rules and data server was less than 20%.

8.8. Peak (Absolute) Capacity Test Graphs



8.9. Stress Test 2- Excluding Authentication and Authorization

Test Parameters: 441 Concurrent Virtual users for 2 hours, target 3,097 submissions an hour.

Executed: 06/06/2013 between 21:58:03 - 00:49:31

Objective: To apply a stress test on Blaise IS using 441 concurrent virtual users and targeting 3,097 survey submissions an hour. This test was aimed at pushing the limits of the Blaise IS in its current configuration, but without the ABS authentication and authorisation module. For this test the custom ABS authentication and authorisation module was removed in order to test Blaise IS performance in its 'vanilla form' without custom modifications. The purpose of this was to see how well the platform performs without the additional load that is caused by authentication and authorization being done in Blaise.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

This test was based on a single MPS survey instrument.

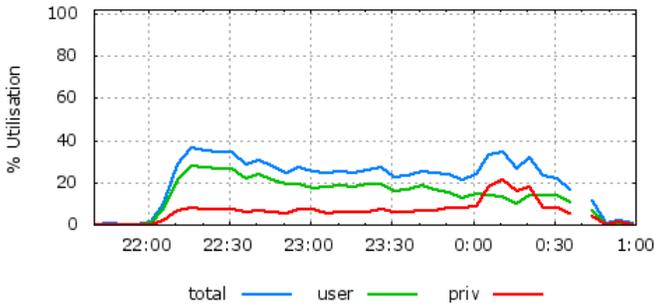
This test targeted a very high throughput of 3,307 submissions.

Key Observations

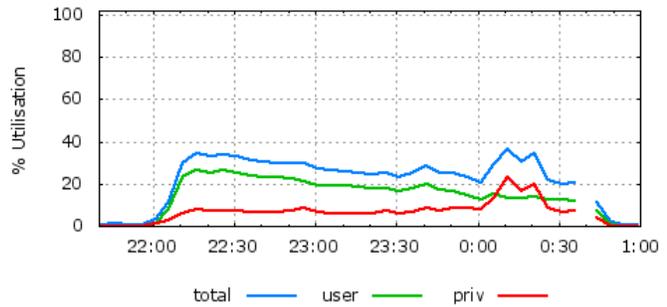
- Successful ramp up of 441 users.
- A lot of errors and failures were seen throughout the test run. These errors were due to out-of-memory errors reported on the Rules Servers. The target of 3,307 submissions per hour was not reached as there were many failures.
- Interestingly, while the out of memory errors were reported by the Blaise Rules Servers, the affected Rules Servers had a significant amount of available memory, at least 1GB on each Rules Server.
- The results from this test need to be investigated further. Additional comments on the results from this test are provided in Section 11 - Challenges and Issues.

8.10. Stress Test 2 Graphs

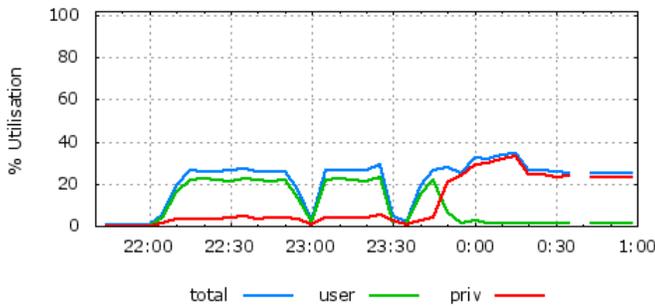
Web Server 1 - CPU Average Utilisation 06Jun13 - 07Jun13



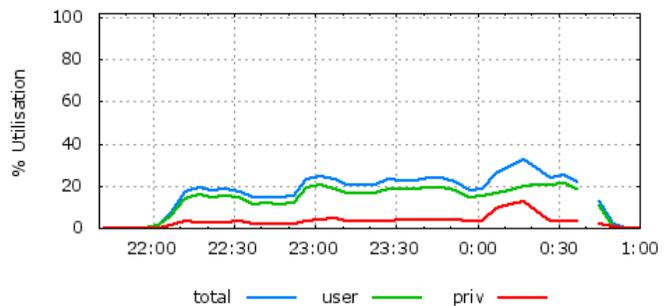
Web Server 2 - CPU Average Utilisation 06Jun13 - 07Jun13



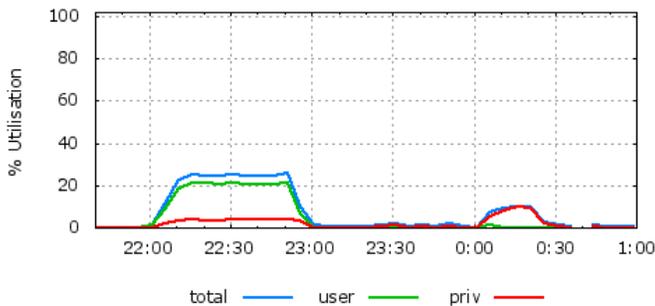
Rules Server 1 - CPU Average Utilisation 06Jun13 - 07Jun13



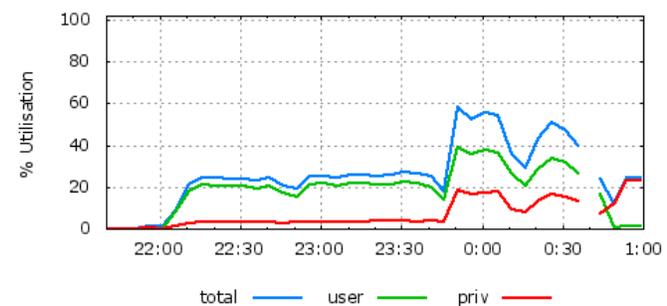
Rules Server 2 - CPU Average Utilisation 06Jun13 - 07Jun13



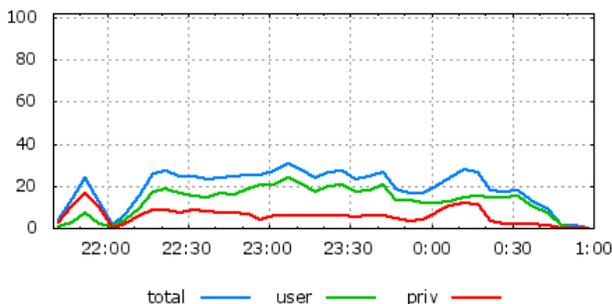
Rules Server 3 - CPU Average Utilisation 06Jun13 - 07Jun13



Rules Server 4 - CPU Average Utilisation 06Jun13 - 07Jun13



Data Server - Live Database - CPU Average Utilisation 06Jun13 - 07Jun13



The gaps in if the graphs at approximately 00:45 is due to system updates when performances metrics were not being updated. It is not related to the load and performance test.

8.11. Stress Test 3 - Including Authentication and Authorization

Test Parameters: 441 Concurrent Virtual users for 2 hours, target 1,397 submissions an hour

Executed: 11/06/2013 17:19:39 - 19:49:11

Objective: To apply twice the absolute peak load to Blaise eCollect system using different network speeds to verify if the system can sustain the load without any issues for MPS, QBIS, REACS, CAPEX and ECS surveys.

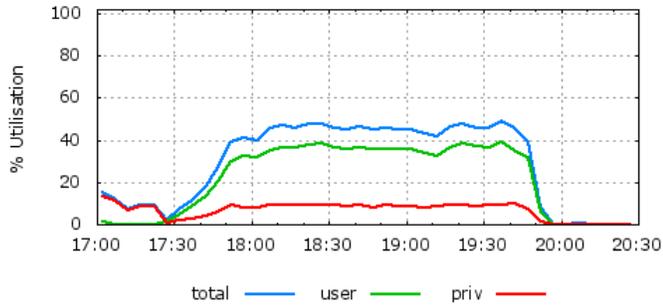
The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

Key Observations

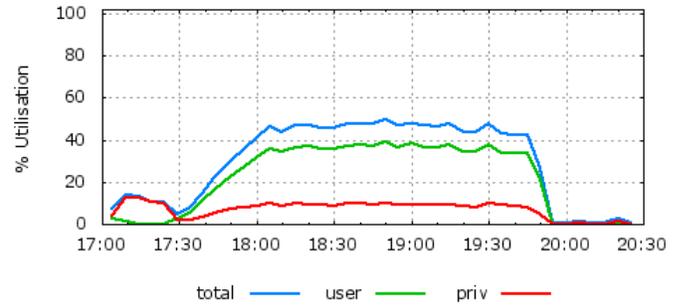
- Total surveys submitted were 2,795 and it was as per transaction rate of 1,397 submissions an hour targeted for peak load test.
- The total number of survey forms submitted was 3,231
- There were no errors seen throughout the execution of the test run.
- Response times for 95% of the transactions for submission of survey pages at 512Kbps and 2048Kbps are within acceptable SLA of 5 secs.
- Response time for 95% Login module transactions and click survey transactions at 512Kbps and 2048Kbps is within the SLA of 15 secs.
- Response times for 95% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for 95% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- Throughput was averaging at 8 Mbps.
- Total transaction throughput was averaging 13 transactions per second
- CPU Utilization on web servers was averaging around 50%.
- CPU utilization on rule servers was averaging at 30% and on data server was 20%

8.12. Stress Test 3 Graphs

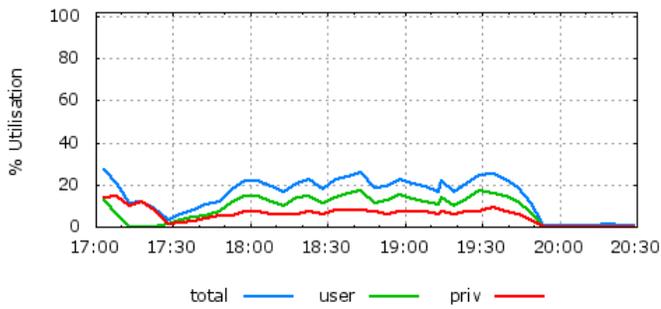
Web Server 1 - CPU Average Utilisation Tue 11Jun13



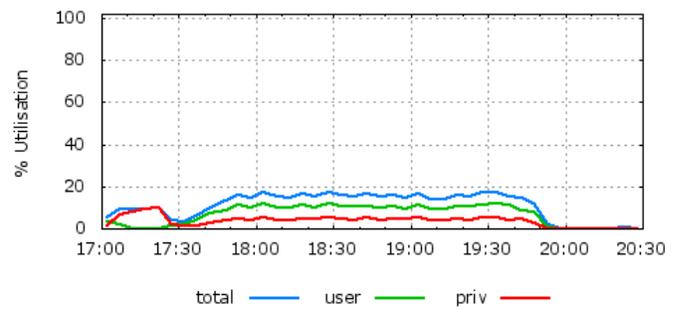
Web Server 2 - CPU Average Utilisation Tue 11Jun13



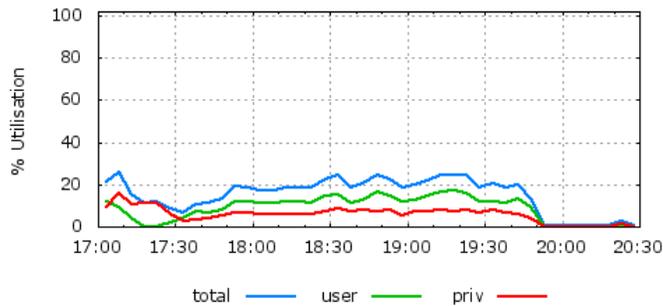
Rules Server 1 CPU Average Utilisation Tue 11Jun13



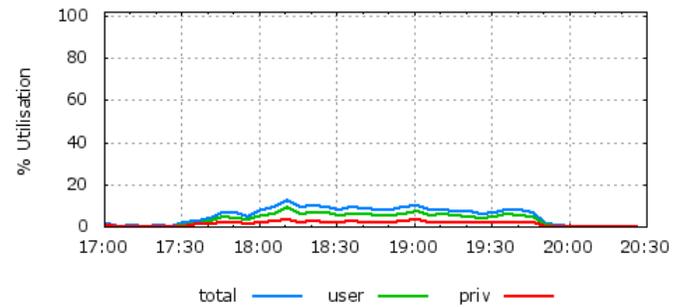
Rules Server 2 - CPU Average Utilisation Tue 11Jun13



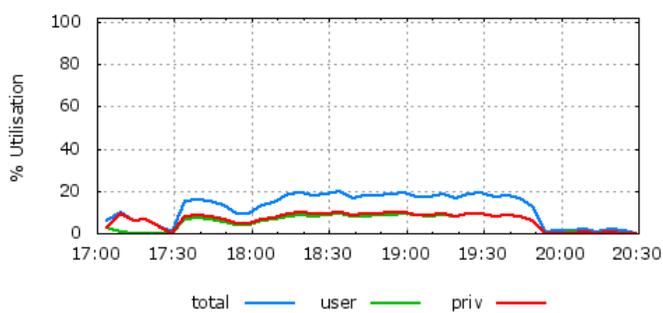
Rules Server 3 - CPU Average Utilisation Tue 11Jun13



Rules Server 4 - CPU Average Utilisation Tue 11Jun13



Data Server - Live Database - CPU Average Utilisation Tue 11Jun13



8.13. Data Extraction Test

Test Parameters: 221 Virtual users for 2 hours + Data Extraction, target 696 submissions an hour.

Executed: 03/06/2013 between 18:00:08 - 21:05:42

Objective: To apply the absolute peak load to Blaise eCollect system using various network speeds and run data extraction process in parallel to the load in order to verify the effect of data extraction on the end user response times and also to validate the performance of the data extraction module. For this test, the peak load was run for a full 1 hour before data extraction was run. This was done to create the appropriate number of records (survey submissions) for the expected peak load, and is based on the assumption that data extraction will be run every hour. It needs to be noted that data extraction is implemented as an incremental process, meaning that survey submissions that were previously extracted are not re-extracted on subsequent runs.

The following network speeds were applied: 56Kbps, 64Kbps, 512Kbps and 2048Kbps.

Key Observations

- Total surveys submitted were 1,685 and it was as per transaction rate of 696 submissions an hour targeted for peak load test.
- There were no errors seen throughout the execution of load test run.
- The data extraction module was able to handle 1 hour of data in less than 2 minutes and had negligible impact on front end system performance.
- On average it took 20 seconds to extract 300 records (survey submissions).
- Response times for 95% of the transactions for submission of survey pages at 512Kbps and 2048Kbps are within acceptable SLA of 5 secs.
- Response time for 95% Login module transactions and click survey transactions at 512Kbps and 2048Kbps is within the SLA of 15 secs.
- Response times for 95% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for 95% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- Throughput was averaging 4.48 Mbps.
- Total transaction throughput was averaging 4.0 transactions per second.
- CPU Utilization on web servers was averaging around 20%.

- CPU Utilization on rules and data server was less than 20%.

9. Summary of Results

The following is the overall summary of the results obtained through load and performance testing:

- Response times for at least 90% of the transactions for submission of survey pages at 512Kbps and 2048Kbps were within acceptable SLA of 5 seconds.
- Response times for at least 90% Login module transactions and click survey transactions at 512Kbps and 2048Kbps were within the SLA of 15 seconds.
- Response times for at least 90% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for at least 90% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.
- The data extraction module was able to handle 300 records (survey submissions) in under 20 seconds and had negligible impact on front end user experience.
- Blaise eCollect system was able to meet the defined twice the Peak Load for 2 hours without any performance degradation with 441 concurrent users achieving 1,397 survey submissions an hour.
- Blaise eCollect system was able to meet the defined Normal Load for prolonged period of time of 8 hours without any performance degradation such as memory leaks, user response time degradation, or any other observed negative trend with 127 users. The number of survey submissions achieved in 8 hours was 3,391.
- CPU utilization was averaging around 50% on Web Servers, 30% on Rule Servers and 20% on the Data Server for the highest load tests. On all servers memory available was sufficient throughout the test. These metrics were for the stress test at 441 concurrent users and 1,397 submissions an hour.
- In Stress Test 1, which included 370 concurrent users and a target rate of 1,090 submissions an hour, a lot of errors were observed due to the large build-up of 1,600 TCP/IP sockets in TIME_WAIT state on the Blaise Data Server. The build-up of the TCP/IP sockets was due to the number of connections that the Blaise API processes on the Web Servers were making to the Data Server for Blaise journaling calls. A fix in the form of a Windows Registry setting for the TIME_WAIT value was applied to the Data Server and the stress test was re-run successfully.
- In Stress Test 3, which included 441 concurrent users and a target rate of 3,307 survey submissions an hour a lot of and failures were seen throughout the test run. These errors were due to out-of-memory errors reported on the Rules Servers. Interestingly, while the out of memory errors were reported by the Blaise Rules Servers, the affected Rules Servers had a significant amount of available memory, at least 1GB on each Rules Server. The results from this test need

to be investigated further. Additional comments on the results from this test are provided in Section 11 - Challenges and Issues.

10. Scalability Observations and Analysis

10.1. Web Servers

In the solution architecture for Blaise eCollect, a specialised F5 load balancer was used to distribute the load to the web servers. In all the tests conducted it was observed that the load on the web servers was well distributed and evenly spread.

Also, it should be noted that the number of CPUs on each of the Web Servers was upgraded from 2 to 4 CPUs based on results of Stress Test 1 - Section 8.4, where CPU utilization on the Web Servers was observed to peak at 80%. The additional CPUs resulted in a much lower CPU utilization in further stress tests, which peaked at only 50% utilization for the same load.

Rules Servers

In the solution architecture for Blaise eCollect, the built-in Blaise IS load balancing feature was used to distribute the load between the rules servers. This feature was configured to “round-robin” load balancing setting. In all tests conducted, it was observed that the load on the rules servers was fairly well balanced and reasonably well distributed, although in some cases load distribution was not completely even.

In some of the earlier tests conducted the focus was to establish how well Blaise IS scales with additional server resources. Early on it was observed that it was the Rules Servers that took on a lot of the computational load. For example, the 2 web servers were 20% CPU utilised while the 2 rules servers were above 40% CPU utilised. Due to these observations, the number of web servers was fixed at 2 servers. The number of rules servers was varied from 1 to 3 to establish how well the performance of the rules servers scales under load. This number was increased per test setup until errors or time-outs were encountered. Based on those tests it was concluded that Blaise IS performance scales reasonably well and performance scalability appears to be linear with additional Rules Servers. The results of those tests are shown in Table 2.

Table 2 Blaise Rules Servers Scalability

Web Servers	Rules Servers	Max. Concurrent Users	Total Survey Submissions per hour
2	1	20	100
2	2	40	200
2	3	60	300

10.2. Data Server

In terms of Blaise IS performance and scalability in a single Blaise Park, the Data Server is the limiting factor. This is because, there can only be one Data Server per Blaise Park. The data server is also the

single point of failure in the solution architecture, as there are no secondary data servers that can take-over in case of primary data server failure.

In terms of scalability the limitation of a single data server can be overcome by utilising more than one Blaise Park, however this increases the overall solution complexity, as any points of integration, such as for example with an organisation's back-end processing systems will need to take into account multiple Blaise Parks. The multiple park approach will also present significant additional challenges for extremely large surveys, that is, surveys with large sample sizes, which cannot be run on single Blaise Park.

10.3. Blaise 4 - 32bit Memory Limits

As Blaise 4 runs as a 32bit Windows process there is also a practical limitation on the amount of virtual memory available to the system. In a 32bit Windows environment this is limited to 2GB (3GB under special compile time configuration). While this limitation was not encountered in any of the tests conducted to date, it is likely that the Data Server will eventually reach this limitation under a high load. It is also possible that this limit has the potential to affect the Rules and Web servers. However, this can be easily mitigated by spreading the load to additional web and rules servers.

10.4. TCP/IP Socket Build-Up on Data Server

In one of the earlier stress tests, it was observed that under a high load with 370 concurrent users targeting 1,090 submissions an hour, the users experienced a large number of time-out errors, towards the end of the 2 hour test. These errors were caused by the requests to the system that could not be completed in less than 120 seconds. The reason for this was identified as the large build-up of 1,600 TCP/IP sockets in TIME_WAIT state on the data server. The cause of this was the number of connections that the Blaise API processes on the Web Servers were making to the Data Server for Blaise journaling calls.

On further investigation it was established that the Windows 2008 R2 server on which the Data Server was hosted has a default windows registry setting of 4min for TCP/IP sockets to remain in TIME_WAIT state after the connection is closed by the client. The purpose for this is to allow any belated network packets to arrive after the connection has been closed, however some industry vendors recommend changing the default to a much lower setting of 30 seconds to improve server performance (MSDN, 2013; IBM, 2013).

The default value for the TIME_WAIT sockets was changed to 30 seconds on the data server and the stress tests were re-run successfully without any errors. The change to this setting allowed the overall throughput performance to improve significantly, up to 441 concurrent users and 1,397 submissions an hour. The number of sockets observed in TIME_WAIT state with 441 concurrent users was approximately 700.

It is however likely that this will be one of the early performance limitations in Blaise 4. Based on observations before and after the setting changes to the TIME_WAIT value, it is estimated that the limit of concurrent users could be as high as 900 and the number of hourly submissions about 3,000. However, this is only a prediction and has not been tested.

A potential longer term solution to the socket build-up issue is to implement a connection pooling mechanism, whereby connections between server components are not closed after each use but rather kept

in a connection pool and reused when needed. This technique is used in a large number of enterprise industry solutions including relational database systems (Wikipedia, 2013).

11. Challenges and Issues

11.1. Authentication and Authorisation in Blaise

The Authentication and Authorisation module in the Blaise eCollect solution presented unique challenges from the point of system performance. This module is an ABS developed component primarily used to allow Blaise instruments to interact with other non-Blaise system components. The module is implemented as a .Net C# DLL and is installed on each of the Blaise Rules Servers. This module is referenced and called through the use of Blaise 'Alien' procedure calls in Blaise instruments. The module relies on the use of the Blaise Database and DatabaseManager objects which are used to connect to and query Blaise database files.

One of the issues that was encountered with this approach to authentication and authorisation was with the concurrent calls to the module causing instability on the Blaise Rules servers and causing the Blaise API process (BlAPI3S.exe) to crash under concurrent access calls from as low as 10 simultaneous users. This issue was identified and rectified with concurrent access locking mechanisms available in .Net C#. The module has now been re-tested to support up to 500 concurrent user requests.

Given that this authentication and authorisation approach relies on calls to the Blaise database and Blaise Data Server, it places an additional load on the eCollect system. Therefore there is currently work being undertaken by ABS to utilize a dedicated authentication mechanism based outside of Blaise.

11.2. Slow performance at 56kbps and 64kbps dial-up

The transaction response times for end-user dial up connection speeds of 56kbps and 64kbps were found to be relatively high, as outlined in section.

- Response times for at least 90% of the transactions for submission of survey pages at 56Kbps and 64Kbps exceed SLAs of 5 seconds, and are in the range of 10-20 seconds, and as high as 40 seconds.
- Response times for at least 90% of the Login module transactions and click survey transactions at 56Kbps and 64Kbps exceed 15 seconds and are as high as 80 seconds.

Investigations are currently being made into improving the performance at these end-user connection speeds. This effort is focused around using the Google Closure Tools for JavaScript optimization as well as HTTP traffic compression on web servers and load balancer devices (Google, 2013).

11.3. Access Violation and Out of Memory Errors

One of the tests that was run focused on evaluating the performance of the Blaise IS solution in its 'vanilla' form. That is without the inclusion of an authentication and authorisation module developed by ABS. This test was aimed at assessing the raw performance of Blaise as a way of comparing the additional load the authentication and authorisation done through Blaise places on the system. The parameters for this test included 441 concurrent users, and survey submissions up to 3,307 an hour, over a two hour period. This test was also based on a single MPS survey instrument.

The test ran well for 30 minutes, at this point one of the Blaise Rules Servers experienced serious failures. The errors manifested as time-out errors to the end-users, where the request to process a survey operation could not be completed within 120 seconds. However, on the Blaise Rules servers the errors appeared as Windows Application errors in the Windows Application Event logs. There were several hundred of these errors reported in the logs.

The following is a short extract of the errors and the full errors list is available in the Appendix.

Error 1: TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Access violation at address 00401E6F in module 'BIAPI3S.exe'. Read of address 00000000 E.ErrorCode: -2147192832 E.ErrorSource: Database: 898817088 Catastrophic: true

Error 2: 10:24:44.462 TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Out of memory E.ErrorCode: -2147192832 E.ErrorSource: Database: 373684400 Catastrophic: false

This test was repeated two times with the same overall results each time. On the second run through, two of the four Blaise Rules servers experienced these errors and the simulated users failed to complete the test scenarios.

Interestingly, while the out of memory error was reported by the Blaise application, the affected Rules Servers had a significant amount of available memory, at least 1GB.

The results from this test need to be investigated further. There were a number of key parameters in this test that need further exploration. Specifically, this test had a higher target of 3,307 survey submissions an hour which could be a possible explanation for the observed errors. In addition, this test used a single MPS survey instrument, where previously, multiple surveys were used, and MPS users represented a much smaller proportion of all survey respondents. The MPS has a large and complex hierarchical question structure which can potentially have an impact on memory utilization with a large number of survey respondents.

The findings from this test were shared with the Blaise team from Statistics Netherlands. Currently, there is no confirmed explanation for these errors. However, the Statistics Netherlands team and ABS are investigating this issue further.

12. Conclusion

In conclusion, the Blaise 4.8.4 eCollect platform performs well at 512kbps and 2048kbps end-user network speeds, meeting the performance targets of 5 seconds for survey navigation transactions and 15seconds for login transactions. The platform performs well for concurrent users, supporting 441 concurrent users for 2 hours and achieving over 1,397 submissions an hour. The system also performed well in endurance tests, achieving a throughput of 3,307 submissions in 8 hours with 127 concurrent users and maintaining responsive transaction times and stable performance characteristics.

Challenges still remain in improving system performance for dial-up connections where transaction times were reported at up to 40 seconds for survey navigation transactions and 80 seconds for login transactions. In addition, the out-of-memory errors observed at a higher level of survey submission throughput are an issue and will require further investigation and resolution. Finally, the scalability of a Blaise Park is challenged by the single Data Server, which is further compounded by issues seen with end-user request time-outs caused by TCP/IP socket build-up from journaling API calls between Blaise Web Servers and the Data Server at higher levels of system load. Whilst additional Blaise Parks may allow further scalability, this may present additional complexities for surveys with very large numbers of respondents.

Challenges and issues encountered during the test program will need to be considered and addressed, particularly in the areas of the out of memory reports seen, and poor performance using slower connection speeds.

Overall, the load and performance test program allowed the ABS to gain valuable insights into performance, capacity and scalability of the Blaise 4.8.4 eCollect platform. Based on the test results it was concluded that the eCollect platform will have sufficient resources and scalability to support the ABS June 2013 quarter eForm survey migration goals. Furthermore, there is also potential for further scalability to support additional ABS eForm surveys into the future.

13. References

1. Google, Google Closure JavaScript Optimiser, Accessed: 22/07/2013, <https://developers.google.com/closure/>
2. MSDN, Avoiding TCP/IP Port Exhaustion, Accessed: 22/07/2013, [http://msdn.microsoft.com/en-us/library/aa560610\(v=bts.20\).aspx](http://msdn.microsoft.com/en-us/library/aa560610(v=bts.20).aspx)
3. IBM, Configuring Windows for high network connection rates, Accessed: 22/07/2013, <http://publib.boulder.ibm.com/infocenter/cicstg/v6r0m0/index.jsp?topic=%2Fcom.ibm.cicstg600.doc%2Fcc11a10264.htm>
4. Wikipedia, Connection Pool, Accessed: 23/07/2013 http://en.wikipedia.org/wiki/Connection_pool
5. ABS, Internet Activity Survey Dec 2012, Accessed: 22/07/2013, <http://www.abs.gov.au/ausstats/abs@.nsf/mf/8153.0/>

14. Appendix

14.1. Surveys Selected for Testing

Table 3 Surveys Selected for Testing

Survey	Expected eForm Take-up	Number of Question Fields	Reason to include in testing
MPS (Monthly Population Survey)	11,000	400	Representative of population/household survey structure. Monthly cycle applies more frequent load on the system
CAPEX (Capital Expenditure Survey)	8,648	26	Quarterly survey, chosen as representative of business survey structure, with large sample size.
QBIS (Quarterly Business Indicators Survey)	16,691	50	Quarterly survey, chosen as representative of business survey structure, with large sample size.
ECS (Engineering Construction Survey)	4,409	140	Quarterly survey, chosen as representative of business survey structure, with large sample size.
REACS (Rural Environment and Agricultural Commodities Survey)	35,000	470	Annual survey. Chosen for its length (number of questions) and large sample size

14.2. Internet Connection Speeds

Table 4* Internet Connection Speeds

Speed	No. of Connections	Percentile %
56Kb-dial-up-modem	282,000	2.32
64Kb-ISDN	6000	0.5
512Kb-Satellite (900ms latency)**	92,000	0.76
512Kb-ADSL	4,727,000	38.42
2048Kb-Other	7,060,000	58
Total	12,167,000	100

*This table was based on the ABS Internet Activity Survey Dec 2012. <http://www.abs.gov.au/ausstats/abs@.nsf/mf/8153.0/>

**Note: 512Kb- Satellite (900ms latency) could not be simulated via LoadRunner 11 for load and performance tests.

14.3. Out of Memory Error Logs

Windows Event Application Log 1
<p>Log Name: Application Source: BlaiseAPIService3 Date: 05/06/13 20:18:57 Event ID: 1001 Task Category: None Level: Error Keywords: Classic User: N/A Computer: rulesserver Description: 10:18:57.587 TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Access violation at address 00401E6F in module 'BIAPI3S.exe'. Read of address 00000000 E.ErrorCode: -2147192832 E.ErrorSource: Database: 898817088 Catastrophic: true Event Xml: <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event"> <System> <Provider Name="BlaiseAPIService3" /> <EventID Qualifiers="0">1001</EventID> <Level>2</Level> <Task>0</Task> <Keywords>0x8000000000000000</Keywords> <TimeCreated SystemTime="2013-06-05T10:18:57.000000000Z" /> <EventRecordID>13830</EventRecordID> <Channel>Application</Channel> <Computer>rulesserver</Computer> <Security /> </System> <EventData> <Data>10:18:57.587 TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Access violation at address 00401E6F in module 'BIAPI3S.exe'. Read of address 00000000 E.ErrorCode: -2147192832 E.ErrorSource: Database: 898817088 Catastrophic: true</Data> </EventData> </Event></p>

Windows Event Application Log 2

Log Name: Application
Source: BlaiseAPIService3
Date: 05/06/13 20:19:02
Event ID: 1001
Task Category: None
Level: Error
Keywords: Classic
User: N/A
Computer: rulesserver
Description:
SaveToStream: Access violation at address 00401E6F in module 'BlAPI3S.exe'. Read of address 00000000
Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 <System>
 <Provider Name="BlaiseAPIService3" />
 <EventID Qualifiers="0">1001</EventID>
 <Level>2</Level>
 <Task>0</Task>
 <Keywords>0x8000000000000000</Keywords>
 <TimeCreated SystemTime="2013-06-05T10:19:02.000000000Z" />
 <EventRecordID>13831</EventRecordID>
 <Channel>Application</Channel>
 <Computer>rulesserver</Computer>
 <Security />
 </System>
 <EventData>
 <Data>SaveToStream: Access violation at address 00401E6F in module 'BlAPI3S.exe'. Read of address
00000000</Data>
 </EventData>
</Event>

Windows Event Application Log 3

Log Name: Application
Source: BlaiseAPIService3
Date: 05/06/13 20:24:44
Event ID: 1001
Task Category: None
Level: Error
Keywords: Classic
User: N/A
Computer: rulesserver
Description:
10:24:44.462 TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Out of memory E.ErrorCode: -2147192832 E.ErrorSource: Database: 373684400 Catastrophic: false
Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 <System>
 <Provider Name="BlaiseAPIService3" />
 <EventID Qualifiers="0">1001</EventID>
 <Level>2</Level>
 <Task>0</Task>
 <Keywords>0x8000000000000000</Keywords>
 <TimeCreated SystemTime="2013-06-05T10:24:44.000000000Z" />
 <EventRecordID>13855</EventRecordID>
 <Channel>Application</Channel>
 <Computer>rulesserver</Computer>
 <Security />
 </System>
 <EventData>
 <Data>10:24:44.462 TBIAPIManager.ParseXMLDoc. E.Message: Unrecognized exception: Out of memory E.ErrorCode: -2147192832 E.ErrorSource: Database: 373684400 Catastrophic: false</Data>
 </EventData>
</Event>

Windows Event Application Log 4

Log Name: Application
Source: BlaiseAPIService3
Date: 05/06/13 20:24:45
Event ID: 1001
Task Category: None
Level: Warning
Keywords: Classic
User: N/A
Computer: rulesserver

Description:
IdTCPServerExecute: Out of memory

Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 <System>
 <Provider Name="BlaiseAPIService3" />
 <EventID Qualifiers="0">1001</EventID>
 <Level>3</Level>
 <Task>0</Task>
 <Keywords>0x80000000000000</Keywords>
 <TimeCreated SystemTime="2013-06-05T10:24:45.000000000Z" />
 <EventRecordID>13895</EventRecordID>
 <Channel>Application</Channel>
 <Computer>rulesserver</Computer>
 <Security />
 </System>
 <EventData>
 <Data>IdTCPServerExecute: Out of memory</Data>
 </EventData>
</Event>

Generating Blaise from DDI

Gerrit de Bolster, Statistics Netherlands

July 28, 2013

1. Introduction

Within the Blaise community there is much interest for the Data Documentation Initiative (DDI). Already several efforts have been made to connect Blaise with DDI. Based upon suggestions from ABS and being in the possession of a CAWI instrument generator a working Proof Of Concept (POC) was created to generate operational CAWI instruments from a DDI 3.1 instance. The structure of the DDI 3.1 instances created within this (POC) are mainly based on samples available from the DDI community. Furthermore a DDI 3.1 instance structure was created to define Blaise datamodels (even including a nested Block structure and arrays) including data views and the data itself. A Maniplus was developed to generate the Blaise datamodel from this DDI instance. This paper will give a small introduction to DDI and explain in more detail the structure of the DDI instances and how they link to the Blaise language.

2. What is DDI?

On the web page of the DDI Alliance the following definition is found:

*“The **Data Documentation Initiative (DDI)** is an effort to create an international standard for describing data from the social, behavioural, and economic sciences.”*

From Wikipedia, the free encyclopedia:

*“The **Data Documentation Initiative (DDI)** is an international project to create a standard for information describing statistical and social science data (i.e., metadata). Begun in 1995, the effort brings together data professionals from around the world to develop the standard. The DDI specification, written in XML, provides a format for content, exchange, and preservation of information. Version 3.1 of the DDI standard was released in 2009. DDI fills a need related to the challenge of storing and distributing social science metadata, due to the ubiquity of proprietary file formats and no international standard for the design of codebooks.”*

Started as a standard for metadata for all kind of documents since version 3 the concept of a data collection instrument was introduced. This makes DDI very interesting to use as a standard in a data collection architecture where Blaise is used as the tool. It also supports concepts that can be used as a repository including codebooks. The physical implementation of DDI is a set of XML-schemas.

3. Why using DDI?

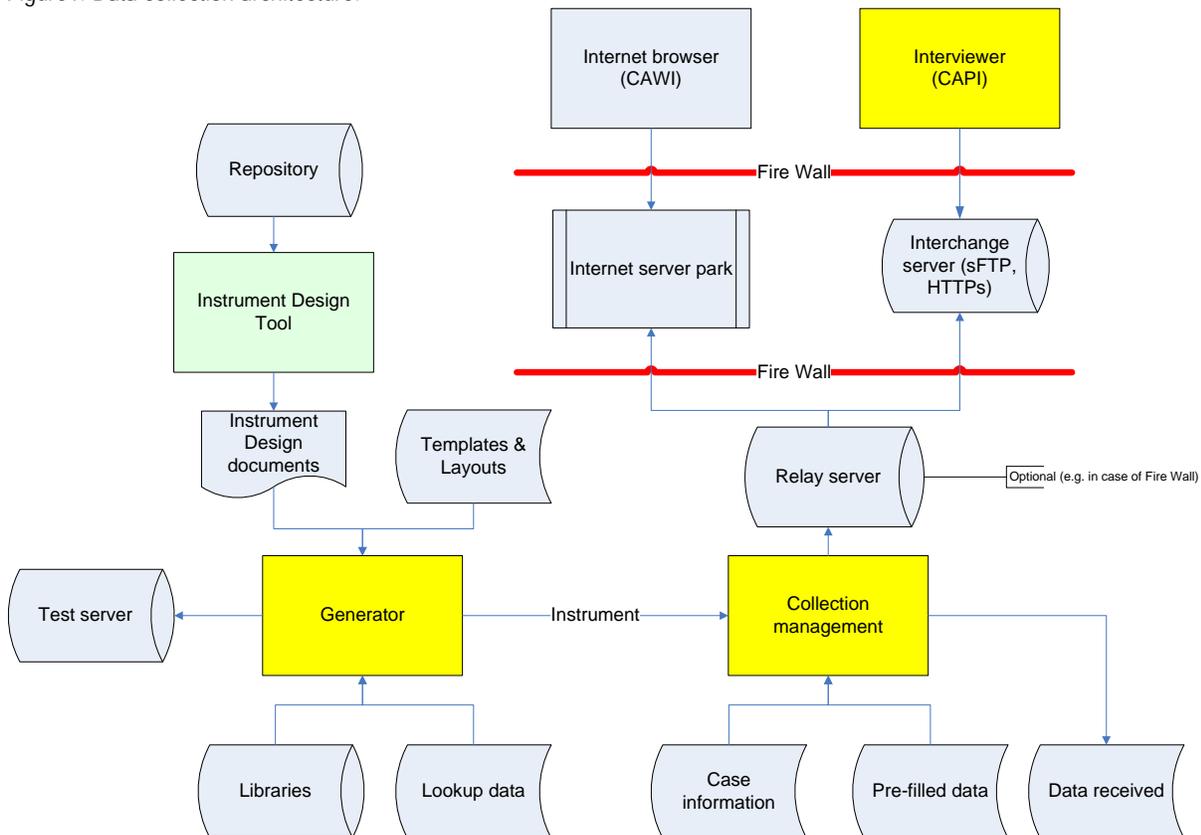
In the Blaise community the involved organisations are already for some time looking for a standard to describe a data collection instrument in a more functional language to communicate between different disciplines in the field of data collection. Although the majority probably use their own standard for documentation a few of them are focussing on DDI. DDI has the advantage to be an open standard expressed in the global supported standard XML. This makes the standard also interesting for commercial software vendors (e.g. Colectica) to create solutions for DDI. For data collection institutions like NSI's this means that not all the software needed to manipulate DDI has to be created by themselves.

The University of Michigan used DDI for the output definition of their MQDS (Michigan Questionnaire Documentation System). ABS (Australia) is already for some time experimenting with DDI as the standard for their QDT (Questionnaire Design Tool). INSEE (France) is also looking for the use of DDI in relation to Blaise data collection instruments. Triggered by these initiatives I started end of last year a hobby project developing a POC for generating Blaise instruments from a DDI version 3.1 (the current version) instance. Therefore this paper focusses on DDI as a meta data source for data collection instruments.

4. The architecture

On first sight DDI looks like a clear path to standardisation of meta data definition of data collection instruments. However, there are different approaches possible. First of all (and some organisations are following this approach) you can try to express the Blaise source language directly in DDI. As Blaise is a meta language on its own the added value of this approach is questionable. The goal of the POC was to create a generic DDI instance for a data collection instrument and to apply a separate generator to create a Blaise instrument. As a BlaiseIS instrument generator was already available the POC was limited to create CAWI instruments. The idea is that in the future any kind of data collection instrument in any kind of programming language can be created from the same generic DDI definition. The future data collection architecture could look like shown in figure 1. Only CAWI and CAPI are presented but CATI and CADI should also be included. This architecture could fit in a “plug and play” initiative which is now trendy among a number of the statistical institutes.

Figure1. Data collection architecture.



In the POC the “Instrument Design Documents” were created as DDI 3.1 instances with a text editor (EditPad Pro 6). The existing generator (BLS-Wizard) was extended to be able to use these instances for input. The generated CAWI instruments were deployed on the test server.

5. DDI, a very open standard

To learn about the possibilities and restrictions of DDI 3.1 a number of examples were examined at the start of the POC. Most of them were downloaded from the Internet but also a result from MQDS and an example from ABS were taken into account. Looking at these examples it became soon clear that the “standard” DDI was very open. In other words: there were a lot of ways a data collection instrument could be defined in the XML-structure and still verify against the XML-schemas. After several attempts the basic structure as was found in the examples downloaded from DDI Alliance related web sites was taken as they seemed the best fit.

The BLS-Wizard generator supported already an instrument definition in an ASCII-file. Elaborating on that definition and the chosen basic structure a DDI instance was created as well as a Maniplus script to convert the content of the XML-file into the ASCII-file. A Maniplus script to read any kind of XML and convert it into flat ASCII was already available too. Including these Maniplus scripts in the workflow of the generator a working CAWI instrument was generated and installed on the test server. This instrument contained a simple IF-structure as well as some enumerated questions. After that success more (even multilingual) instrument specifications were defined in the DDI structure including download and upload portals. The majority of the specifications fitted into the DDI 3.1 definition without a problem. However, in some cases some definitions were forced beyond their boundaries to be able to store the necessary information. Some generic definitions had to be used too. Where possible an attribute was included which generator (BLS-Wizard) was the target. This attribute is only used for documentation. Finally all the meta specifications could be placed in the DDI XML-tags.

The current meta specifications are partly based on the specifications as used by the BLS-Wizard generator. In their turn these specifications are based partly on the Blaise language, e.g. in the case of formulas in calculations and checks. But formulas can be defined in another open standard called EBNF (Extended Backus–Naur Form ISO/IEC 14977). This will make the DDI instance more generic and less programming language dependent. The generator should then convert the EBNF formulas into the grammar of the used programming language.

We should take into account that the target language can have influence on the definition of the instrument. As we know the Blaise language is very rich. This can result in functions that are not supported in other programming languages for electronic forms such as XForms or even HTML/JavaScript solutions. These conflicts are to be solved by the generator even as this results in lesser functionality in the resulting instrument. Perhaps adding a target language attribute at more places in the DDI definitions can be helpful.

6. DDI, a technical or a functional standard?

One thing is very clear: an DDI instance is almost unreadable for human beings! As XML-files are normally already hard to read these XML-definitions are moreover infested with internal references (see figure 2). An enumerated question which takes a few lines in the Blaise language consist of many XML-tags referring to other tags in the DDI instance. And these referring tags are not even grouped together but

spread throughout the XML-file. Probably this has been done to support the reuse of enumeration specifications within the DDI instance. A small instrument soon occupies hundreds of lines in a DDI instance.

DDI is a meta data standard and therefore functional. However, as it is expressed in XML the implementation is technical. Thanks to this technical implementation it can be used to generate instruments from it using software. In the POC a text editor (EditPad Pro 6) was used to create the DDI instances but for really extended and robust use a more sophisticated instrument design tool is a must. This instrument design tool could be developed by the a commercial software vendor or the Blaise community (like the Questionnaire Design Tool from ABS) or in some kind of consortium involving both parties. Such a tool should be using a repository and (as much as possible) a WYSIWYG user interface. The output of that tool should be the complete DDI instance of a data collection instrument. The Blaise community can then be leading in creating the generator for the Blaise language (versions 4 and 5), the software vendor for other languages. This could be profitable for everyone involved.

Figure 2. A lot of references.

```
<d:ControlConstructScheme id="E9AF1D4D-391A-4AE3-8ECD-AAD64ABDB394">
  <d:ControlConstructSchemeName>Keywords*Calculations</d:ControlConstructSchemeName>
  <d:ComputationItem id="A7A325DC-5557-4BAB-AC56-B7B2C1D473CB">
  <d:ComputationItem id="C3B32DA7-9E1B-4006-BAE2-BC7382D80FD9">
  <d:ComputationItem id="3824BFAF-15F4-4082-9658-A8118355C086">
  <d:ComputationItem id="6C2A33E2-1726-42E1-AD81-D0906AF1779E">
  <d:ComputationItem id="E3572295-6F75-44D2-A0C7-D3EC17E77701">
  <d:ComputationItem id="BA25D33F-564C-4EC8-A556-2BCAF9E7BC9F">
</d:ControlConstructScheme>
```

7. DDI alone is not enough

Not all of the features for a CAWI instrument can or should be defined in the DDI instance. As is shown in figure 1 several data sinks (Templates, Layouts, Libraries and Lookup data) are linked directly to the generator. The BLS-Wizard generator uses the predefined templates for the different type of instruments like questionnaire, election form, download/upload portal etc. Colour schemes and logos are also included in the templates as well as the division of the window in separate panels. The libraries contain predefined procedures e.g. to access the Google Maps API and web services but also to check an e-mail address or telephone number. These all belong to the environment of the generator as they are programming language specific.

Another major issue is the layout of the questions. In Blaise 4 the layout definitions are stored in the Mode Library, in Blaise 5 in a resource data base. Other programming languages will store the layout definitions in a complete other form like CSS (Cascading Style Sheets). However, in the instrument design definition (the DDI instance) the instrument designer should be able to define which layout must be applied as this can have methodological consequences. In the POC the name of the field pane in the Mode Library was included in the DDI instance. Of course this is programming language dependent and therefore not the wright solution. For the moment it could be improved by defining a generic list of layout definitions with standard names that the generator links to the technical definition in the layout repository (like the field pane in the Mode Library). A final open standard solution must yet be found.

In the POC the lookup data files were already present in Blaise format as the generator was expecting this. In chapter 9 is described that these lookup files can also be defined in DDI and created in Blaise format using a Maniplus script as was developed during the POC.

8. Some technical details

The basic container used in the POC for a DDI instance for data collection is a <StudyUnit>. Besides some minor tags it contains the main tags <DataCollection> and <LogicalProduct>. The <LogicalProduct> tag contains the values and texts for the enumerated and set questions. In DDI 3.1 the values (codes) and texts are stored in separated containers and connected by references. Within the main tag <DataCollection> the questions and the rules are stored. The questions in the container <QuestionScheme> and the rules in a set of linked <ControlConstructScheme> containers. A separate <ControlConstructScheme> container is used for storing the so-called (BLS-Wizard language) settings of the CAWI instrument. These settings cover amongst others languages and libraries used, navigation buttons and external file references.

Figure 3. The basic structure of a DDI instance for a data collection instrument.

```
<?xml version="1.0" encoding="UTF-8"?>
<ddi:DDIInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ddi:instance:3_1 instance.xsd"
  <s:StudyUnit id="AC7403C9-A62E-4A51-8E84-BC07BE695A7F" version="1.0.0" versionDate="2013-02-11" xml:lang="EN">
  <r:Citation>
  <s:Abstract id="CF46E258-BEE6-446A-8410-8C8BBEFC9568">
  <r:UniverseReference>
  <s:Purpose id="15325127-4078-4460-A183-28846A9F883E">
  <d:DataCollection id="CDEC074C-578B-41A0-ABCB-1BC9D30AE2DB">
  <!-- Mod1 AuxiliaryData -->
  <d:QuestionScheme id="78036F0C-C1B6-478A-896C-EB172B117CFD">
  <!-- Mod1 Questions -->
  <d:QuestionScheme id="296C0855-8D5D-4DF6-AED3-A997223E2E61">
  <!-- Settings -->
  <d:ControlConstructScheme id="2DD3B538-7C94-48E0-A614-0A58E25F21A2">
  <!-- Modules -->
  <d:ControlConstructScheme id="42629456-405E-4F2B-B8E8-CDDF1918AA82">
  <d:ControlConstructSchemeName>Modules</d:ControlConstructSchemeName>
  <!-- Mod1 -->
  <d:Sequence id="8F182644-C0D0-47AF-B3B3-CD43D99E998C">
  </d:ControlConstructScheme>
  <!-- Mod1 Conditions -->
  <d:ControlConstructScheme id="A2D5EE49-D2F4-4417-ACDF-2B7653C4C631">
  <d:ControlConstructSchemeName>Mod1*Conditions</d:ControlConstructSchemeName>
  <d:ComputationItem id="D468C9CC-8F0A-44C7-9264-7613DD9826D0">
  <d:ComputationItem id="40767C1B-BA79-4244-8AF9-1B39CAF8DBEE">
  </d:ControlConstructScheme>
  <!-- Mod1 Calculations -->
  <d:ControlConstructScheme id="D46DBA05-460F-4A7B-AF4A-BE44E07AB014">
  <!-- Routing -->
  <d:ControlConstructScheme id="D9BA6CDD-A508-48B6-AE04-1ABF8B684170">
  <d:Instrument id="BLST7">
  </d:DataCollection>
  <l:LogicalProduct id="7AF835FE-1B69-4DB9-B551-C92D2577A36F">
  <!-- Temp1 -->
  <l:CategoryScheme id="31726B21-E291-414C-BC10-18A3BC475386">
  <!-- Currency1 -->
  <l:CategoryScheme id="2F52AA28-2E6D-41E4-BB26-F910DA5A0BB8">
  <!-- Temp1 -->
  <l:CodeScheme id="0AB865D2-8BBA-42A3-95D5-ED60B6FE6366">
  <!-- Currency1 -->
  <l:CodeScheme id="CFB7017C-031C-494A-9844-3C66B41FE233">
  </l:LogicalProduct>
  </s:StudyUnit>
</ddi:DDIInstance>
```

The common CDATA-container was applied in most of the string type tags to avoid conversion of XML unfriendly signs like “<” and “>”. Texts were defined in UTF-8.

Figure 4. The use of a CDATA container.

```
<d:ComputationItem id="BA25D33F-564C-4EC8-A556-2BCAF9E7BC9F">
  <d:ConstructName>6</d:ConstructName>
  <d:Code>
  <r:Code programmingLanguage="BLS-Wizard"><![CDATA[OccCode2 = &ISCO08DS<aLangName>(Occupation2_a).Code08]]></r:Code>
  </d:Code>
</d:ComputationItem>
```

9. Defining Externals in DDI

In questionnaires external lookup files are often used. In a Blaise instrument these externals are Blaise files based upon Blaise data models. Although a Blaise instrument is also based on a data model there are differences. As a data collection instrument is mainly meant for reporting data the purpose of an external is to retrieve data to be used in a data collection instrument or script. Question text is therefore normally not present as it is not used but a filled data file is a requirement. Even though in the Blaise language the same type of meta data is used (a data model) in the POC a different DDI definition was applied. The main container of the definition of the DDI instance is in this case a <ResourcePackage>. Within this main container a number of sub containers can occur. These containers hold the definition of BLOCKS and the data. One container is holding the definition of the data model on the highest level.

Within a BLOCK another BLOCK can be included by using the tag <VariableSchemeReference>. Inside the BLOCK a concatenated reference is needed to put in on the route with a local name. A tag <VariableReference> is referring to a tag <VariableGroup> which contains the reference to the included BLOCK and its local name. A FIELD is defined by the tag <Variable>.

The data records are stored in the sub container <RecordLayoutScheme>. Each record is stored within this container with the tag <DataSet>.

It is obvious that defining even simple externals in DDI is complex. It becomes even more complex as enumeration and set fields are involved. In that case one or more sub containers <CategoryScheme> and <CodeScheme> must be included too.

Also in this case an user friendly tool is an absolute prerequisite to be able to define an external with or without data. In the POC a separate Maniplus setup was created to translate these DDI definitions to a Blaise data model and related data file.

Figure 5. The basic structure of a DDI instance for an external with data.

```
<ddi:DDIInstance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="ddi:instance:3_1 instance.xsd"
  <g:ResourcePackage id="0AF7CB03-825B-48C2-8BB5-BE5F7125FFBA">
    <g:Purpose id="Purpose">
      <l:VariableScheme id="DAFAB082-12B0-4593-980C-AD02C6F38556" version="1.0.0" versionDate="2013-02-17">
        <r:UserID type="section">DATAMODEL</r:UserID>
        <l:VariableSchemeName>Country</l:VariableSchemeName>
        <l:Variable id="1B50ACB4-423D-462B-BC7D-B341372C40C9">
          <l:Variable id="D28F0142-3300-4ACA-A984-41C271956085">
            <l:VariableGroup id="55693A59-6A0A-4D19-8BEB-2D19B6BB2218">
              <l:VariableGroup id="0C1A4229-22CF-47A0-A050-43C2EDF38696e*A">
                <l:VariableGroup id="0C1A4229-22CF-47A0-A050-43C2EDF38696e*T">
                  </l:VariableScheme>
                <pd:RecordLayoutScheme id="F194FF94-A5EB-4AC3-BE0C-5BC077E1948B">
                  <ds:DataSet id="22F48994-3A53-48D7-B4E3-90E4C21AE160">
                    <ds:DataSet id="9F8AD17D-AEFF-4A84-A920-AFE1FC056A6E">
                      <ds:DataSet id="4E191354-7EE6-41AA-AE86-BCAA8356A8CF" version="1.0.0">
                        <ds:DataSet id="529A71E0-6F22-4D7D-A5D3-64F7679FCF5C" version="1.0.0">
                          </pd:RecordLayoutScheme>
                        </ds:DataSet>
                      </ds:DataSet>
                    </ds:DataSet>
                  </ds:DataSet>
                </l:VariableGroup>
              </l:VariableGroup>
            </l:Variable>
          </l:Variable>
        </l:VariableSchemeName>
      </l:VariableScheme>
    </g:Purpose>
  </g:ResourcePackage>
</ddi:DDIInstance>
```

10. Compatibility with newer versions: a must

In the past the Dutch government initiated several projects to create a standard for the data collection at enterprises by the different governmental organisations including Statistics Netherlands. The main issue was to involve the vendors of administration software and accountant organisations. One of the first attempts appeared to be successful. Software vendors included at their own costs the standard in their

administrative software. Then, before it was even taken into production, a next project changed the definitions making the already developed software useless. From that moment on for a lot of years software vendors were not so willing anymore to implement any governmental standard in their software unless someone else was paying for it. A basic demand for the survival of a standard is compatibility between versions.

At this moment DDI version 3.1 is still the current version. A DDI 3.2 version is still under development but the provisional XML-schemas are already available. When a simple DDI 3.1 instance was taken from the POC and verified against the XML-schemas of version 3.2 not less than 365 errors occurred! Such an incompatibility between 2 MINOR versions of a standard is an attempt to commit suicide.

11. Conclusion

The lesson that was learned from this exercise is that, if we (the Blaise community) really want a standard instrument definition in DDI some kind of manual should be produced describing how to apply DDI for this purpose. The DDI instances created in this POC can be used as a start. We should also feed the DDI Alliance with proposals for improvement of existing definitions and besides that filling in some gaps. As some members of the Blaise community already are involved in the DDI Alliance this should not be a problem.

The use of DDI for instrument definitions will not be successful without the necessary user friendly tools. A text editor (as EditPad Pro 6) is totally insufficient for the job.

Subsequent DDI versions of DDI should be compatible as much as possible. Only then organisations are willing to invest in their systems to adopt this powerful standard. The DDI Alliance should be very aware of that.

12. References

1. The home page of the DDI Alliance:
<http://www.ddialliance.org>
2. The Wikipedia link:
http://en.wikipedia.org/wiki/Data_Documentation_Initiative
3. Sample of DDI instances created in the POC:
https://www.dropbox.com/s/3c6hx2oq1ax4uyu/POC%20samples_DDI.zip

Automatic Generation of Blaise Data Models

*Saliha Zayoum and Lars Peter Jørgensen, Interview Service division, Statistics Denmark
Leif Bochis Madsen, IT Center, Statistics Denmark*

Abstract

The survey division of Statistic Denmark has utilized a system to speed up the process of authoring Blaise questionnaires.

The system is a compromise between on the one hand, an easily adoptable way of describing the questionnaire for customers and methodologists unaware of Blaise syntax and, on the other hand, a document format that is manageable for automatic processing.

Customers and methodologists can describe questions, answers and filters grouped in tables in a MS Word document, questionnaire designers can add information about web form layout etc. and the system may automatically produce a Blaise data model almost ready for testing and deployment.

The system has proved efficient as it has reduced the time needed for production of Blaise questionnaires considerably.

1. Introduction

The Interview Service division (IS) is a central unit in Statistics Denmark responsible for collection of data via telephone interviewing and via combined telephone and web interviewing. Interview tasks may be requested by customers – external as well as internal (within our organization). Our external customers include public institutions, universities, organizations, private companies and EU.

The primary purpose of using auto generation is to standardize and optimize our procedures. The auto generation results in a faster set-up of Blaise data models for the use of combined CATI/CAWI surveys. As a basis of further processing an MS Word document is produced – structured mainly with the aim of auto generation in mind.

Another important purpose is to involve the customer in the detailed specification of the questionnaire and to have the costumer make sure that the content of the survey is correct and accepted before the set-up. For this purpose, the produced document also serves as a reference document as part of the agreement between customer and IS.

The different types of customers participate in the development of questionnaires at different levels of involvement. Typically, EU delivers a questionnaire which is already final and cannot be changed. In these situations we copy the received questionnaire into the word document as a preparation of the Blaise set-up.

Other customers deliver a non-finished questionnaire which we can comment on and make corrections. The development process is a matter of cooperation between the customer and survey experts at Statistics Denmark.

1.1 Why Word?

The format of the reference document has evolved over some years of experiments with auto generation using various formats of Excel spreadsheets and Word documents. We decided to use Word documents instead of using e.g. Excel spreadsheets, mainly because of the widespread usage and because we wanted a tool that was simple enough to expect our customers to use it without too much instruction. Word is a well-known tool and some of our customers are not as familiar with spreadsheets as they are with word processing. Furthermore it is possible to specify e.g. formatted text to be inserted in the electronic questionnaire automatically. In spreadsheet documents there may be limitations on the size and formatting of text.² There are no practical limits on the amount of space using MS Word and the set of constructions we use in the documents are widely supported also by any other word processing tool. In the end it is a compromise between different purposes, where ease of use is considered most important.

1.2 Flow of document

Users of the reference documents are:

- 1) Customers
- 2) Survey and questionnaire designers (IS)
- 3) Blaise questionnaire developers (IS)

The document is used by different users who have the opportunity to contribute to the design of the questionnaire/survey. The different kinds of users, however, typically contribute with different levels of detail.

Our customers are limited to only describe the content of the questionnaire. They are also able to note their wishes of the web-design of the survey. E.g. if they want a special set-up as a group table or a matrix they can make a note of that.

We receive a non-finished questionnaire from the customer which our survey designer critically comments on. Afterwards the customer receives the document with the purpose of accepting the questionnaire.

The document is usually sent back and forth with comments and corrections between our internal survey designer and the customer and sometimes the Blaise programmers before the set-up in Blaise.

Our customers usually do not have any Blaise experience and it makes it difficult for them to describe the routing of the questionnaire. In those cases we construct the routing on their behalf to make it easier.

The Blaise programmers may add/correct technical details before using it for auto generation. Of course, a part of this role is to make sure that the field names and the logic (filters etc.) are correctly defined.

² Recent versions of MS Excel allow larger texts as well as formatting (bold, e.g.). However, it is not attractive to depend on the products and versions of the office packages our customers use.

2. Description of the document

The document must be structured in accordance with the following definition:

The document consists of tables. Each table describes a section (block) of questions (fields) in the questionnaire (datamodel). The tables consist of rows and columns.

The tables are structured so the first row describes the block and the other rows describe the fields in the block.

2.1 Table

Fundamentally a table is divided into 6 columns:

1. Fieldname
2. Text
3. Type name
4. Type definition
5. Filter (condition)
6. Comments

Example: A table defining a block with one question

BlockFieldName				Filter (optional)	Remarks
FieldName	Question Text	TypeName	Type Definition	Filter (optional)	Remarks

2.2 Definition of Blocks

The first line in the table describes the block and the content of the cells will result in the following generation:

```
BLOCK B<BlockFieldname> "<Text>"
FIELDS
  <...description of the fields...>
ENDBLOCK

FIELDS
  <BlockFieldname> : B<BlockFieldname>

RULES
  IF <Filter> THEN <BlockFieldname> ENDIF
```

Or if no filter is defined:

```
RULES
  <BlockFieldname>
```

The text in the second column is often omitted for block definitions. Type name (third column) and type definition (fourth column) are irrelevant regarding block fields. The type name is always produced as a concatenation of "B" and the given field name. It is possible to denote a keyword "TYPE" at the second line of the first column in order to identify blocks that are used more than once. These blocks may then be

referenced in the type name column of field rows in other tables. Certain layout instructions may occur in the third column.

2.3 Definition of Fields

Generation of field definitions follows a different pattern. Rules are generated in the same way as blocks but fields are generated as follows:

```
FIELDS  
  <Fieldname> "<Text>" : <Type name>
```

If a type definition is given the following will be generated:

```
TYPE  
  <Type name> = (<Type definition>)
```

The type can be enumerated as well as a numeric interval.

2.3.1 The first column (fieldname)

The first section (line) in the cell is the fieldname. The content must follow the rules for field names. Typically it consist of letters and numbers e.g. A1, A1_1a and so forth. Blaise cannot accept numbers alone as a fieldname or a fieldname starting with a number. Our customers usually assign the fieldnames and we only make changes if necessary.

The additional lines are ignored, but later on they can be used to define different versions of the same question, e.g. a version only for CAWI/CATI or versions of different languages besides Danish.

2.3.2 The second column (text)

The text is transformed into question text in Blaise syntax. Empty lines between the texts are transformed as line breaks.

Italics, bold, underline are transformed into Blaise text code (@B, e.g.).

In the second column we and our customers are able to define the final set-up of the text, line breaks etc.

2.3.3 The third column (type name)

The first section (line) is transformed into type name and is used directly in the field definition. If you have a question with the option Yes/No (YN) it means that YN is a type name. Instead of a type name it is also possible to indicate other type statements such as STRING, STRING[250], 1..100, Datetype, Open etc.

Furthermore a comma will result in an end of the type name. E.g. you can define a type name as YN, DONTKNOW; YN, REFUSAL or YN, EMPTY. The type name will be YN and the rest will be used as an attribute to the field.

The second section (line) can be used to modify the type name with e.g. a SET OF or ARRAY[1..10] OF. This modification will be placed before the type name in the field definition. Also, it must be in

compliance with the Blaise syntax. If there is content in the third section (line) it will be used as an instruction to the layout for CAWI. Note that it is the name of the field pane which can be noted in the third section.

An example of the content in the third column:

```
MyAnswerType, EMPTY  
SET [3] OF
```

”MyAnswerType” is the field type and the field definition will be produced as:

```
SET [3] OF MyAnswerType, EMPTY
```

Another example:

```
MyAnswerType, DONTKNOW  
  
DropDown
```

The field will have the type name MyAnswerType and respondents may answer the question with DONTKNOW. Furthermore a layout instruction “At fieldname FIELDPANE DropDown” will be created. Notice that the second line is empty. As mentioned the second line is reserved for SET OF and/or ARRAYs.

2.3.4 The fourth column (type definition)

The type definition can be omitted, but it will typically be used in combination with the third column to define enumerated types or numerical intervals which will be re-used.

During the definition of enumerated types each section (line) will be used as an answer. Empty sections (lines) will be ignored.

If a numerical value is defined in a bracket in the beginning of the line e.g. “(0) None”, the answer category will be assigned the given code zero in the type definition.

An example of the content in the fourth column:

```
None  
1-4  
5-10
```

If a type name is assigned in the third column e.g. “MyAnswerType”, a type will be defined as:

```
MyAnswerType =  
  (s1_None "None",  
   s2_1_4 "1-4",  
   s3_5_10 "5-10")
```

And the answers will be given the code values 1, 2 and 3.

Another example:

```
None  
(5) 1-4  
5-10
```

If a type name is assigned to "MyAnswerType" in the third column, a type will be defined as:

```
MyAnswerType =
(s1_None "None",
s5_1_4 (5) "1-4",
s6_5_10 "5-10")
```

And the answers will be given the code value 1, 5 and 6.

Notice that any automatic numbering of answers in the word document will be ignored e.g.:

0. None
1. 1-4
2. 5-10
- 3.

And will be transformed as shown in the first example.

2.3.5 The fifth column (filter)

A filter defines a condition whether a question should be asked or not. A filter can be defined in a simple syntax where code names and code values can be used, e.g.:

- a) Question5 = Yes
- b) Question5 = 1
- c) Question6 = 1-4
- d) Question6 = 1-2, 5, 7-9

If we presume that Question5 is defined with the field type name YN, then (a) and (b) will be interpreted in the same way. Again, if we presume that Question6 is an enumerated field with at least 9 answer categories, then (c) and (d) will be interpreted as answer quantities and transformed into:

- c) IF Question6 IN [answer1..answer4] THEN
- d) IF Question6 IN [answer1.. answer 2, answer 5, answer 7.. answer 9] THEN

If you have a filter for the whole block, it can be defined in the first row of the table e.g.:

Example: Definition of a block with an introduction and a question

BlockB	Discrimination			Citizenship <> Danish	The questions in BlockB are not asked of Danish citizens
IntroB	The following questions deal with various types of discrimination you may have experienced in Denmark due to your ethnic background				
B1	Within the past year, have you experienced, because of your ethnic background, being denied access to places which other people were allowed to enter? (such as a bus, taxi, nightclub or swimming baths)	YNDR	1 Yes 2 No 3 Do not know 4 Prefer not to answer		

Provided the background information includes citizenship of the respondents we can structure a filter for the entire block as defined in the table above. This means that only the non-Danish respondents are asked the questions in BlockB.

2.3.6 The sixth column (comments)

The column is used for remarks to the field. The remarks are inserted as comments immediately after the fieldname in the RULES section of the block. E.g. remarks about signals and checks. Comments may be added in free format.

Also, it is possible to assign a field description in this column by using s special syntax:

```
DESCRIPTION my description... ENDDescription
```

3. Technical description of the generation process

In the following we will describe the single procedures in the process of generating a Blaise web questionnaire from a description in a Word document.

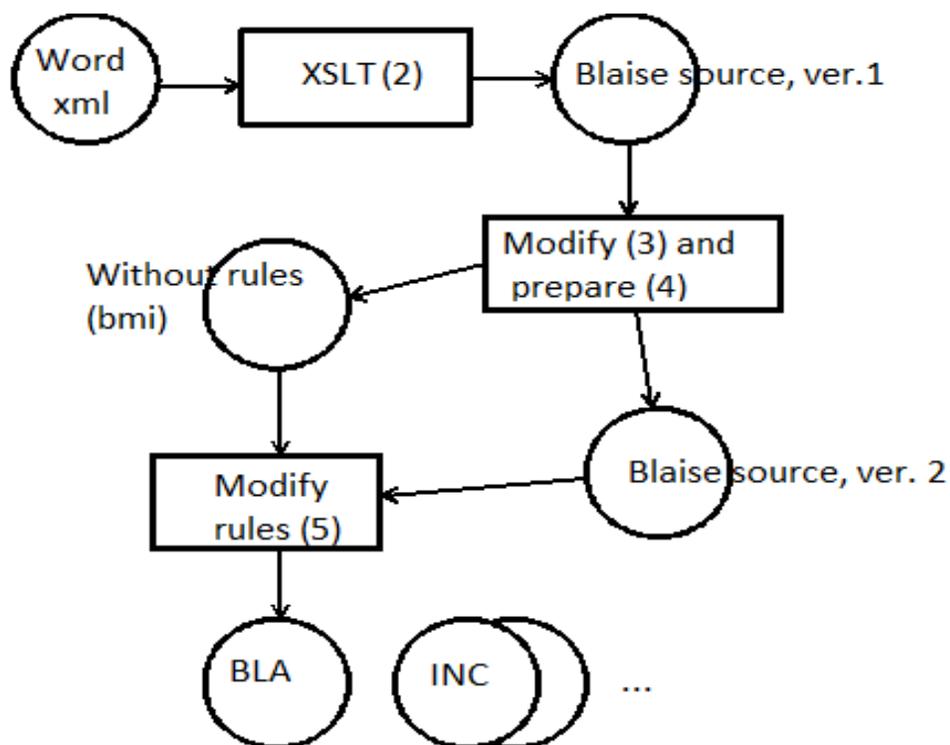
The main instrument for the generator is an XSL program transforming the Word document (in XML format) into a text file while producing Blaise syntax. This program is supplemented by a number of Manipula programs and VB scripts.

Steps of generating Blaise source code from a word document comprise:

- 1) Conversion of word document into an xml representation
- 2) Transformation of xml document into Blaise source code
- 3) Modification of Blaise source code
- 4) Preparation of a Blaise data model without rules instructions
- 5) Modification of rules instructions
- 6) Inclusion of generated source code into basic template
- 7) Preparation of final Blaise data model
- 8) Generation of layout groupings
- 9) Generation of a Blaise database (test data)
- 10) Validation of Blaise Internet Specification (.bis)
- 11) Generation of a Blaise Internet Package (.bip)

Generation is an iterative process: At any step the generation may be halted. For example, if the Word document deviates from the requirements, the generation may be halted and the Word document has to be corrected before regeneration.

Fig.: Steps 2-5 of the transformation proces



3.1 Conversion of word document into an xml representation

Initially, the word document is converted into xml representation (Word 2010 format) in order to facilitate the use of XSL transformations.

Xml format is a complete representation of the word document – actually, the Word 2010 file format (.docx) is merely a compressed file containing xml. Therefore, conversions from docx to xml or vice versa can be done without loss of data. The task is automated using a VB script referencing the Word API.

3.2 Transformation of xml document into Blaise source code

XSL transformations comprise identification of the constructs of the tables, rows and columns in the Word document. Each table is converted into a block definition consisting of:

- 1) The block definition itself
- 2) A field definition defining an instance of the block (unless it is marked TYPE)
- 3) An ASK instruction for the rules part of the main questionnaire block (unless it is marked TYPE)

From each of the rows in the table, proper FIELDS, RULES and LAYOUT instructions are generated with respect to the specific properties of the different columns. For example, the second column is used for definition of question text and requires that formatting codes – e.g. italics and bold – are preserved and converted into Blaise syntax. The fourth column may contain answer texts which should be handled the

same way, while all other columns contain only syntactical definitions where any formatting codes must be ignored.

3.3 Modification of Blaise source code

The source code generated by XSL transformation needs some modifications. For example, a Word document may usually contain characters that are not representable in the Windows character set used for Blaise source code.³ Therefore, the XSLT process produces output in utf-8 which will need to be converted.

Also, the XSLT process stores the generated source code in one single text file. We will prefer to split this file into several files – one for each defined block.

These conversions are handled by a Manipula program.

3.4 Preparation of a Blaise data model without rules instructions

The most difficult part of the source code generation is the proper generation of Blaise rules instructions. Therefore, the previous process also generates a Blaise data model without rules instructions. The first reason is that at this stage we can try to prepare a Blaise data model without rules and layout instructions, but comprising the overall structure including blocks, fields and types.

If the preparation of this data model fails then there is something wrong with the specification in the Word document and there is no reason to proceed any further. However, if the preparation is successful we may use this data model for looking up fields and types and retrieving complete path names for the fields and checking or converting between code names and codes of enumerated fields.

3.5 Modification of rules instructions

Conditions and expressions need to be properly checked and this is done by a Manipula program using the metadata functions (GETFIELDINFO, mainly) to look up the different elements of an expression in the rules-free data model prepared in the previous step.

The rules parts of the source code are searched for structures like “IF condition THEN” and for lines containing “name := something”. By looking up the fields and their types in the data model names of fields and categories may be checked, constants may be checked for validity and – possibly – field paths may be added to fields belonging to another block than the block currently analyzed.

3.6 Inclusion of generated source code into basic template

The basic Blaise-based data collection depends on usage of common templates for different kinds of data models.

Business and household surveys, for example, differ in the kind of background information needed and specific CATI handling. However, most of the differences may be defined in a few blocks holding the

³ The generator was initially made to produce code for Blaise 4.8.2. From Blaise 4.8.3 it is possible to represent question texts etc. as utf-8, but we haven't experimented with this representation.

background data and non-response codes and treatment. Also, there are differences in the way web and telephone surveys are carried out. Therefore, we use separate data models for the two modes, but these data models only differ in the general handling of the form and still share the same questionnaire.⁴

The general template contains hooks that can be used for automatic inclusion of the generated code.

For example, in the main questionnaire block the template contains a hook:

```
{### INCLUDES BEGIN ###}
```

At which place the following code could be inserted:

```
INCLUDE "BBlockA.inc  
INCLUDE "BBlockB.inc"  
INCLUDE "BBlockC.inc"
```

Likewise, there are defined hooks for placement of `FIELDS` and `RULES` instructions for the main questionnaire block.

This task is also carried out by a Manipula program.

3.7 Preparation of final Blaise data model

If the previous steps were successful, the Blaise parser will prepare the data model!

3.8 Generation of layout groupings

If the preparation was successful, it is possible to inspect the data model in order to generate layout groupings for the web questionnaire. The layout generator is a C# program using the Blaise APIs in order to retrieve information about field pane settings from the generated data model and to generate and store layout groupings in the Blaise Internet Specification file.

There are a number of conventions applied in order to figure out which groupings should be generated. For example, a series of fields on the same page all using the same GroupTable field pane will result in the generation of a GroupTable layout group comprising all these fields.

Also, the mere existence of two or more consecutive fields with the first field of type enumeration or set and the second and possibly following fields of type string or numeric and using the field panes named OtherSpecify will result in the generation of an Other-Specify group.

⁴ The generator is included in the Cati Survey Management System, maintained since 2000 and described in a paper to the IBUC/2003. Templates and generators have been an integral part of this system for many years.

3.9 Generation of a Blaise database (test data)

For the purpose of testing the questionnaire a test database should now be initialized. Depending on the needed background information the data set can be a standard test data set or a data set created for the specific survey modified for the purpose of testing.⁵

3.10 Validation of Blaise Internet Specification (.bis)

Through a call to the IsValid method of the Blaise Internet API the modified specification file may now be checked for validity. If it is valid, the generation process has been completed and the questionnaire is ready for testing. This task is carried out by a VB script.

3.11 Generation of a Blaise Internet Package (.bip)

Last step of the automatic generation is construction of a package file (the same VB script as above), possibly followed by installation on a web server (by another VB script referencing the Blaise Internet Server API).

4. Conclusions

The automatic generator has been in use since the beginning of 2012 and has proved useful in order to:

- Save time in the production of questionnaires
- Produce a source code more safely and with fewer errors
- Improve the cooperation with the customer in the process

The time saved is mainly in the more “boring” parts of questionnaire generation which allows the questionnaire developers to focus their efforts on the more difficult parts of questionnaire development, i.e. the rules and conditions, and also to improve the layout of web questionnaires.

Therefore, automatic generation has led to an improvement of the quality of our questionnaires within the given and unchangeable time frames the development process is subject to.

Development of the generator has been focused on current needs, i.e. the requirements have been formulated stepwise as the needs have turned up. Thus, the features developed have been the most important and work-saving.

A number of enhancements are still on the list and may be implemented as they reach sufficient importance. For example, we can mention support for randomization procedures and there is still discussion going on how we should supply more support for the data delivery, e.g. by incorporating descriptions in the document.

⁵ Test data generators and standard input programs has been part of the SMS for a number of years. They have been implemented using Manipula and C#.

5. Acknowledgements

We wish to thank the Blaise community for long lasting and varied inspiration in how to do as much as possible with as little effort as possible. Special thanks to Gerrit de Bolster, Statistics Netherlands, who kindly submitted manuals and descriptions of the Blaise IS generator. All these different ways to generate and get the job done has been a great inspiration for the development of our own generator.

6. References

Gerrit de Bolster: BlaiseIS at Statistics Netherlands, in: Essays on Blaise 2009. Proceedings of the 12th International Blaise Users' Conference, Riga 2009.

Appendix: Questionnaire “2012 Integration Barometer”

BlokA	Citizenship				
IntroA	First, a few questions about your participation in Danish society				
A1_2a	Denmark has many associations and clubs, such as trade unions, sports clubs, tenants' associations, cultural and religious associations, consumer societies like Brugsen and FDM, as well as fundraising organizations like the Red Cross and the Danish Cancer Society. Are you a member of a club or an association?	YNDR	5 Yes 6 No 7 Do not know 8 Prefer not to answer		
A1_2b	Within the past year, have you taken part in a meeting or other activities held by an association or club of which you are a member?	YNDR		A1_2a = 1	
A1_2c	Within the past year, have you carried out unpaid, voluntary work for any association or club of which you are a member?	YNDR		A1_2a = 1	
IntroA1_3	The following questions deal with your political participation in society	QuestionTextOnlyGrouptable NEWPAGE			
A1_3a	Within the past year, have you been a member of a political party or taken part in a political meeting aimed at social change?	YNDR GroupTable			
A1_3b	Within the past year, have you written a letter to the editor of a newspaper or taken part in a debate on the internet aimed at social change?	YNDR GroupTable			
A1_3c	Within the past year, have you taken part in a petition drive, a demonstration or a strike aimed at social change?	YNDR GroupTable			
A1_3d	Within the past year, have you contacted a politician, a public official, the media, an association or an organization for the purpose of social change?	YNDR GroupTable			
A1_3e	Within the past year, have you boycotted or deliberately chosen to buy specific products, such as organic products, for the purpose of social change?	YNDR GroupTable			

A1_3f	Within the past year, have you been involved in fundraising for or contributed to an organization like the Danish Cancer Society or a political party for the purpose of social change?	YNDR GroupTable			
-------	---	--------------------	--	--	--

BlkB3MatrixRow TYPE		NONEWPAGE			
B3_YN	Yes/No	YNDR_DD MatrixFieldpaneDropdown	Yes No Do not know Prefer not to answer		DESCRIPTION Yes / No: ENDDescription
B3_HowM	Within the past year, how many times?	TManyTimes_DD MatrixFieldpaneDropdown	1 Once 2 2 – 5 times 3 6 – 10 times 4 More than 10 times 5 Never 6 Do not know 7 Prefer not to answer	B3_YN = 1	DESCRIPTION Within the past year, how many times? ENDDescription
B3_Where	Where (most often)?	TPlace_DD MatrixFieldpaneDropdown		B3_HowM = 1-4	DESCRIPTION Where (most often)? ENDDescription

BlokB	Discrimination			UndersType 1 = 1,2	The questions in Block B are not asked of Danish citizens
IntroB	The following questions deal with various types of discrimination you may have experienced in Denmark due to your ethnic background				
B1	Within the past year, have you experienced, because of your ethnic background, being denied access to places which other people were allowed to enter? (such as a bus, taxi, nightclub or swimming baths)	YNDR			
B1_1	<i>Within the past year</i> , how many times have you experienced,	TManyTimes	8 Once	B1 = 1	

	because of your ethnic background, being denied access to places which other people were allowed to enter? (such as a bus, taxi, nightclub or swimming baths)		9 2 – 5 times 10 6 – 10 times 11 More than 10 times 12 Never 13 Do not know 14 Prefer not to answer		
B1_2	Where did you experience this most often?	TPlace_DD Dropdown	1 In the street 2 On a bus, train, taxi, airplane or the like 3 In a shop 4 In a bank 5 At a café, bar, nightclub or the like 6 At swimming baths, a water park, an amusement park or the like 7 In a sports club or a cultural or religious association or the like 8 At or near your home 9 At work 10 While attending school, training courses or the like 11 In contact with a doctor, nurse, hospital or the like 12 In contact with the municipality 13 In contact with the police 14 At a court of law 15 Other 16 Do not know 17 Prefer not to answer	B2 =1-4	Dropdown
B2	Within the <i>past</i> year, have you experienced, because of your ethnic background, being rejected after having applied for a job, bank loan, housing, mobile phone subscription or similar?	YNDR NEWPAGE			
B2_1	<i>Within the past year</i> , how many times have you had an application rejected because of your ethnic background?	TManyTimes		B2 = 1	
B2_2	Where did you experience this? (more than one answer is possible)	TPlaceVar SET OF	1 In the street 2 On a bus, train, taxi, airplane or the like 3 In a shop 4 In a bank	B2_1 = 1-4	Validering: Ved ikke kan ikke kombineres med andre svar.

			5 At a café, bar, nightclub or the like 6 At swimming baths, a waterpark, an amusement park or the like 7 In a sports club or a cultural or religious association or the like 8 At or near your home 9 At work 10 While attending school, training courses or the like 11 In contact with a doctor, nurse, hospital or the like 12 In contact with the municipality 13 In contact with the police 14 At a court of law 15 Other 16 Do not know 17 Prefer not to answer		SET OF
B2_2Other	Where?	STRING[100] InputAndErrorText		B2_2 = 15	Other specify

BlokB3Matrix				UndersType 1 = 1,2	
B3Intro	<i>Within the past year, have you experienced the following situations because of your ethnic background?</i> (such as in a shop, at a café or hospital, at the doctor's, from the municipality or from the police)				Matrix
B3_1	Being given poor service?	BBikB3MatrixRow			DESCRIPTION Being given poor service? ENDDescription
B4	Being the object of offensive words or degrading jokes?	BBikB3MatrixRow			DESCRIPTION Being the object of offensive words or degrading jokes? ENDDescription
B5	Being spat on, shoved, jostled or hit?	BBikB3MatrixRow			DESCRIPTION Being spat on, shoved, jostled or hit? ENDDescription

BlokC	Social supervision				
IntroC3	The following questions deal with your family's role in your choice of boyfriend/girlfriend and spouse			Alder = 18-29	
C3_1	Are you married?	YNDR		Alder = 18-29	
C3_1a	Did your family permit you, or do you think they permitted you, to have a boyfriend or girlfriend before you were married?	YNDR		C3_1 = 1 AND Alder = 18-29	
C3_1b	To what extent do you feel that your family has let you freely choose your present spouse?	TExtent	To a great extent To some extent To a lesser extent Not at all Do not know Prefer not to answer	C3_1 = 1 AND Alder = 18-29	
C3_1c	Did your family choose a spouse for you against your will?	YNDR		C3_1 = 1 AND C3_1b= 3, 4, 5 AND Alder = 18-29	

Blaise Code Generator

Jason Gray, Statistics Canada, Collection Systems Division

Éric Joyal, Statistics Canada, Collection Systems Division

Sam Threinen, Statistics Canada, Collection Systems Division

Background:

Since Statistics Canada began using Blaise in the late 1990's, the block development process has been a slow and error prone task. It consisted of copying and pasting from the specifications and adding the proper Blaise syntax for formatting. A code generator was developed fairly early on but it still relied on copying and pasting from the specifications.

In early 2011, Statistics Canada released the Questionnaire Design Tool (QDT). This tool is used to specify and manage the development of a questionnaire. All specification elements for new social survey development including question text, interviewer notes, answer types, conditions, and dynamic text are now stored in a central SQL database.

With the QDT in place and being used by our clients to specify their collection instruments, our Blaise team wanted a tool to take advantage of this new specification format and improve our block development process. The Blaise Code Generator (BCG) was created with this in mind. The BCG was created to partially automate block level coding. The goal of the BCG is to write 70-80% of the block level code. This removes all the slow, error prone work and allows the developer to focus on the more difficult block level programming such as logic, flows and edits. It uses business objects and stored procedures from the QDT database to retrieve the data directly using an SQL adaptor.

The BCG can build a block of code (the largest so far has been 3000 lines) in 1-2 seconds. This has drastically reduced the time needed for block development as well as increasing the quality of the code.

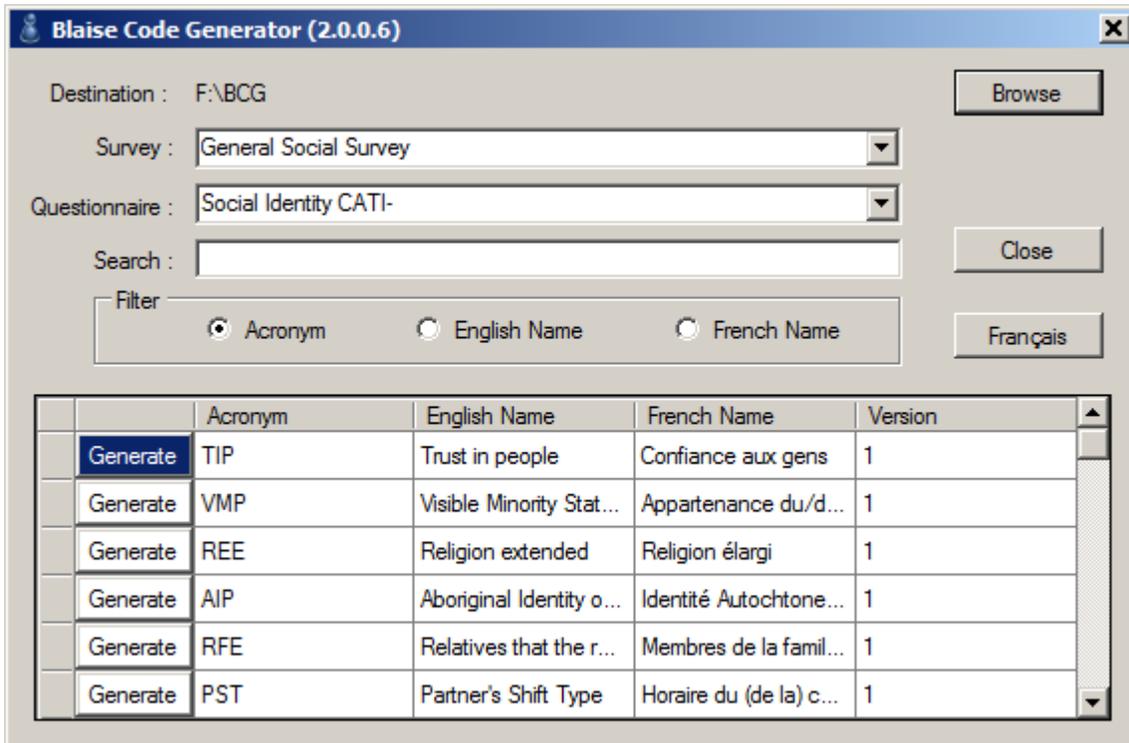
Inner workings of the Blaise Code Generator

QDT data is stored in a MS SQL database. Each element of a Blaise specification document is stored in different columns in the database. These elements include, English question text, French question text, Interview Note English, Interviewer Note French, Dynamic text English/French, Answer types, Conditions, plus more.

Using N-Tier architecture, the business layer is built using CSLA business objects. Some of the business objects include DynamicText, DynamicTextCollection, UserDefinedTypeItem, UserDefinedTypeItemCollection, Element, ElementCollection. The SQL Data adapter connects to the datastore and runs a predefined stored procedure. The data from the SQL data reader is loaded into the business objects.

Each element in the ElementCollection has an element type. Each element type has a corresponding string template. The template is a string constant that defines how that element type is to be written to a source file and how it should be displayed to the interviewer. Each element is then added to a stringbuilder object where it is finally written to a file.

INSERT diagram of QDT - BCG



Screenshot of the Blaise Code Generator interface.

Future extensions of the Blaise Code Generator:

The question level help text is also being stored in QDT database, attached to the individual question element. A component is currently being created to assemble all the question level help topics into a properly formatted rich text format file. This file can then be run through the help file creation software to generate the windows help file for the application. This new help file creation component will allow us to change a tedious manual (and error prone) process into a quick automated job.

Web-based CAI System for Blaise Instruments Development

Lilia Filippenko and Sridevi Sattaluri, RTI International, USA

1. Introduction

Over the course of conducting data collection for many studies, RTI recognized the need to unify the instrument design and documentation process for all CAI studies regardless of the mode of data collection. To achieve this goal, RTI developed a full featured web-based CAI system, Hatteras™, which allows instrument designers and programmers to work concurrently and supports multi-mode data collection efforts for CAWI, CAPI, and CATI instruments.

Hatteras™ consists of three major components: SurveyEditor, SurveyEngine, and Commander. All specifications are stored in database tables through the user interface provided by SurveyEditor. For web-based instruments SurveyEngine reads the tables to produce the web pages. For Blaise instruments a client utility Hatteras™ Commander is used to generate Blaise source code and scripts for Audio Computer Assisted Self Interview (ACASI). Hatteras™ provides multiple language support. Translation specialists can edit and comment the instrument via the SurveyEditor. Through version control, Hatteras™ permits multiple tasks to be performed on the same instrument, allowing programmers, translators, reviewers, and the design team to work simultaneously. It keeps an audit trail or history of any change made to the instrument.

This paper describes in some details how Hatteras™ SurveyEditor and Hatteras™ Commander were adapted to speed-up development of multilingual Blaise instruments.

2. Background

Since 2004 the Questionnaire Specifications Database (QSD) was used at RTI to develop Blaise instruments. QSD consists of tiers for the user interface and business rules, and the back end relational database. Microsoft Access had been chosen at that time based on its simplicity, its familiarity among the programmers, and to ease the transition from working with specifications exclusively in word processing documents. In using QSD on a number of studies, we at RTI International have found that it reduces Blaise questionnaire development time especially those with a second language. But problems with using Access databases remotely and limitation creating only bilingual instruments pose a need for using a web interface and SQL Server as the back end.

We decided to use existing web-based CAI system Hatteras™ that is used for all web-based studies at RTI and was developed by RTI programmers. Hatteras™ is in production since 2006 and has since been utilized on several large scale survey projects. We looked at different features of Hatteras™ and found that many of them are very close to what we used to have:

- The Hatteras™ Commander has an option to upload a text file that conforms to the Hatteras™ spec format – it is the first step in getting specs in QSD.
- The Hatteras™ Survey editor website is where all instrument specifications are saved – it is sufficient to use for Blaise instruments as well.
- Comments are a way to communicate changes needed – they are compatible with feature in QSD to monitor problems.
- Specs can be outputted to a readable format

There are many additional features that are used for web instruments, but are not needed or cannot be used with Blaise instruments. The most important one of them is the Hatteras™ engine that creates pages on-the-fly to conduct web interview. For us, it meant that Hatteras™ needed to have an option to generate Blaise code and have a few additional settings to recognize what kind of instrument is loaded. Although Hatteras™ is a great system for work with the web instruments, we will describe in this paper only the features used to develop Blaise instruments.

3. Hatteras™ SurveyEditor

3.1 Overview

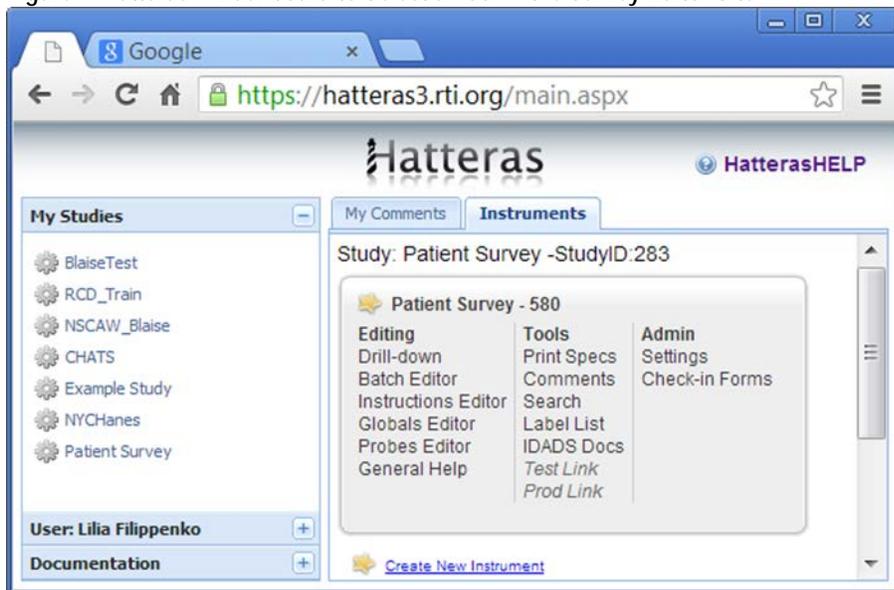
SurveyEditor is the interactive instrument development portion of Hatteras™. SurveyEditor provides the user interface for editing all aspects of a survey instrument including forms, items, comments, program logic, and other applicable information. Hatteras™ stores the meta-data for all instruments created using Hatteras™ in the SQL Server database.

A system admin is the only person that can setup a new user or change instrument assignments for a user. When new user is added to Hatteras™, admin will assign a user name for the user, enter their first and last names, enter their RTI email address, and assign a password for the user. These credentials will be used for all studies available for the user in Hatteras™. There are 5 Roles to use in the Hatteras™ Editor:

- Client - Read only. Client may only add comments to forms.
- Language Specialist / Spec Writer - Can change specs in the instrument, except for code blocks.
- Programmer - Can change all specs in the instrument.
- Study Admin - Can access the Admin tools available for the instrument.

These roles can be assigned for each instrument. Once the user has logged in, a list with available studies is shown on the left. When the user clicks on a study, the instruments for that study will show up on the right, under the Instruments tab. To start editing an instrument, the user clicks on the Drill-down editor for that instrument.

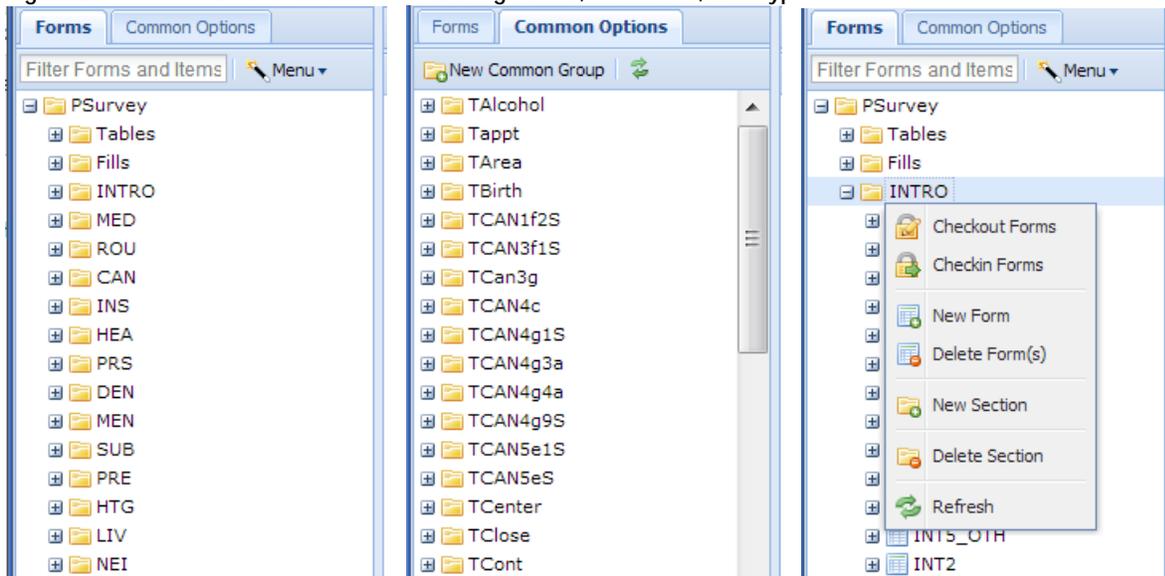
Figure 1. Hatteras™ Dashboard to Select a Tool in the Survey Editor Site



3.2 Drill-down Editor

To start editing an instrument the user clicks on the Drill-down editor for that instrument under “Editing” on the dashboard. The Drill-down editor is the primary instrument editing interface. The left menu displays all sections in the instrument under “Forms” tab and all enumerated types used in the instrument under “Common Options” tab.

Figure 2. Hatteras™ Drill-down Editor for Editing Forms, Questions, and Types

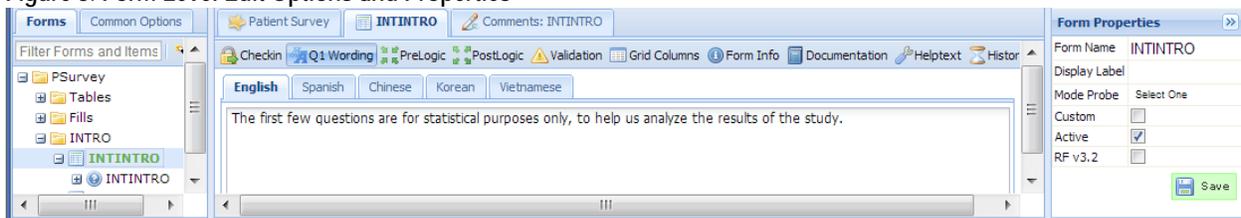


There are five levels of detail in an instrument:

- Section – a grouping of forms
- Form – the page which will be shown in a web instrument
- Question – the main question on the form
- Item – the data element(s) collected on the form
- Item option – properties of the item

To view a specific form, the user clicks on that form in the left menu. This will expand to display all questions and items contained in that form. In Blaise instruments each form can have only one question and usually one question has only one item. The item name is served as a field name in the Blaise instruments. When a question has a few items, the question wording is added as a stem to all items for this question.

Figure 3. Form Level Edit Options and Properties



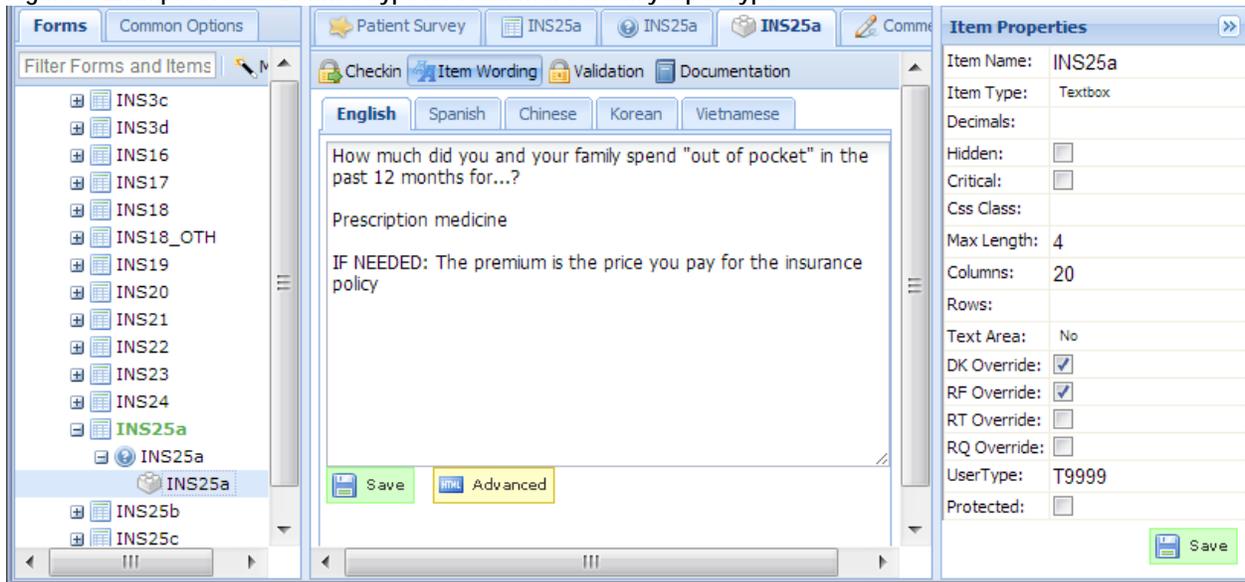
Different options are available at form, question, and item levels. Some of them are described in this paper.

3.3 Blaise Specific Options

Hatteras™ section, form, and question options are used for Blaise instruments development exactly in the same way as for web instruments. But at item level new properties were added and special rules for using existing properties were designed to accommodate specific Blaise instrument needs.

To use Blaise user-defined types in Hatteras™, property “UserType” was added. It is also used when a question or item type unique to Blaise is needed. For example, an array of textboxes can be specified as a simple looping section with one textbox type question but a simpler way is to just specify the “ARRAY [1..n] OF string[20]” in the “UserType” property of the item.

Figure 4. Example of User-Defined Type for Field with Currency Input Type



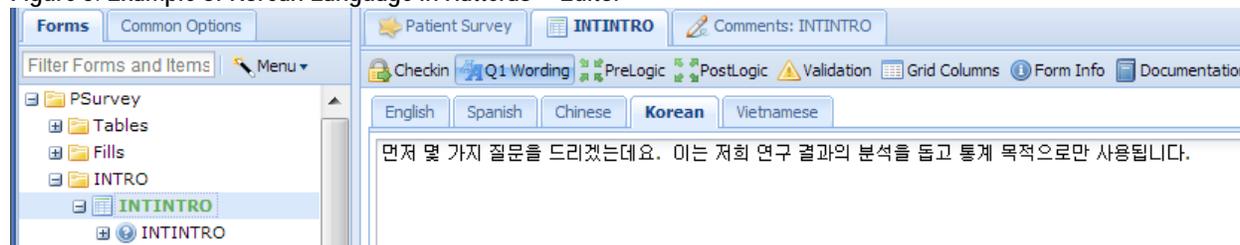
In addition, following rules are used for the Blaise instruments:

- Special Blaise field attributes “description” and “tag” can be specified on the “Documentation” tab for the item. The “Label” serves as the description, and the “Analytic” property is served as the tag for the Blaise field.
- To use a field with the “SET OF” type in Blaise, in Hatteras™ the question is defined as “checkbox” and its item is defined as “radio”.
- Within the drill down editor, the instrument level setting can be overridden by setting the “DK Override”, “RF Override” and “RQ override” properties of the item to not allow “Don’t Know”, “Refusal”, and/or “Empty” respectively for the item.

3.4 Managing Languages

Hatteras™ allows language specialists to translate and edit the instrument in multiple languages. The translated text can initially be mass imported into Hatteras™, and edits can be made via the Hatteras™ Editor. The language specialist can view the “master” language (usually English) at the same time they are viewing and editing the alternate language(s).

Figure 5. Example of Korean Language in Hatteras™ Editor



3.5 Comments and History

Hatteras™ allows structured, documented communication of specifications between programmers, specs writers, translators, and clients. It provides a robust comment feature, which allows all designers, testers, and programmers to log comments associated with a particular form. Comments provide a way to communicate changes needed and provide to-do lists based on comment statuses for instrument designers and programmers. Comments can be accessed two ways. The main way is through the comments link under Tools on the dashboard. Clicking on the “Comments” link from the dashboard takes the user to the comment report for that instrument. The other way is through the Drill-down editor.

Figure 6. Example of Comments Report

Select Comments Status

All
 New
 Under Discussion
 Action Determined
 Programmer Action Needed
 Non-programmer Action Needed
 Programmer In Progress
 Programmed Not Deployed
 No action needed
 Changes completed
 Testing failed
 Verified

[Show All Sections](#)
[New Comment](#)
[Print Comment Details](#)

Section	Form	Priority	Comment	Assigned To	By	Created on	Status	Last update
B1Loop	B1b	Normal	This screen will not let you enter 1 week or 1 month, having problems entering anything here for number and week/month. Had to enter DK to both to move forward	jperkins		7/7/2011 3:21:57 PM	Verified	11/14/2011 11:49:41 AM
B2Loop	B2_desc	Normal	The word "area" is missing e.g., "...living area vacuumed."	lflicker		8/1/2011 10:04:42 AM	Verified	11/14/2011 11:49:59 AM
CA13up	CA13upTr4	Normal	I corrected a typo	lflicker		8/1/2011 5:38:47 PM	Verified	11/17/2011 2:45:33 PM

Hatteras™ allows individuals to check-in/check-out forms which prevents duplication of effort. It keeps an audit trail of changes to each form in the instrument. History tab in the Drill-down editor provides a detailed record of all changes made to the form. Clicking the “Show History” button will display all changes.

Figure 7. Example of Form History Changes

Forms Common Options

All
 New
 Under Discussion
 Action Determined
 Programmer Action Needed
 Non-programmer Action Needed
 Programmer In Progress
 Programmed Not Deployed
 No action needed
 Changes completed
 Testing failed
 Verified

[Show All Sections](#)
[New Comment](#)
[Print Comment Details](#)

[Show History](#)

Source	Form	Question	Item	Details	Modified By	Modified On
Item Props UserType	INTDOB	INTDOB	INTDOB	tabDOB	liliaf	6/19/2013
Question Wording	INTDOB	INTDOB		What is ^Fill your name date of birth? ·	liliaf	6/19/2013
Item created	INTDOB	INTDOB	INTDOB	INTDOB	liliaf	6/19/2013
Question created	INTDOB	INTDOB		INTDOB	liliaf	6/19/2013
Form created	INTDOB			INTDOB	liliaf	6/19/2013

4. Hatteras™ Commander

4.1 Overview

The Hatteras™ Commander is the client utility used to assist with testing, debugging and other tasks needed for Hatteras™ Instruments. When the user starts it, the credentials as the one used in the

Hatteras™ Editor website are entered to login. After logging in, the user selects one of the available Study/Instrument to see more tabs. Only a few of these tabs are used by the Blaise developers.

Figure 8. Hatteras™ Commander Main Screen



The “Multi-language” tab is used to access the multi-language features of Hatteras™. Languages used for the study are checked in the list of supported by Hatteras™ languages and the “Master language” is selected to define the default. Once the user has setup supported languages, they are available in the Hatteras™ Editor website for translators. In addition, the “Multi-language” tab is used to do bulk exporting and importing of strings for translation.

The “Spec Loader” tab is used to upload a text file that conforms to the Hatteras™ spec format. Once uploaded, an instrument programmers, spec writers, and translators can modify the instrument from the Hatteras™ Editor website.

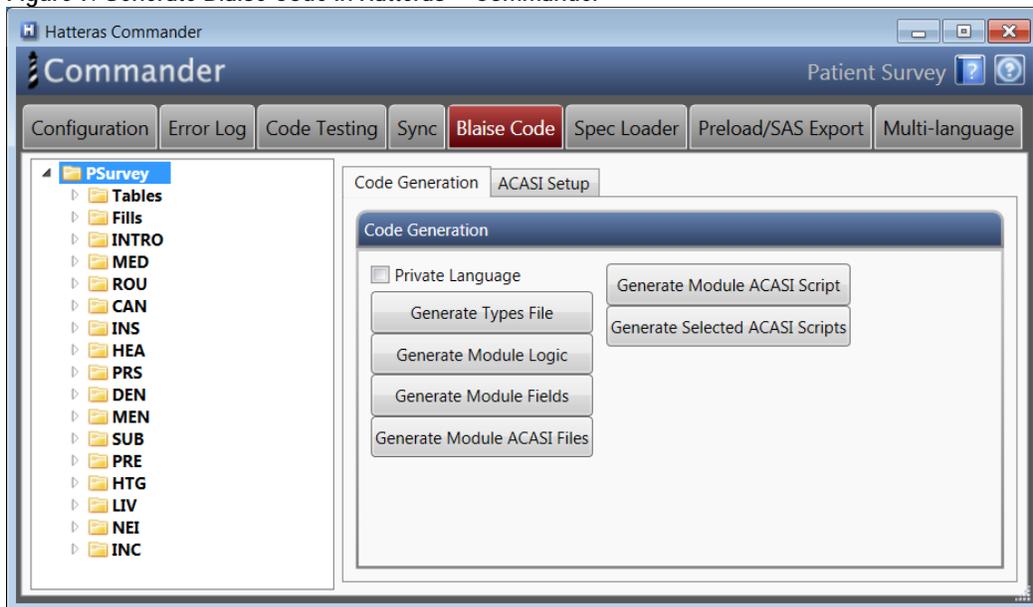
4.2 Generating of Blaise Code

The “Blaise Code” tab is used to generate Blaise files from the currently selected instrument. Each section in our Blaise instrument is programmed as a separate block with additional sub blocks as desired. All enumerated types are defined at the instrument level and are in one Types file. Since Blaise allows the separation of fields and rules at the block level as well, each block has an INCLUDE file which contains all fields associated with the specific block. This important feature of Blaise allows dividing responsibilities between programmers, specs writers, and translators.

To generate Types file for the instrument, the user clicks on the "Generate Types Files". A dialog appears asking the user to enter a location and file name. Once they are selected, a progress bar appears informing user of the file generation status.

To generate Blaise Logic or Fields file for a section, the user selects the section in the list and then clicks on the "Generate Module Logic" or the "Generate Module Field" button respectively. The file names will be pre-fixed with the name of the section. It is possible to generate files for all sections at one; to do so the user clicks on the section at the root level.

Figure 9. Generate Blaise Code in Hatteras™ Commander



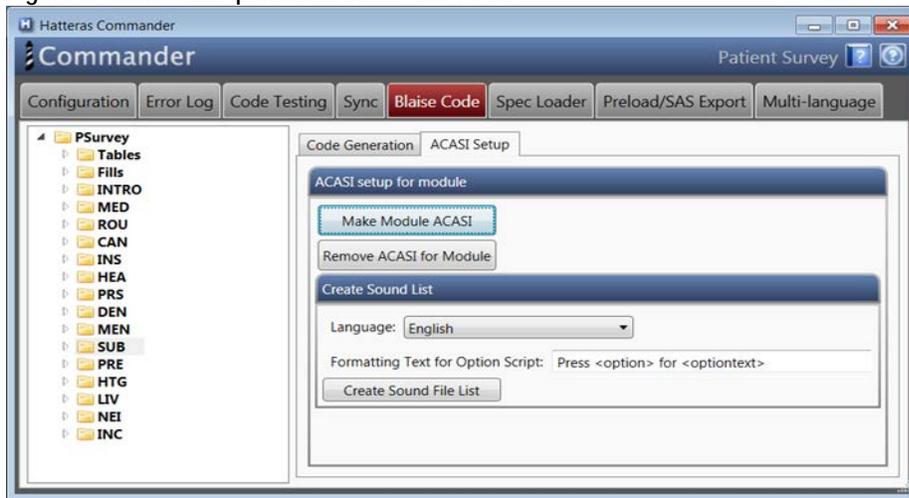
Programmers work in the Logic files only. Fields and Types files are created after changes are completed in the Hatteras™ Editor by specs writers and translators. The programmer’s goal is to use an approach where there is no hard coded text in the Blaise instrument. This is especially important for the Blaise instruments with Non-Western languages when Blaise Control Center cannot be used to edit text in such languages. All files are created as UTF-8 with a BOM (Byte Order Mark).

4.3 Setup ACASI with Blaise Instruments

For studies that require use of audio files, Hatteras™ helps to produce scripts for recording and generate Blaise code to play them during an interview. To set up a module as an ACASI module and manage the sound files associated with the questions in the module, user uses the “ACASI Setup” tab in Hatteras™ Commander.

After selecting the module, user clicks on the “Make Module ACASI” button to associate a tag that identifies the questions as an ACASI items.

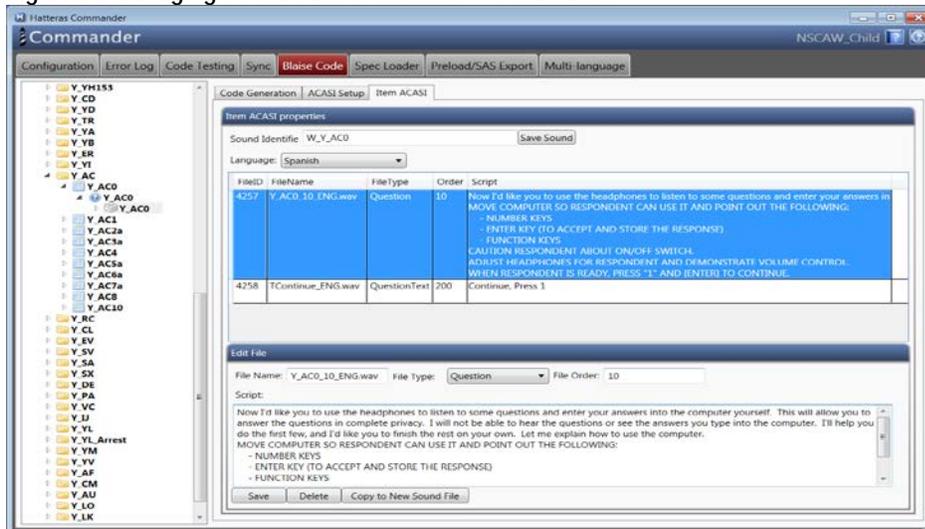
Figure 10. ACASI Setup in Hatteras™ Commander



In the “Create Sound List” panel, user selects the language and clicks the “Create Sound File List.” This will create the scripts for the questions and for the response options text for each question in selected language. The formatting text for the options script can be modified as required. With the default text, the option text script for as Yes/No question would be generated as “Press 1 for Yes; Press 2 for No.”

By selecting each question’s item, user can view the item level ACASI properties. A new tab “Item ACASI” appears. This tab provides the interface for editing an existing script, adding a new script file and deleting existing ones.

Figure 11. Managing Sound Files in Hatteras™ Commander



In the Blaise instrument “Sound Identifier” is used as an “AUXFIELD” to hold information about all sound files associated with the question. The list of sound files for a question can be modified and text for scripts can be edited in the “Edit File” panel.

The “Generate Module ACASI files” on “Code Generation” tab is used to generate relevant ACASI code files used in the Blaise instrument. The “Generate ACASI Scripts” and “Generate selected ACASI scripts” are used to generate word documents containing scripts for the recording artiste.

5. Summary

Hatteras™ is the software used at RTI International for developing the web-based instruments during multiple years. With addition of Blaise Code Generation feature, Hatteras™ became the standard system used by all questionnaire developers at RTI International regardless what programming language is required for the study.

Features of Hatteras™ used for the Blaise instruments development include:

- **Enhanced Workflow Process** – Hatteras™ allows structured, documented communication of specifications between programmers, designers, and clients. It keeps an audit trail and history of changes to each form in the instrument.
- **Multiple Language Support** – Hatteras™ allows translation specialists to edit, comment, and test the instrument in multiple languages.
- **Output of Programming Code** – Hatteras™ outputs the complete programming code for items of simple structure without logic branching. The output code serves as a starting point for more complex programming.
- **Import Specifications in Multiple Formats** – Hatteras™ can import specification files from word documents formatted according to a pre-defined template.
- **Output Specifications** – Hatteras™ displays and outputs interview sections in several formats, depending on the needs of the reviewer.
- **Output Codebooks** – Hatteras™ stores all metadata about the instruments. The metadata is used to generate the codebooks with the original questions, variable names, and response frequencies.

Using the unified web-based Hatteras™ system across RTI International allows programmers, specs writers, and language specialists simplify training process and speed up development of multi-mode and multilingual CAWI, CAPI, and CATI instruments.

Collecting Interviewer Observation Data via a Mobile Survey: Lessons Learned

Jennifer Kelley and Peter Sparks, University of Michigan, ISR-SRC

1. Introduction

On a major national, face-to-face, interviewer-administered survey, interviewers complete an interview observation form for each completed interview. The interview observation form consists of about 30 questions about the interviewing experience (e.g. Where was the interview conducted?). Presently (and in past cycles of data collection), interviewers complete the interview observations via paper-and-pencil while the respondent completes a self-administered section of the interview on the interviewer's laptop. The self-completed section, administered via ACASI, takes approximately 10 minutes to complete. The interviewer explains that they will be completing administrative tasks while the respondent completes this section of the interview. At a later time, usually at the end of the interviewer's field work for the day, the interviewer transfers the paper-and-pencil responses to the Blaise instrument on the laptop.

Transferring data from paper to computer not only introduces a risk for increased measurement and processing error, but also increases the interviewers' administrative time. To reduce error and increase interviewers' efficiency, the management team proposed to shift the paper-and-pencil interviewer observation survey to a mobile survey to be completed on the interviewers' smartphone.

2. Technical Systems

Three software applications were examined for creating the mobile interview observation survey: Open Data Kit, Illume, and Blaise IS. Each had strengths and weaknesses which are described below.

2.1 Open Data Kit (ODK)

ODK is an "app" on the mobile phone that uses instructions created by an Excel spreadsheet or other means. It is a specialized survey program for Android mobile phones that allows the survey to access features on the phone otherwise unavailable, such as the camera or GPS. ODK can store interviews on the phone, both partials and completes, until the data is ready to be sent to the data server. The interviewer can complete a number of tasks such as sending, receiving, deleting, and selecting surveys. ODK is an open source project and additional features may be available in future releases.

Development of ODK surveys are programmed using a web-based utility or by using an Excel spreadsheet that is then exported and converted into a survey. The surveys provide basic routing and calculations, but are lacking features when compared to Illume or Blaise IS.

2.2 Illume

Illume is a server-based interviewing system with web-based sample management features. Surveys are created via an editor program, and are stored as XML files. Formatting of the surveys is optimized for desktop systems. Specialized processing during the interview can be accomplished by writing DLLs that are called by Illume, such as prompting once if a field is left empty. While we began development in Illume for this project, we did not complete development using this system. Development was started on creating a mobile survey in Illume but was not completed for the mobile observations project. It became apparent that getting Illume to function the way we wanted would take some considerable effort, so the decision was made to focus our efforts on Blaise IS as Blaise IS would take considerably less effort.

2.3 Blaise IS

Blaise IS is also a server-based interviewing system that allows for control of many features of the survey, can perform complex calculations, and has a great deal of flexibility. However, like Illume, mobile phones require a constant connection to the Blaise IS server. Blaise IS does have a form-based (offline) survey with the following limitations:

- All questions are on one page in a large grid:
 - Question text is in the left column
 - Answer categories in the right column
- Look and feel is not optimized for a mobile survey
- The questionnaire, as an HTML page, would have to be sent to the interviewer
- The interviewer would have to open the HTML page to run the survey
- No rules logic
- Range checks only (such as numeric ranges)
- No consistency checks (i.e., can't check "none" and something else)
- A submit button at the end of the page and no previous/next buttons
- The default receipt page needs to be customized
- When the page is closed before submitting all data is lost
- Some fields don't appear (e.g., field for secondary key)
- Multiple sessions are not supported

Enabling Blaise IS for use on a mobile phone would require creating an “app” that would function similarly to ODK and implement the Blaise rules engine. Such an “app” would take considerable development time and resources and the decision was made not to pursue this avenue further.

Facing project deadlines, the decision was made to use Blaise IS for the mobile interview observation survey. ODK would have provided the functionality of the survey itself, but with the added complexity of integrating ODK with our systems (Blaise, SurveyTrak, and WebTrak), Blaise IS was determined to best fit the project's needs given the time constraints and available resources. Illume was not selected for similar reasons and that the usability of Illume was not as high as Blaise IS.

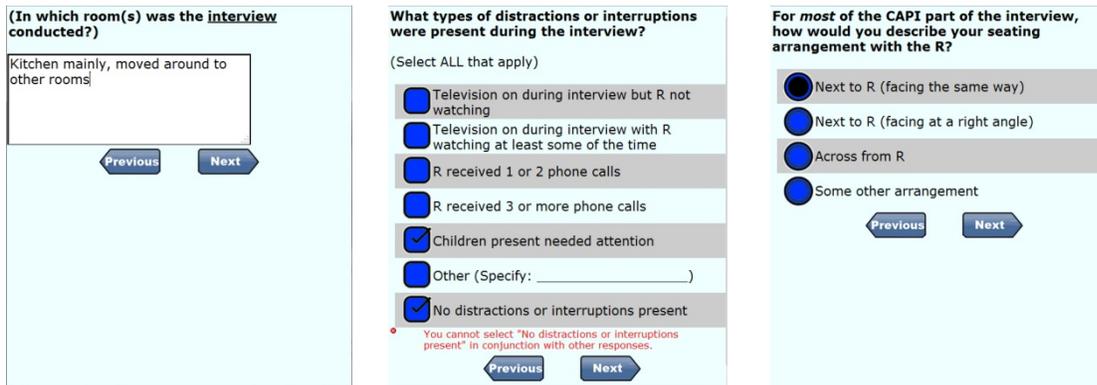
3. Implementation

The existing interview observation survey (desktop Blaise) was not changed or adapted for the mobile version. The question wording and the display was developed to mirror the desktop Blaise data entry, which displays one question per screen. However, three variables were added to the mobile version: Sample identification number, interviewer identification number and respondent concern (which is discussed later).

The default Blaise IS implementation of the survey did not function well on the phone. The text, radio buttons and check boxes were too small and did not resize. Question text, navigation buttons, and code frames were sometimes clipped if there was too much information on the screen. In such situations navigation was impossible because the buttons were unavailable.

Because the security of the data is of utmost importance, the data had to be handled carefully. It was decided to use a two-server setup: one web & rules server, and one data server. When the Blaise IS survey is deployed to the web/rules server Blaise automatically creates relative Blaise OLEDB Interface (BOI) files that point to the secure database on the data server. Initially, the data was going to be stored in SQL Server, but significant performance gains were observed when the native Blaise database was used.

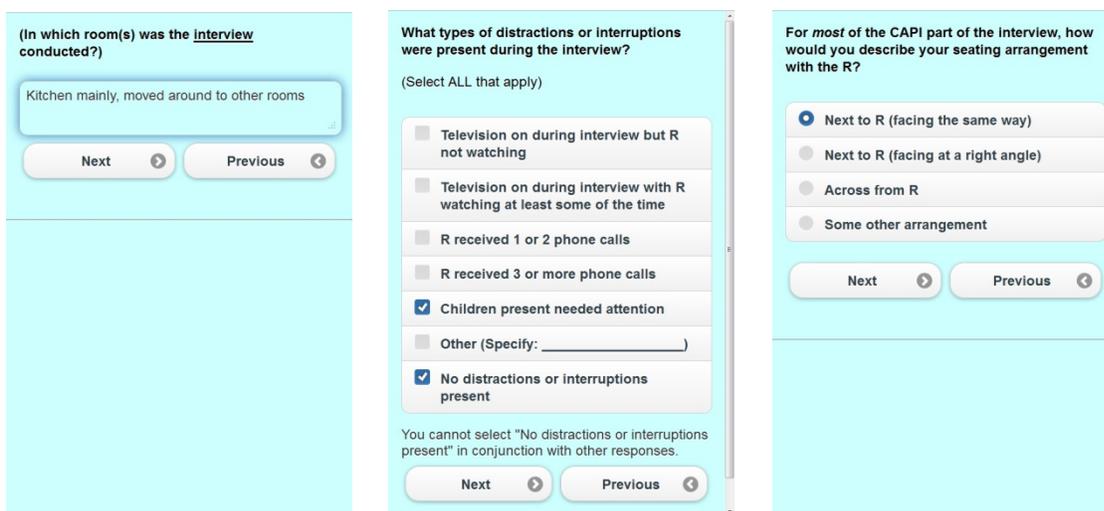
The initial programming modified the Blaise IS stylesheets by utilizing jQuery to customize the radio buttons and check boxes with clickable, resizable images, and add row highlighting on code frames. Clicking anywhere on the row would select the radio button or check box. The color and size of the radio buttons, checkboxes, next and previous buttons were entirely customized. The purpose was to minimally modify the stylesheets so that all the features of Blaise IS would remain. The images below are simulated screen shots from this implementation.



However, pages with too much information were still clipped, and the pages themselves still did not resize well on the phone. This behavior stemmed from the default stylesheets used in Blaise IS. There was also a significant lag when navigating because of the amount of information that had to be transferred between the mobile phone and the web/rules server.

The next approach was to try the C-Moto stylesheets from Tilburg University (Amin and Wijnant, Blaise On-the-Go: Using Blaise IS With Mobile Devices). The C-Moto stylesheets were based on the Blaise default stylesheets and were simplified by removing the extra features that were not used in the Tilburg survey. The C-Moto stylesheets also used jQuery mobile- a JavaScript library with features customized for the mobile phone.

Using the C-Moto stylesheets solved the problem of having the screens clipped and the navigation, but too many features had been removed to use with the survey. The stylesheets were then modified by adding certain features, such as text formatting (underline, italic). The images below are simulated screen shots from this implementation.



4. Field Test

A feasibility study was conducted in April 2013 for collecting interview observation data via a mobile phone using Blaise IS. Our main goals for the field test were to evaluate connectivity issues (i.e. network coverage), usability, interviewers concerns, respondent concerns and data transmission.

4.1 Design

Four field interviewers were selected to test over a three week period based on their smartphone skills. Since one of the goals was to assess connectivity or network coverage issues, interviewer's location (i.e. rural or urban) was also used as a selection criterion. A special training session was held with the selected interviewers to acclimate them to the mobile survey and the field test procedures. The interviewers were instructed to explain to the respondent that they would be completing administrative tasks on their mobile phone while the respondent completed the ACASI section. The interviewers were also instructed to have the paper-and-pencil version as a backup in case they had difficulties with the smartphone version and to notify their field manager if they could not complete the interview observations on their mobile phone.

As indicated earlier, a question was added to the mobile version of the survey to assess whether or not the respondent objected or voiced concerns about the interviewer using the smartphone during the interview. The ACASI section consists of sensitive questions and our concern was that respondents may perceive a decrease in privacy since the smartphone acts as a link to the outside world, which may in turn negatively impact data quality.

4.2 Monitoring

Daily checks were performed to assess connectivity issues, identify respondent concerns, verify the correct sample identification number was entered linking the mobile survey to the main survey data, and that the data was transmitted successfully.

4.3 Results

The four interviewers completed 51 interviews during the three week period. Of the 51 interviews completed, 43 of the interview observations were collected via the mobile survey, with the remaining eight collected via paper-and-pencil. Two of the interview observations collected via paper-and-pencil was due to problems with the link to the survey and six due to connectivity issues. For the 43 mobile surveys completed, all reported no respondent concerns.

4.4 Interviewer Debriefing

In addition to the data analysis, an interviewer debriefing was held shortly after the field test. In general, the interviewers gave positive feedback about the mobile survey process and the usability of the survey, with the all interviewers expressing that they especially liked saving data processing steps ("I don't have to do it when I get home").

Despite the interviewers reporting that the mobile survey was easy to use and that they liked the "look and feel" of it, they all reported connectivity issues. While the data showed 43 interviews completed via mobile, the interviewer debriefing revealed that of the 43 completed, the majority had considerable lag time. During in-house testing, the mobile survey took three minutes on average to complete. However, for the interviewer field test, the survey time was longer and in some cases, longer than the time it was taking the respondent to complete the ACASI section (i.e. more than 10 minutes). The interviewers reported that they had to reload pages or hit the answer multiple times before the survey would advance to the next question. When the interviewers were asked if they would like the mobile version implemented for production, they all said yes, if the lag time could be fixed.

5. Future Steps

While the interviewers reported the usability of the interviewer observations (programmed in Blaise IS) was high, the connectivity issues were significant enough that other approaches need to be explored, with further testing, before a mobile version of the interview observation survey can be deployed more broadly. Since we cannot control the level of network connection any given field interviewer will have at a respondent's home, we are currently exploring applications that have off-line capabilities, such as Open Data Kit and Form Hub.

References

Alerk Amin and Arnaud Wijnant, Blaise On-the-Go: Using Blaise IS With Mobile Devices, IBUC 2012 Proceedings

Blaise On A Windows 8 Tablet. The Caribbean Netherlands Implementation.

Lon Hofman, Ralph Dolmans & Gerrit de Bolster (Statistics Netherlands)

1. Introduction

In 2010 the relationship between the Netherlands and the six ‘Dutch’ islands in the Caribbean was changed. Three of the islands, Bonaire, Saint Eustatius and Saba are since then considered to be municipalities of the Kingdom of the Netherlands⁶.

The total number of inhabitants of the three islands is around 21,000. The official language is Dutch; many speak English, Spanish or Papiamentu. The US \$ is the official currency.

Statistics Netherlands (SN) received the task to produce statistics for those three Islands and opened a very small office on Bonaire, the biggest of the three islands. The distance between Bonaire and the other two islands is considerable. It is about 800 km (500 miles) by plane.

As part of the innovation program of SN, a program was started to improve and modernize the data collection. Someone came up with the idea to use tablets and soon it was decided to buy some 10+ DELL Latitude ST2 tablets with an Intel Atom processor, all running Windows 8 Pro 32bit. The tablets are used only on Bonaire. So each survey will be partially electronic and partially on paper.

Although for each tablet also a keyboard and a mouse were bought, the idea was to use the tablets during interviewing the way a tablet is intended to be used: without a keyboard and without a mouse. So the interviewing has to be done, if possible, by using the fingers.

There were two potential projects for using the tablets. One project was the ‘Household budget’ survey. It was decided to program the data collection application using Microsoft Access. It could have been done very well using Blaise / Maniplus but the department responsible did not have Blaise / Manipula knowledge, there was only a very limited time to get the project up and running plus the questionnaire was relatively simple because of limited routing.

The second project was in a sense more challenging because of the length and the complexity of the questionnaire. The survey is called the Omnibus survey. It is a survey about many different topics like work and education, the living situation, transport, internet usage, appliances, health and so on. According to lab tests it will take about half an hour to administer the survey. This paper is about the implementation of that survey.

2. The user interface on the tablet

The screen of the DELL Windows 8 tablet is a 16 by 9 screen. It is wide and not too high in landscape mode and it is high and wide enough in portrait mode. The on-screen keyboard that can be activated when trying to enter data in an input line is rather large: it covers almost half of the screen in landscape mode.

⁶If you want to find out more about the structure of the Kingdom of the Netherlands we suggest to Google the YouTube video ‘Holland versus the Netherlands’.

When the DEP is running in full screen mode, the on-screen keyboard can potentially cover questions on the formpane and this is not desirable. In portrait mode this problem does not exist. Because it was unclear what would be the preferred operating mode by the interviewer, two sets of screens were setup for the tablet: a set of screens for the portrait mode and a set of screens for the landscape mode. This was done by defining two layout sets in the mode library.

3. The Menu file

To minimize the use of the on-screen keyboard it was decided to implement a button panel in the menu file. The panel contains buttons for the most used options plus the numeric keys. It looks as follows:

Figure 1. The provided keyboard



The first row with buttons is used for navigation between questions. The arrows implement the 'next field' and the 'previous field' action.

The next four rows are used to implement keyboard actions. Each button represents one key stroke: the numeric keys zero through nine, the decimal key and the backspace key.

The buttons on the next row can be used to activate the remark dialog and to answer 'Don't Know' or 'Refusal'.

The four buttons with a flag can be used to switch between the four languages available in the questionnaire: Dutch, English, Spanish and Papiamentu.

The questionnaire has a parallel that is used for administration. By pressing the admin button the parallel is activated. When this parallel is active another button (not visible in the image) is available to switch back to the main parallel.

To allow for switching between landscape and portrait mode two buttons are available. The button to switch to portrait mode is visible in the image; the button to switch to landscape mode is not.

4. Implementing the questionnaire

The challenge was not in the programming of the questionnaire (just a lot of work because of the four languages which were not all delivered at the same time...). The challenge was in making it 'look nice' and making it easy to operate on the tablet. Given the length of the question text, the font size used (Tahoma 18) and the number of possible choices in the enumerations there was a fair chance of input lines not being visible because of the on-screen keyboard. It was therefore decided to put only one question on each page of the DEP. An example of a typical screen is displayed below, both in portrait and in landscape mode and both with and without the on-screen keyboard.

Figure 2. A DEP page in portrait mode

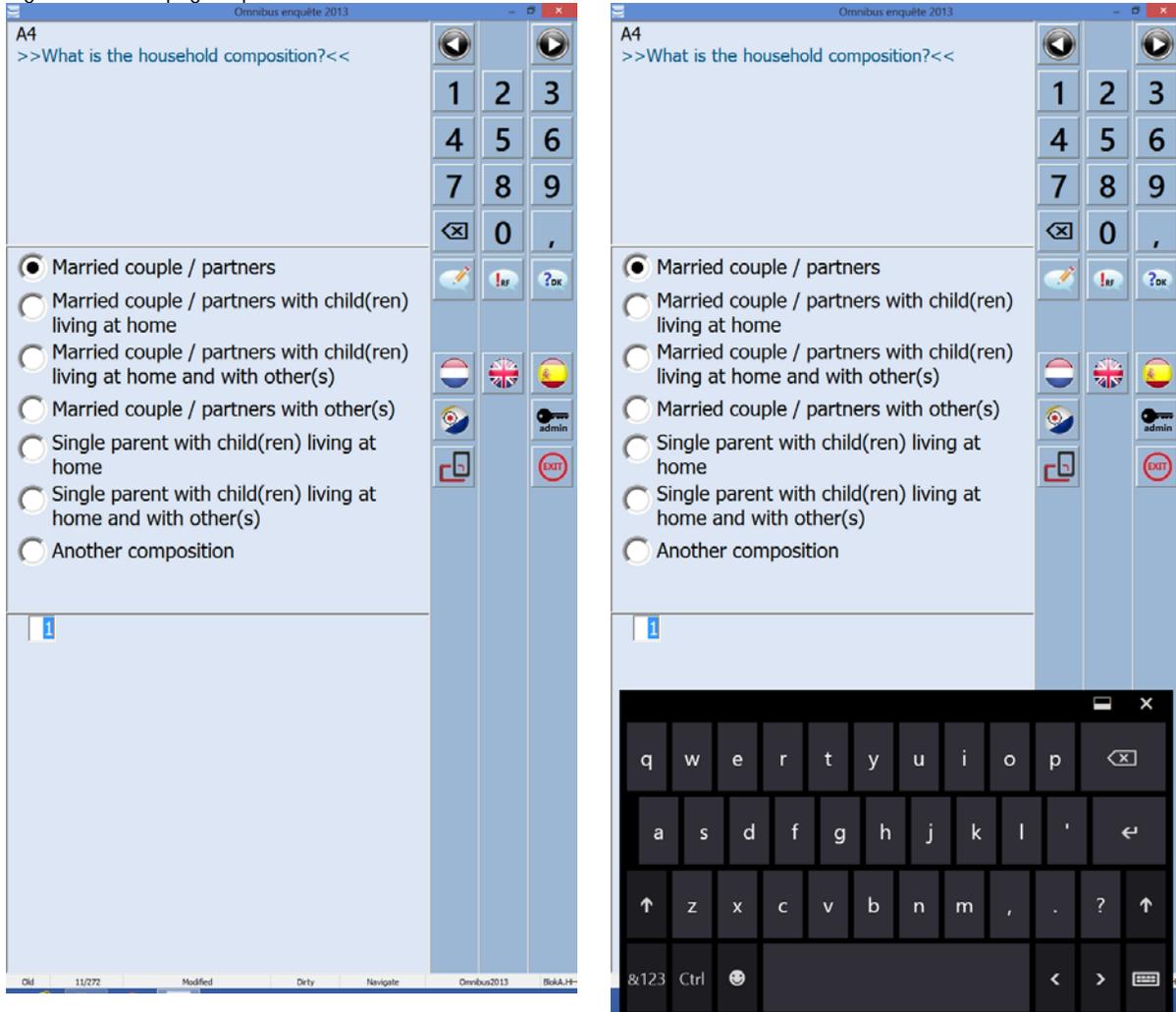
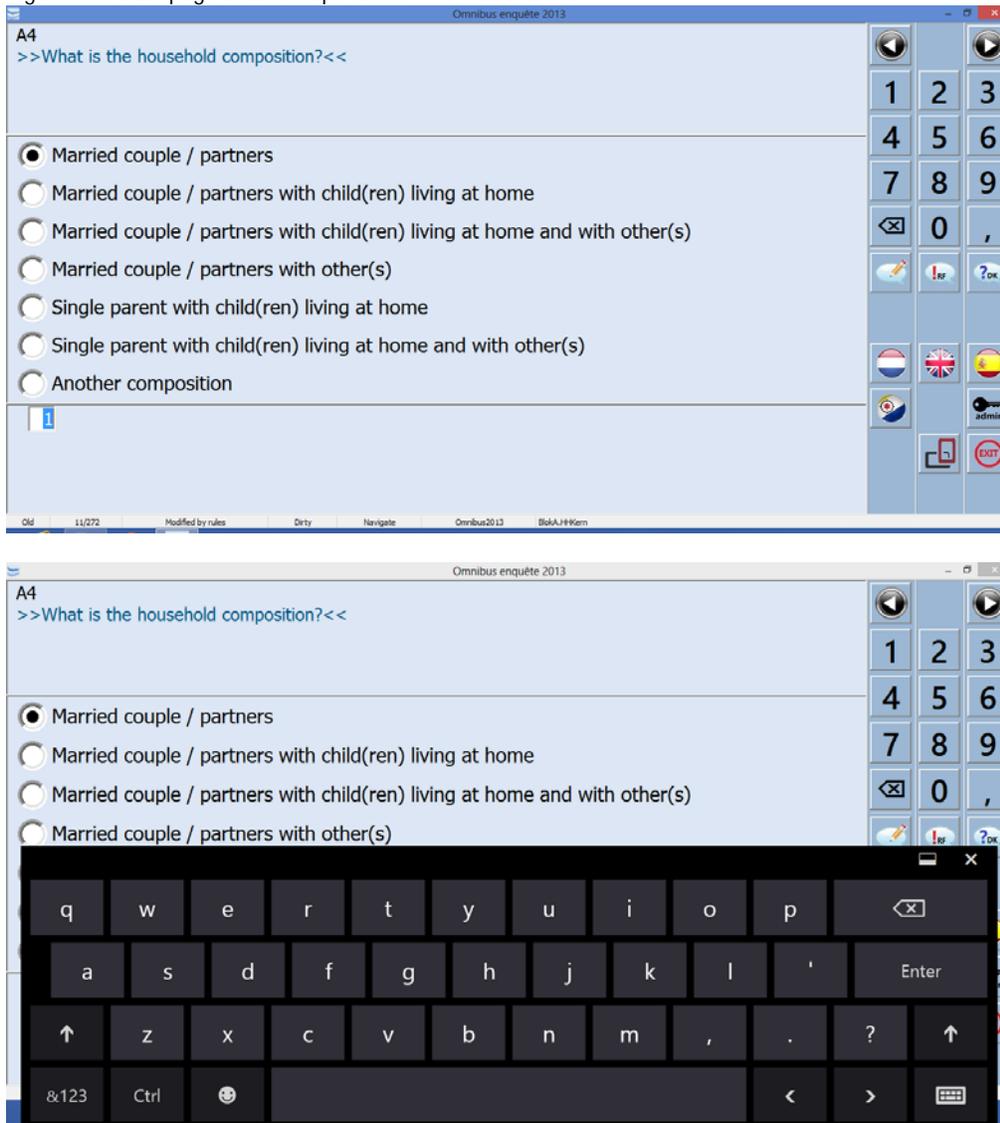


Figure 3. A DEP page in landscape mode



For each of the orientations a layout section has been defined in the questionnaire. The grid has been setup in such a way that only one fieldpane fits (a grid with a page width and a page height of 1). Most often an infopane instruction is used to get the correct page layout. Some 10+ infopane definitions have been defined. They differ for instance in height and in the number of columns in the answer list.

In the layout section a very handy but not very well know layout option is used: layout instructions based on user defined types. In the source code it looks as follows:

```
LAYOUT Interviewing_1 {portrait mode}
  AT TInteger2  INFOPANE InfoPaneWithInputLine
  AT TString140 INFOPANE InfoPaneWithInputLineOther
  AT TGebLand  INFOPANE InfoPaneWithAnswerList2Columns
```

By cleverly defining and using the types, this is an easy way to get to the right layout.

The questionnaire has a total of 272 pages in portrait mode and the same number in landscape mode. Plus there are four languages. To make sure that the right choices were made in the layout sections all pages needed to be inspected in both modes and in all four languages. That is a lot of work when this is done in dynamic routing mode. To guarantee that all pages have been inspected you have to enter data and you have to make sure that all routing paths are visited. Much easier is to switch to the editing toggle set. This toggle set allows you to navigate through the questionnaire without entering any data and when paging, all pages are visited, also the pages that contain off-route fields. It turned out that for most questions the Spanish texts were the longest. Making sure the screens were okay in that language was most often enough. Only a limited number of questions have text fills and the pages of those questions were inspected separately.

5. Security of the tablet

The following security measures were considered sufficient to run surveys:

1. Each interviewer was provided a username and password. They have limited user rights, just enough to run the surveys but for instance not enough to make changes to the Windows registry.
2. The USB-port and the SD-card reader are both disabled for mass-storage devices (so no USB-key or SD-card can be accessed) and the wireless internet is not available for the interviewer. So it is impossible for the interviewer to communicate with the outside world.
3. Access to the machine by USB/SD-card is only allowed for the administrator. He has to enable the reading of mass-storage devices by making a change in the Windows registry.
4. Once installed the access to the mass-storage devices is handled by the shell application.

6. The installation

A small installation was made using Microsoft Visual Studio 2005. The installation of the survey on the tablet was handled by the staff of the Statistical office on Bonaire. It was designed to be simple (which caused some challenges given the way security and shortcuts are handled by Window 8) and to prepare the tablet for the use of the Omnibus survey. The installation contains an encrypted sample file and it creates the shortcut on the desktop to start the survey. The installation package is provided to the staff by a secure download portal from Statistics Netherlands. The staff stored it on a USB-key for installation on the tablets.

To install the Omnibus survey on an interviewer tablet, the staff had to do the following:

- Enable the USB-port for mass-storage devices. This is done by changing a value in the registry.
- Plug in the USB stick with the installation package.
- Run the “Setup[Omnibus2013].msi”.
- Enter a password when prompted to decrypt the sample file.

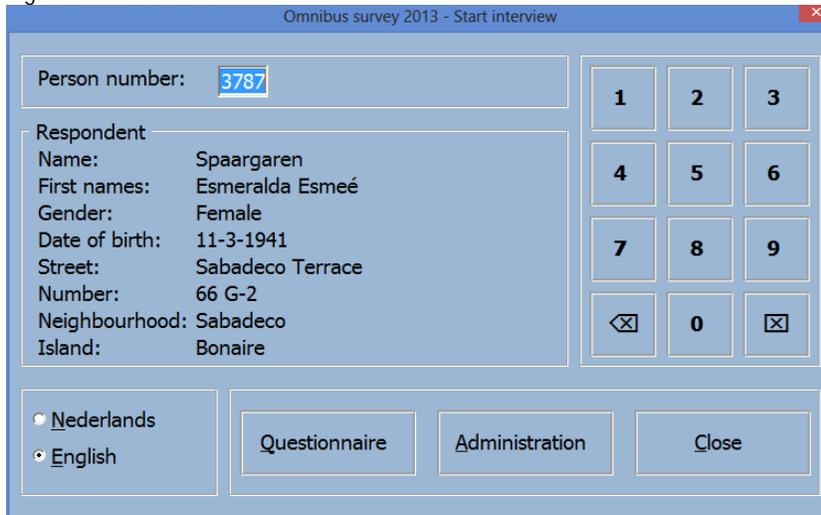
When ready, the installation automatically disables the mass-storage device and prepares the USB-key for safe removal.

The installation creates a folder in C:\ProgData with all the necessary files, including the sample file. By using the shortcut on the desktop the interviewer can then start the Omnibus shell.

7. The omnibus shell

The omnibus shell has been designed to be simple to use. Each interviewer has on paper a list of cases to interview. The interviewer enters a valid person number from the list in the shell. Once a correct number has been entered the credentials of the respondent are automatically displayed to enable the interviewer to verify that she/he is visiting the correct person. Two buttons are enabled when a valid person number has been entered: one button to start the main parallel of the questionnaire and one button to start the administration parallel of the questionnaire. The language of the shell can be switched between Dutch and English. That was considered to be sufficient.

Figure 4. The 'Start interview' shell



The screenshot shows a software window titled "Omnibus survey 2013 - Start interview". It features a text input field for "Person number:" containing the value "3787". To the right of this field is a numeric keypad with buttons for digits 1-9, 0, and backspace. Below the input field is a section titled "Respondent" with the following details: Name: Spaargaren, First names: Esmeralda Esmée, Gender: Female, Date of birth: 11-3-1941, Street: Sabadeco Terrace, Number: 66 G-2, Neighbourhood: Sabadeco, and Island: Bonaire. At the bottom left, there are radio buttons for "Nederlands" and "English", with "English" selected. At the bottom right, there are three buttons: "Questionnaire", "Administration", and "Close".

When the questionnaire is started, the orientation of the tablet (landscape or portrait) and the selected language are provided to the DEP. The landscape and the portrait mode both use a different modelib layout set.

Once the case has been completed (this is established based on a question in the questionnaire), the main parallel is closed for further editing. The administration parallel can still be edited.

Only the administrator is allowed to export collected data from the tablet. When he starts the shell the following dialog is displayed:

Figure 5. The 'Management' shell

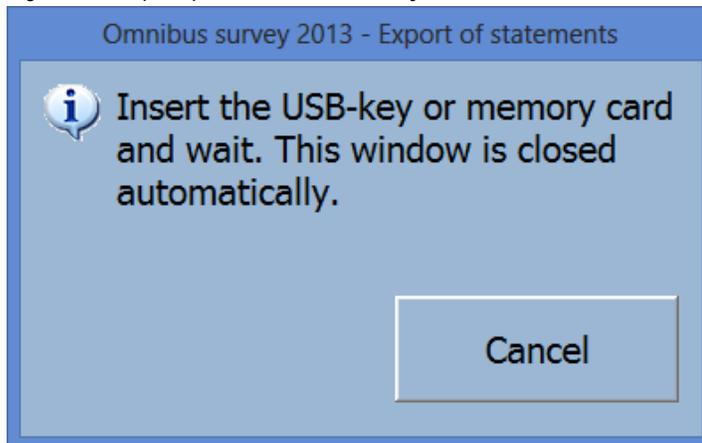


The screenshot shows a software window titled "Omnibus survey 2013 - Management". It features a section with radio buttons for "Nederlands" and "English", with "English" selected. To the right of this section are two buttons: "Export" and "Close".

The export button takes care of extracting the data and storing the data on the USB-key. It works as follows:

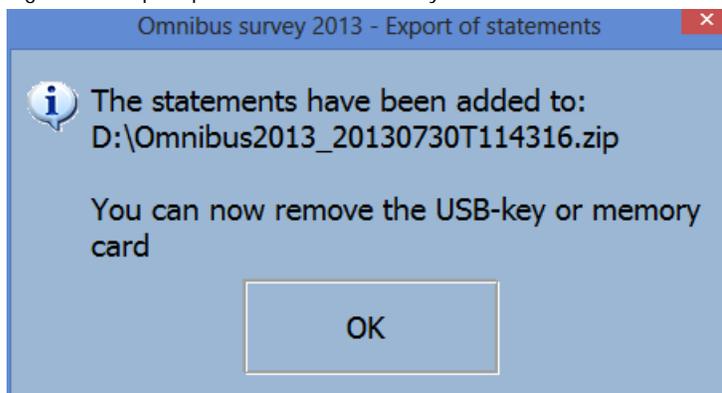
- The administrator presses the export button
- The application opens the USB-port for mass-storage devices and waits until an event occurs that a mass-storage device has been inserted. The administrator is prompted to insert his USB-key.

Figure 6. The prompt to insert the USB-key



- Once the USB-key has been detected the cases on the tablet are extracted and added to an encrypted zip-file on the USB-key.
- When successful the USB-port is closed again and the USB-key is prepared for safe removal. The administrator is then prompted to extract the USB-key.

Figure 7. The prompt to remove the USB-key



Once the data of all the tablet has been extracted the administrator can upload the encrypted zip file to Statistics Netherlands through an upload portal by using a PC with internet connection.

8. CADI

All the paper forms are entered by the staff of CN using the virtual environment of Statistics Netherlands. The same datamodel is used, but now in CADI mode and using a special layout set that mimics the pages of the paper questionnaire.

9. A few words on technologies used

Besides the Blaise tool Manipula.exe the following was used:

- VBScript. This is used for instance to detect events (the plug-in of the USB-key), create shortcuts during installation and detect the orientation of the tablet.
- HTA (HTML application). This is used for non-modal displays such as a splash screen and to prompt the administrator to insert the USB-key.
- Two freeware programs: RemoveDrive.exe and EjectMedia.exe to safely remove mass-storage devices.

10. Extending Blaise a little bit

Once you have to operate the DEP without a keyboard or a mouse, you immediately notice that there is something missing: an easy way to navigate between the pages. The operating system on a tablet is able to recognize many touch screen gesture. In case of Windows, a gesture that is recognized will be translated to something that could also have been done by a mouse. Because of this, many Windows controls like a menu and a radio button can be operated by tapping with the finger on the screen. So by default, when you run the DEP on a touch screen enabled device, it is possible to operate the DEP by tapping. No changes to the DEP were required for this. But not all gestures can be translated to a mouse action. Such gestures are recognized by Windows but they do not influence how the DEP behaves. Correct DEP behaviour can only be achieved by making a change to the DEP by the Blaise team. An example of such a gesture is the left/right swipe. Nothing happens because it is not detected by the DEP, but it would be nice if the DEP would do something that seems logical to the user.

Although not a requirement for the use of the tablets in CN, we decided to make a small change to the DEP. The 'left swipe' has been implemented as 'go to next page' and the 'right swipe' has been implemented as 'go to previous page'. In theory there is room for the support of more gestures.

One thing that is missing is an auto detect of the orientation of the tablet by the DEP and to automatically choose an appropriate layout set when the orientation changes. This will require some changes to the DEP. Currently there are no plans to add this to the system.

In the omnibus survey it was solved in two ways:

- The omnibus shell written in Manipulus uses VBScript to detect the orientation, and when an interview is started, the DEP is started with the correct layout set for that orientation.
- When the interviewer changes the orientation during interviewing, buttons are provided in the menu of the questionnaire to switch manually.

Note that in Blaise 5 this functionality is available, the user can specify resource sets specifically for certain resolutions or orientations, and the DEP will automatically choose the right one.

11. Some final remarks

The Omnibus project has not been fully evaluated yet but everything points in the direction of a success. The interviewers were able to handle the survey with limited instructions and the data quality (as expected with CAPI) is much better in the electronic part of the survey compared to the paper part of the survey.

Developing A Web-Smartphone-Telephone Questionnaire

Richard Boreham/Arnaud Wijnant, NatCen/CentERdata

This paper will explain how we implemented a web questionnaire that works on both PCs and tablets/Smartphones for the GUS project.

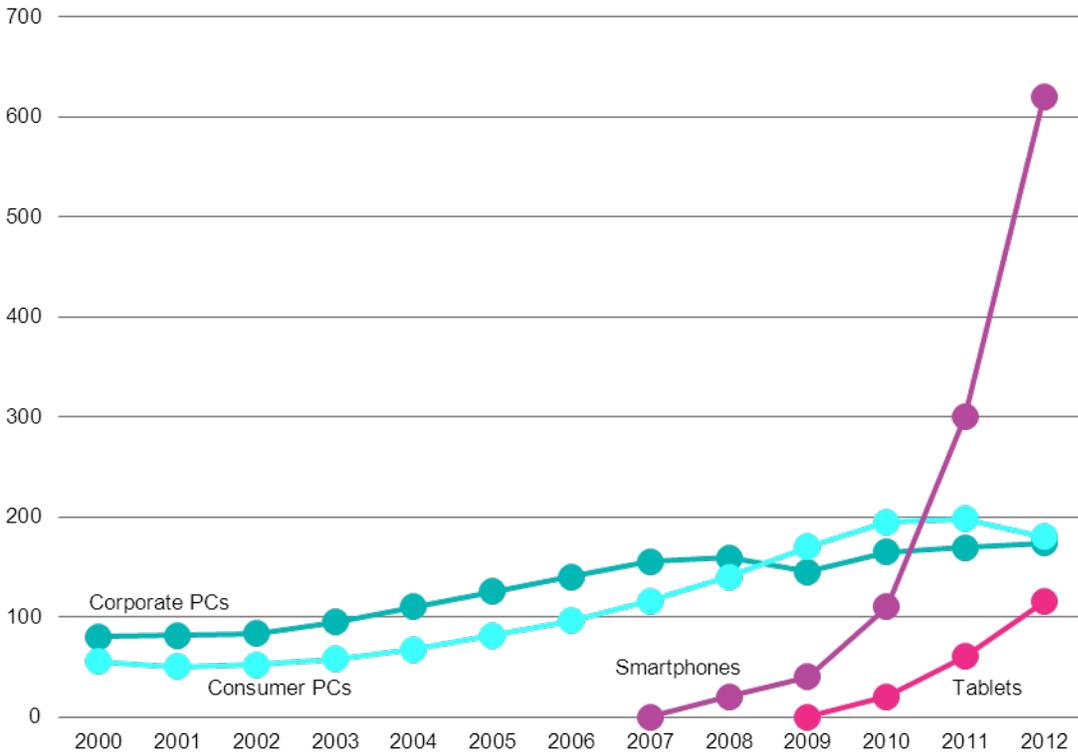
The Growing Up in Scotland (GUS) study is a major cohort study funded by the Scottish Government, following three groups of children through their early years, into childhood, adolescence and, possibly, beyond into adulthood. Seven waves with face to face interviewing have been completed so far, and the most recent wave was designed to be a short web questionnaire. We felt that this was an opportunity to develop a CAWI questionnaire using CentERdata's C-MoTo software to make it accessible from a smartphone or tablet as well as from a standard desktop or laptop computer.

C-MoTo makes it possible to create Blaise CAWI questionnaires that are suitable for mobile and touch devices. The C-MoTo questionnaire is user-friendly because the layout is automatically scaled to fit exactly on the device's screen. Also the user input is optimized by supporting automatic keyboard selection and gesture control. C-MoTo is available for free for non-commercial use.

1 Trends in annual sales of devices

Sales of Corporate PCs have risen from around 90,000 per year in 2000 to around 190,000 per year in 2012 and sales of Consumer PCs have followed a similar trajectory. Although Smartphones were only introduced in 2007, their growth has dwarfed that of PCs, and stood at over 600,000 units per year in 2012. Tablets were introduced in 2009, and the rate of growth in their annual sales is faster than PCs. Tablets look like they will overtake sales of both corporate and consumer PCs by 2014.

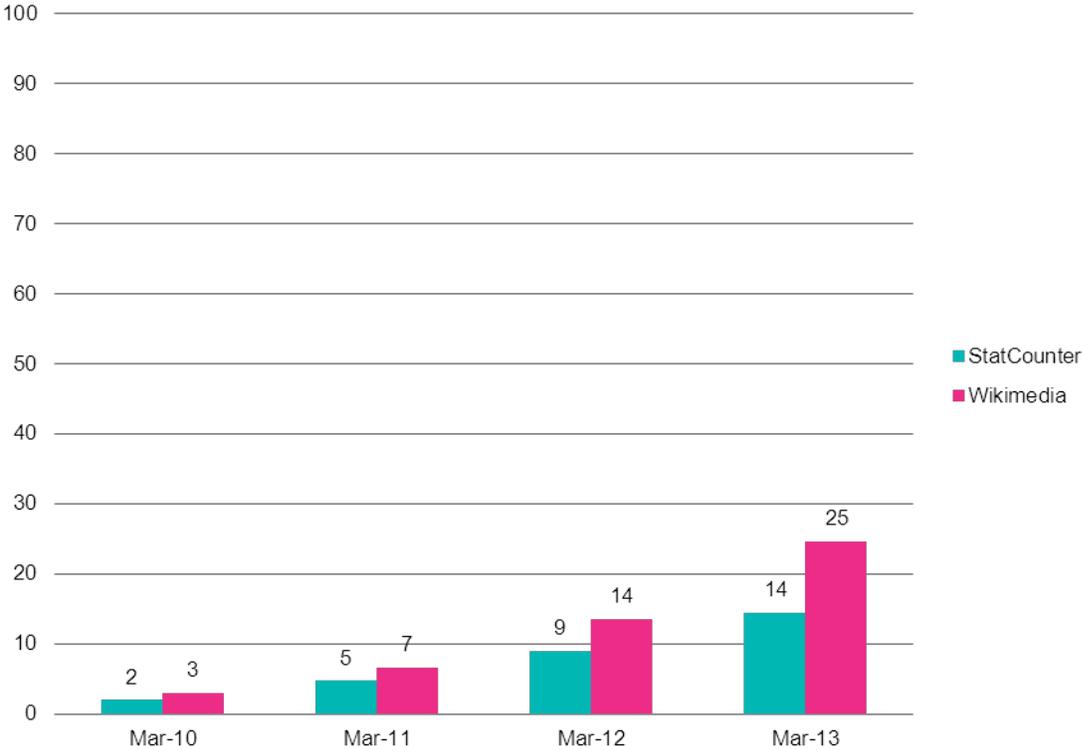
Figure 1. Global Annual Unit Sales



¹ Source: www.slideshare.net/bge20/2013-05-bea

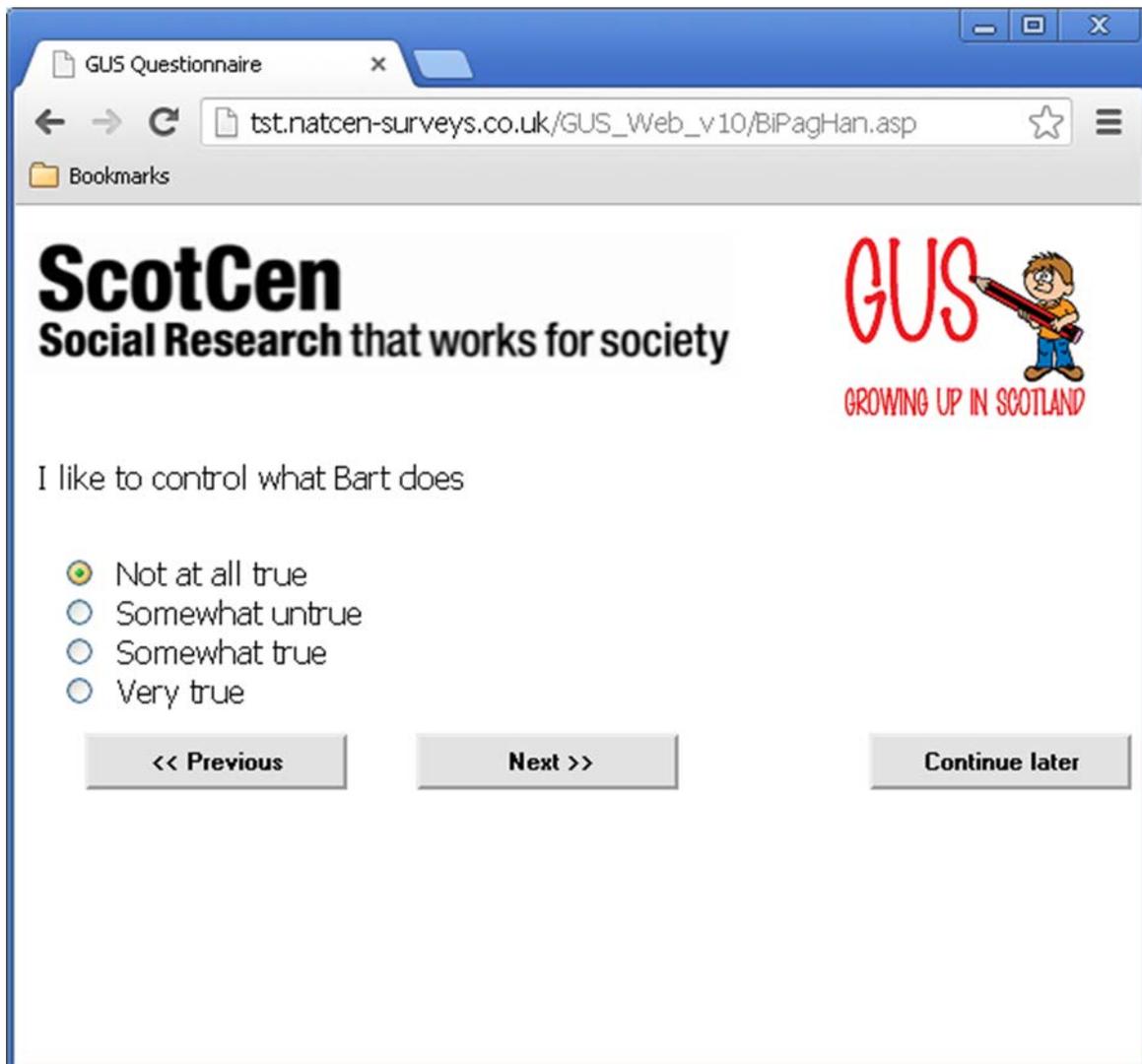
Also when you have a look at the percentage of web-traffic that originates from a mobile device, you see that since 2010 mobile devices are increasingly used to access the internet. Figure 2 shows the percentage of web traffic by mobile devices over time. One source is Wikimedia which measures the traffic of popular websites like Wikipedia and Wiktionary. The other source is StatCounter that tries to publish more global web usage statistics. Both statistics show a steady increase of mobile internet usage.

Figure 2. Percentage of mobile web use compared to all internet usage



2 Displaying a web questionnaire on different devices

The screenshot below shows a typical page from the standard GUS web survey when accessed from a PC. The type is large enough for people to read, and people can easily select the appropriate answer category and move to the next question.



GUS Questionnaire

tst.natcen-surveys.co.uk/GUS_Web_v10/BIPagHan.asp

Bookmarks

ScotCen
Social Research that works for society

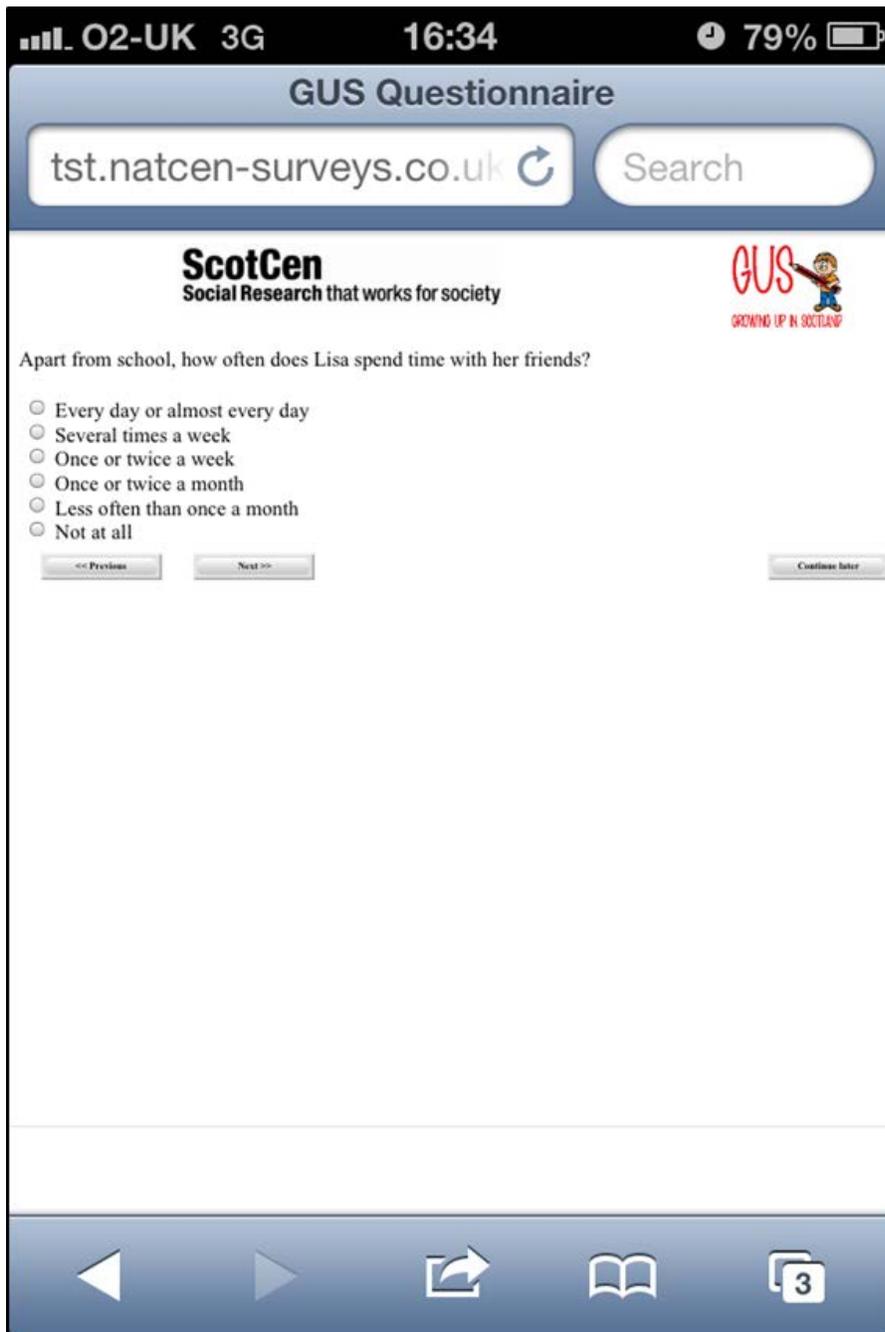
GUS
GROWING UP IN SCOTLAND

I like to control what Bart does

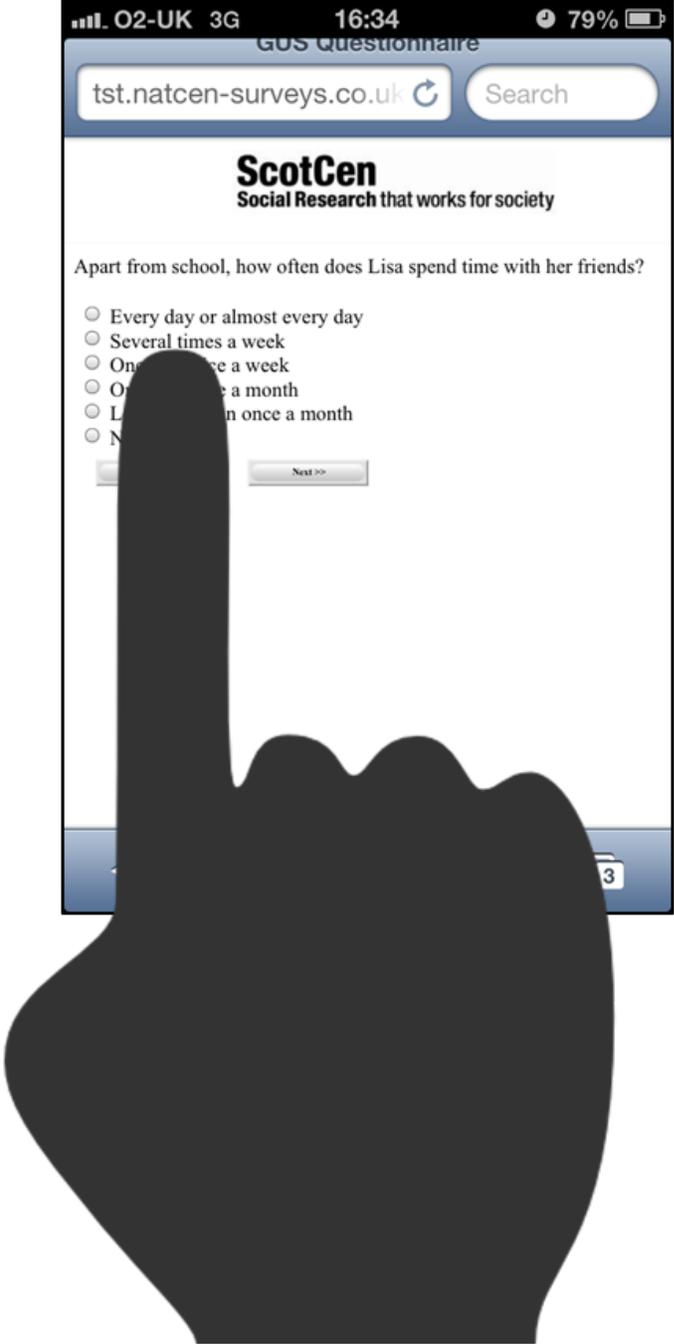
Not at all true
 Somewhat untrue
 Somewhat true
 Very true

<< Previous Next >> Continue later

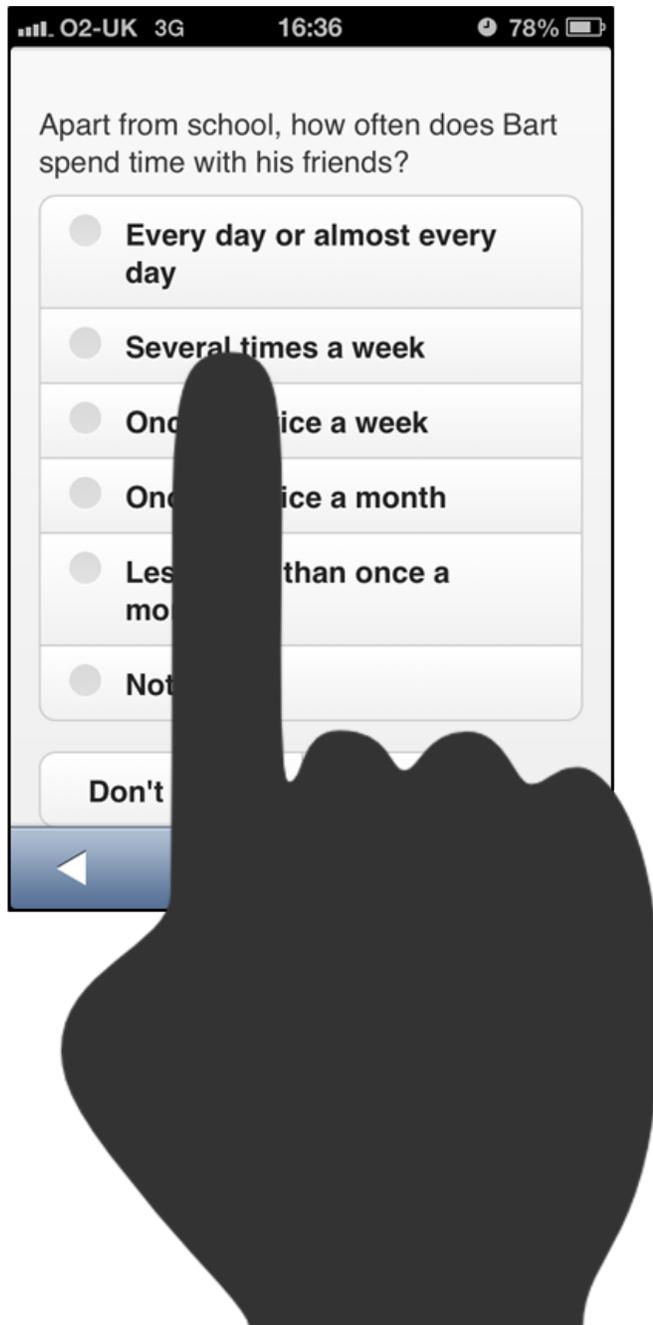
However, if the same questionnaire is accessed from a tablet or smartphone, then the text size is affected by the overall screen size of those devices. The C-MoTo software formats the questionnaire in a way that is much more suitable for a tablet or Smartphone user. The following screenshot on the left shows how the default Blaise web questionnaire displays on an iPhone, whereas the screenshot on the right shows the reformatted C-MoTo version.



We can see that the text for the default web format is much smaller and therefore harder to read. If a respondent pinches to zoom in, then although the text size increases, the question and answer texts do not wrap to the enlarged screen. In addition it is very difficult to select the appropriate answer because the radio buttons are small. Even when the user zooms in on the radio button in a traditional web questionnaire it is hard to select the correct one.



For this reason the complete area around the radio button in C-MoTo is clickable. This makes it much easier to select the correct answer.



The screenshot above shows how the questionnaire is formatted on a iPhone, but we also tested on Android phones and tablets and an iPad.

However, the C-Moto formatted questionnaire doesn't look good on a standard PC screen, so we had to design our questionnaire so that the format would vary according to the device used to connect to the

questionnaire, but that the data would be stored in one central database. The way that we did this is described in the following section.

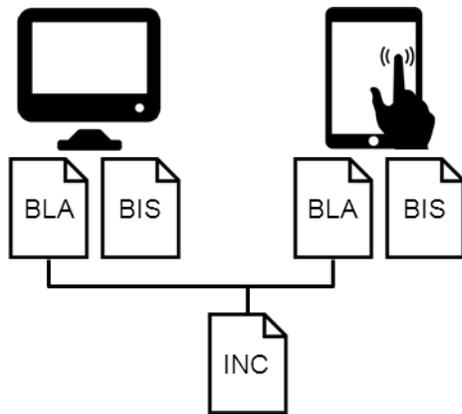
3 Designing a questionnaire to work in web and tablet modes

There were actually three modes that we need to take account of in the questionnaire design, as we needed a version of the questionnaire that could be accessed by our telephone interviewers for chasing incomplete questionnaires. From a design point of view, there is no difference between adding one additional mode and adding two additional modes, so for simplicity we focus purely on creating an instrument that would work in two modes: normal web and tablet/Smartphone.

3.1 Two questionnaires with common source code

We planned to have one questionnaire datamodel with two different BIS files to generate the alternatively formatted versions of the questionnaire. However, in practice we found that we needed respondents to enter their access code after we had rerouted them to the device-specific questionnaire, which meant that we could not feed a parameter about the device type into the questionnaire. It may be that this could have been solved with more time, but the short development timetable meant that we went with a pragmatic solution rather than necessarily an optimal one. Therefore we needed to have two different datamodels with corresponding BIS files.

We wanted to avoid programming two instruments, so we structured our code so that we could use as much common code as possible, and so that the only difference between the two datamodels would be code contained in the top level BLA file. So there was a pair of BLA and BIS files for web mode and a pair of BLA and BIS files for tablet mode, and then common source code for each datamodel contained in numerous INC files.



The differences between the BLA files for different modes were minimal. There were 3 languages defined in both instruments as follows:

```
LANGUAGES =  
  Web  "Web",  
  Tab  "Tablet",  
  HLP  "Help"
```

The reason for having multiple languages was that there might have been occasions where we amended the question or answer text for tablet mode if either of these texts was too long. For a more general setup we would also probably have two separate help languages, but this was not necessary for this particular project.

The only differences between the BLA files were the datamodel name and the definition of Prepare Directives.

Web .BLA	Tab .BLA
DATAMODEL Web "WebQure"	DATAMODEL Tab "TabQure"
{'x-' if not used}	{'x-' if not used}
{\$DEFINE Web}	{\$DEFINE x-Web}
{\$DEFINE x-Tab}	{\$DEFINE Tab}

Each BLA file had a different datamodel name (web and tab for web-mode and tablet-mode respectively). Prepare Directives were defined differently in each BLA file – we used a convention of putting “x-“ before a Prepare Directive to switch it off, as there was no code in the body of the questionnaire that was enclosed within an “x-something” directive.

The only place where we actually ended up using Prepare Directives was in the global rules section:

```
RULES
  {$IFDEF Web}
    Device:=Web
    SetLang:=Web
  {$ENDIF}

  {$IFDEF Tab}
    Device:=Tab
    SetLang:=Tab
  {$ENDIF}

  SETLANGUAGE(SetLang)
```

This hardcoded the device type and language into each instrument. Ideally we would have only had one datamodel and passed these in as parameters, but because of the need to enter the access code on the page following the re-direction to the appropriate datamodel, it was not possible to implement this.

We set up the fields and rules for the rest of the questionnaire using common source code. This meant that the web and tablet versions of the questionnaire were compatible and so we could easily add them together.

3.2 Questionnaire Scripting Issues

The C-MoTo software uses the tags on field definitions to define how to display certain types of fields. So for example open fields need a textarea tag.

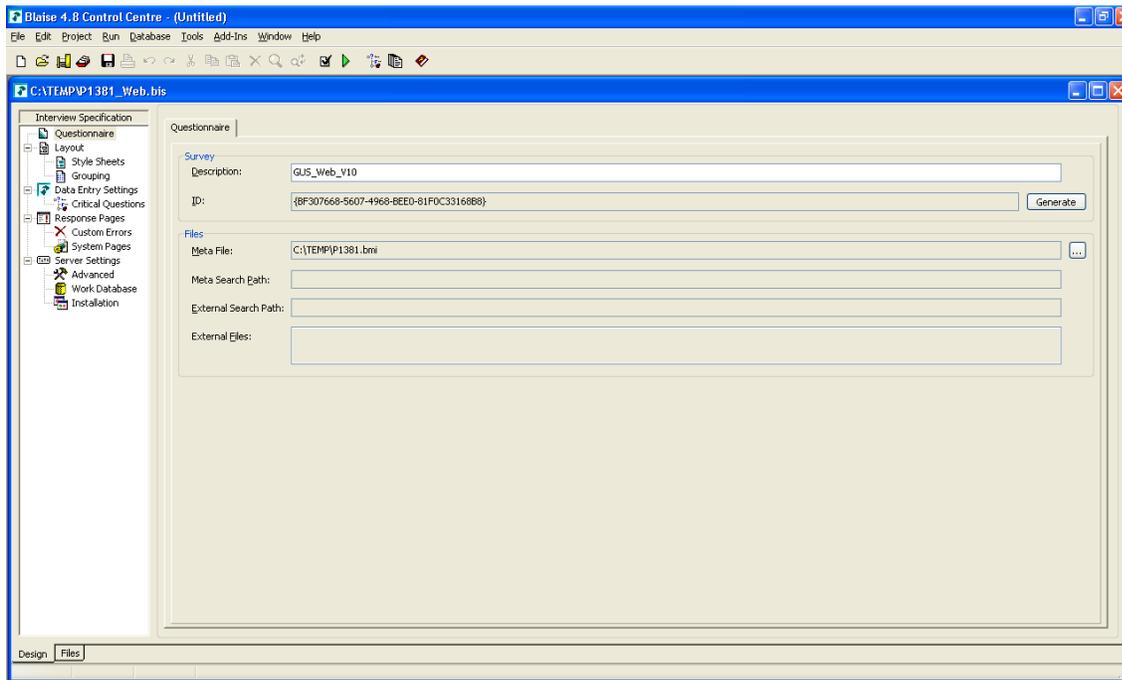
```
End1(textarea)
"Do you have any comments or feedback to give us on completing this
questionnaire?"
: TOpen
```

Datatype questions can have one of a series of date tags, depending on whether you want to use the default date picker for that device or a specifically designed one.

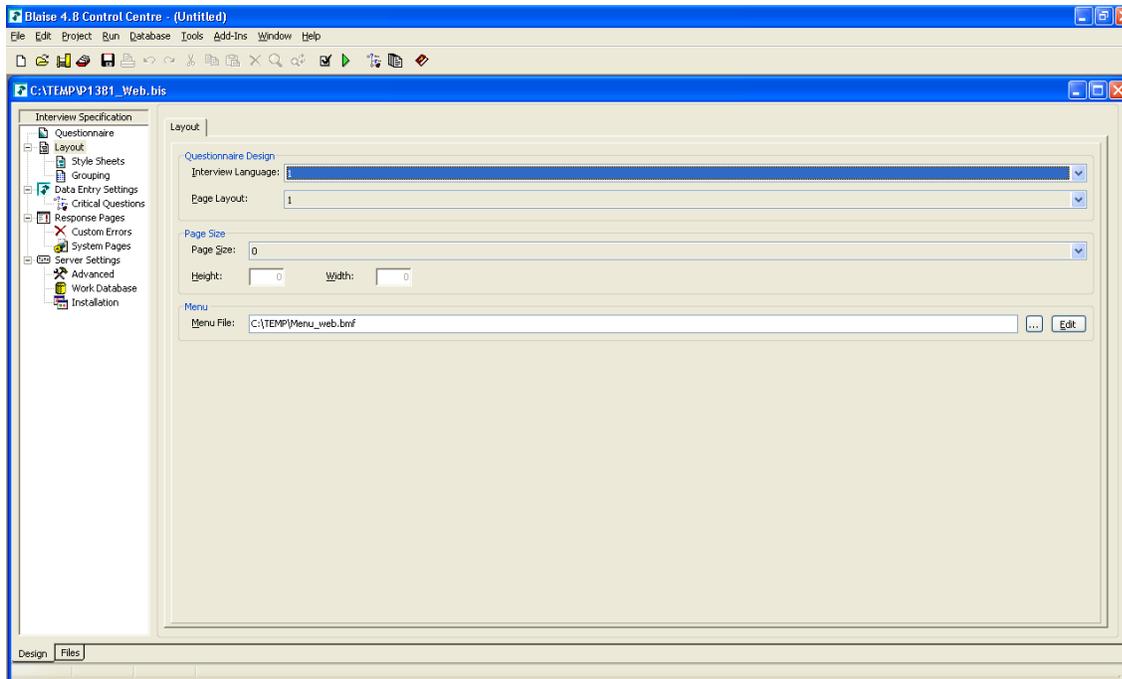
```
Pardob (datejq2)"
Please enter your date of birth
@#<br><br>@#Please enter this as DD/MM/YYYY e.g. 03/08/1975"
:DATETYPE
```

3.3 Two BIS files

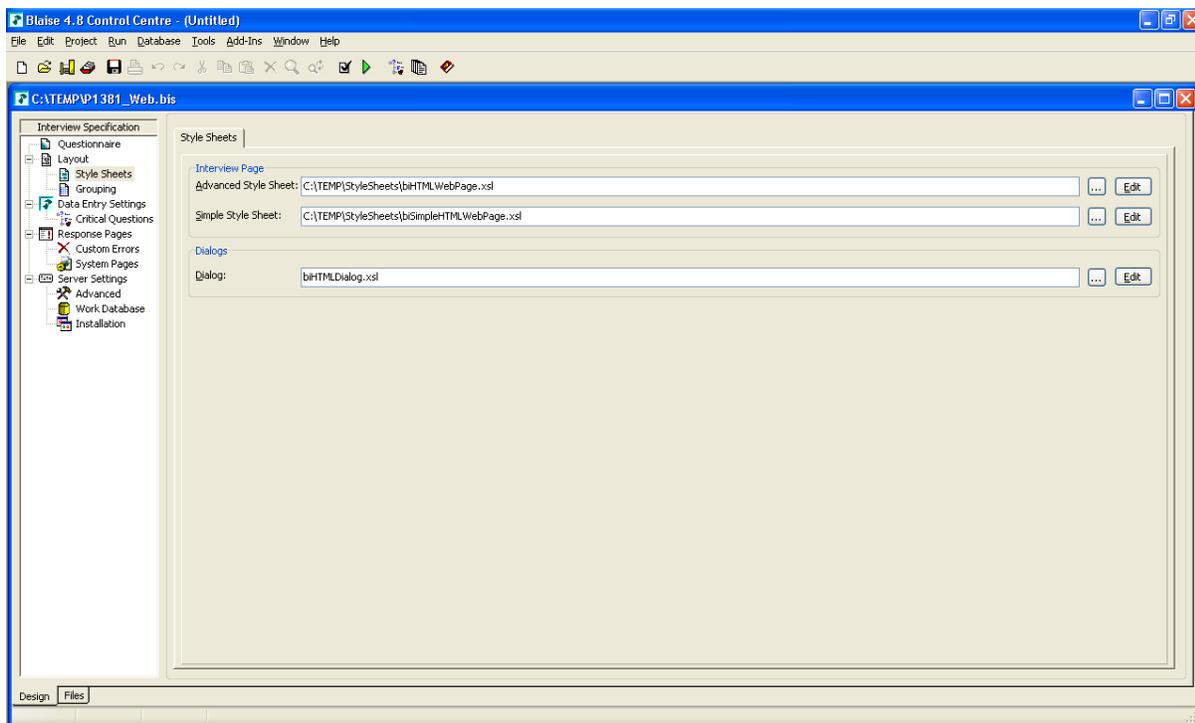
There were several places where the two versions of the BIS file differed. On the questionnaire section of the BIS file, the description and the meta file location were different. The description for web was “Gus_web_vxx” and it used the P1381.BMI meta file (this was the default meta file, but for future we would probably always use the _web, _tab notation. For tablet mode the description was “Gus_tab_vxx” and the meta file was P1381_tab.BMI. The screenshot below and all subsequent screenshots are for the web version only to illustrate where these settings need to be amended.



There were slightly different versions of the Blaise Menu File. On the layout tab, the menu file for web is set to Menu_web.BMF and is set to Menu_tab.BMF for tablet mode.

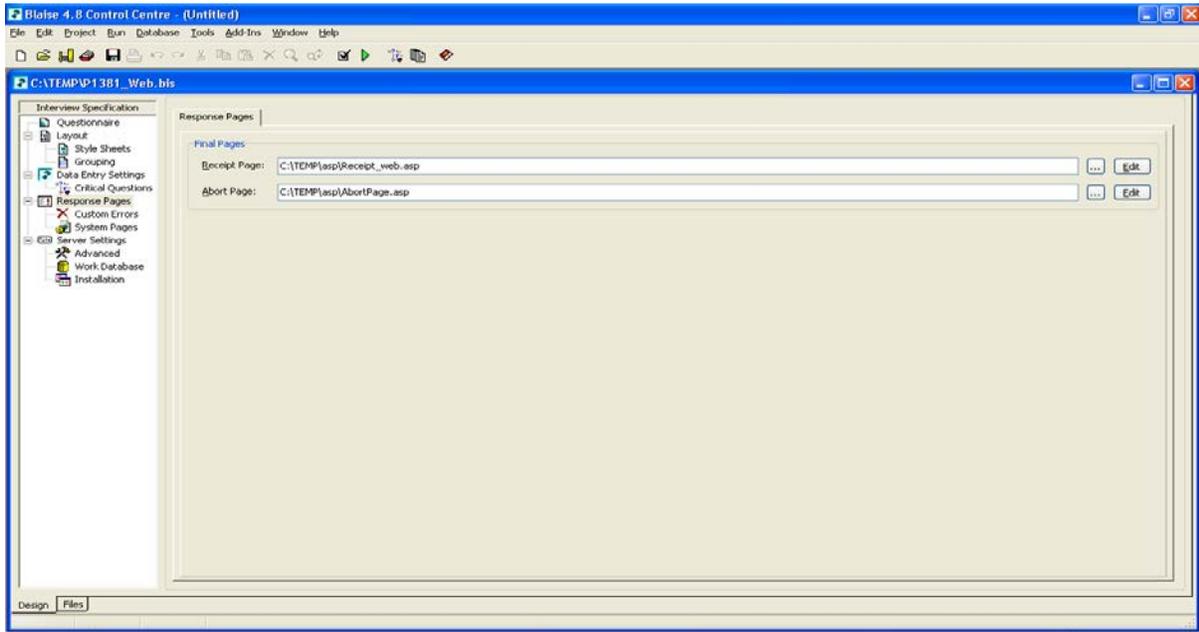


Style sheets for web were set to the Blaise defaults of biHTMLWebPage.xsl and biSimpleHTMLWebPage.xsl. There is one stylesheet for tab mode that replaces both of these – biMinimal.xsl. These are set in the Style Sheets tab.

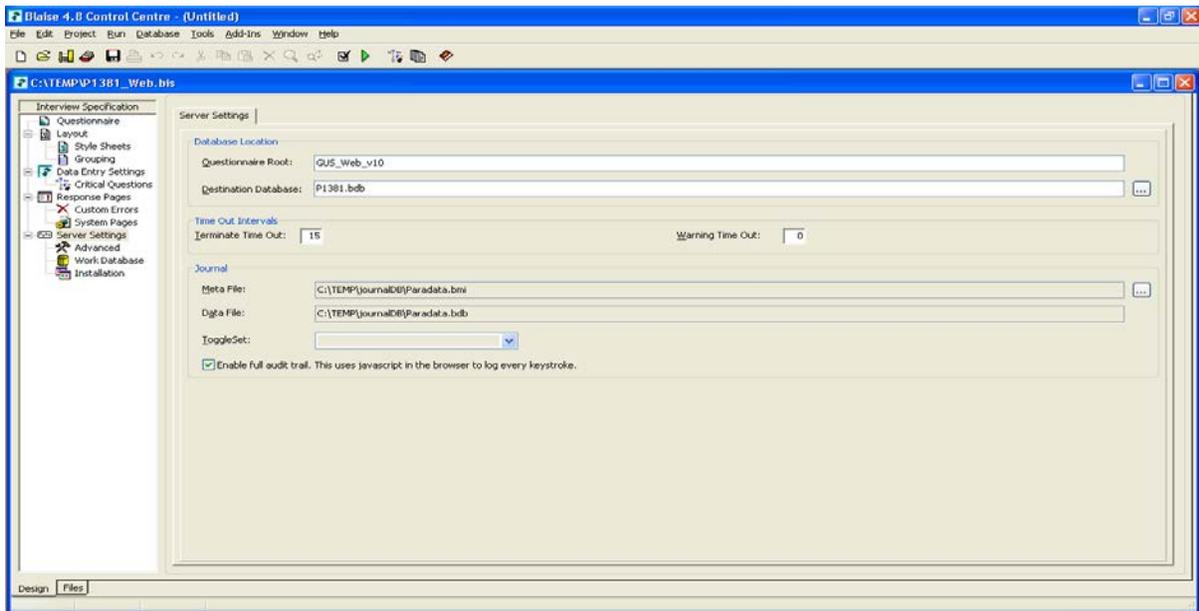


We deleted all groupings in the tablet version of the BIS file.

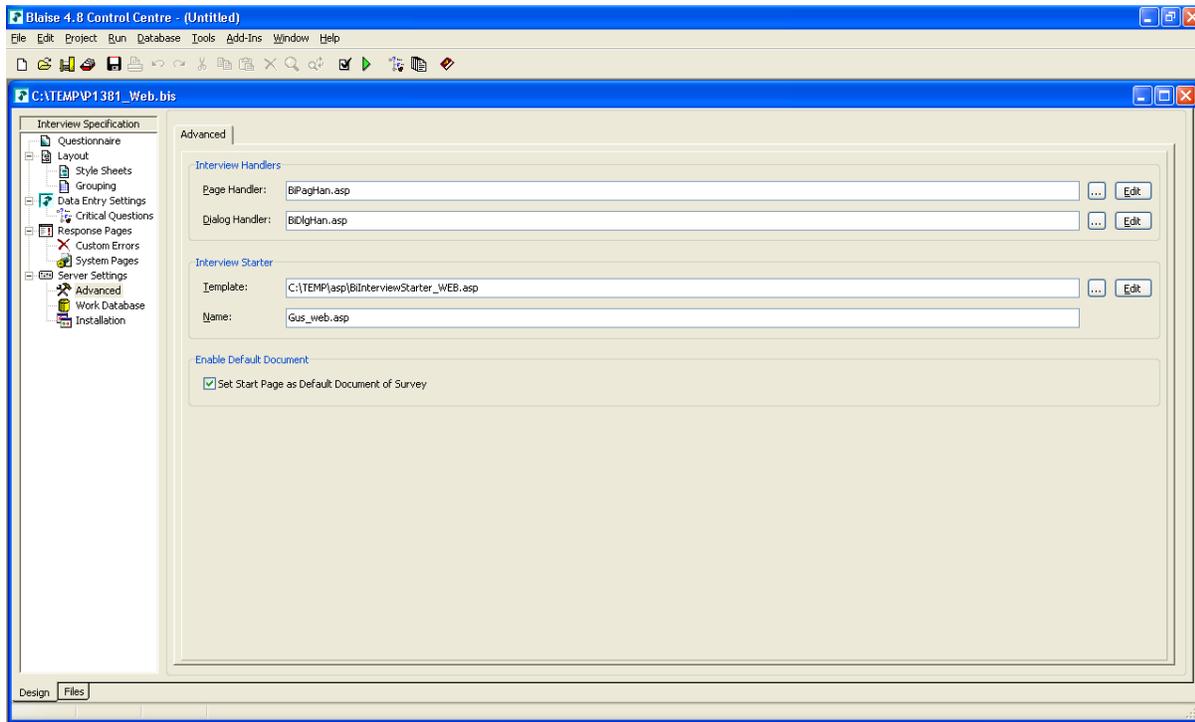
We had to create different versions of the receipt page for web and tablet to simplify the tablet version. These were set in the Response pages tab



A key difference was in the server setting tab. We had to set mode specific questionnaire roots, but we set the same default destination database for both modes as P1381.BDB (for the future this would be named xxx_web.BDB). The reason for this is that this forces Blaise to create a BOI file for tablet mode when you install the survey on a web server host and then this BOI file can be redirected.

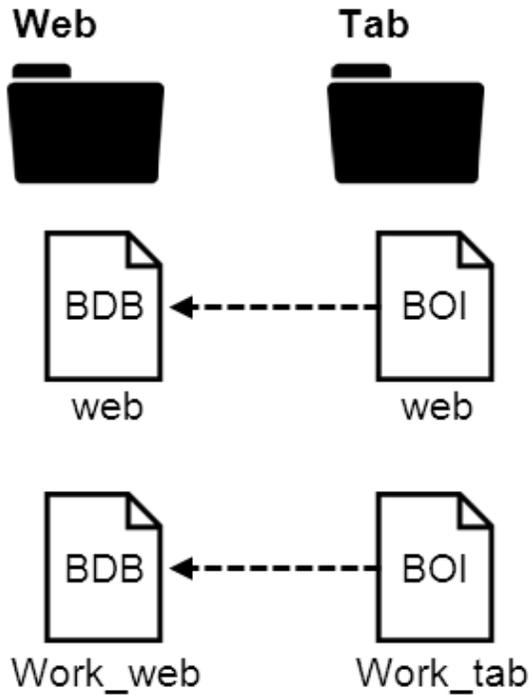


The last difference was that we had to create different starter pages for web and tablet mode so these were set in the advanced server settings tab. We amended the template starter to produce _web and _tab versions.



3.4 Installing the two datamodels

We had two datamodels and two BIP files created by compiling the BIS files, so we needed to install these on a webserver. Each datamodel/project is installed into a separate project directory, but then we amended the BOI files in the tablet mode so that they pointed to the completed and work BDB files in the web directory.

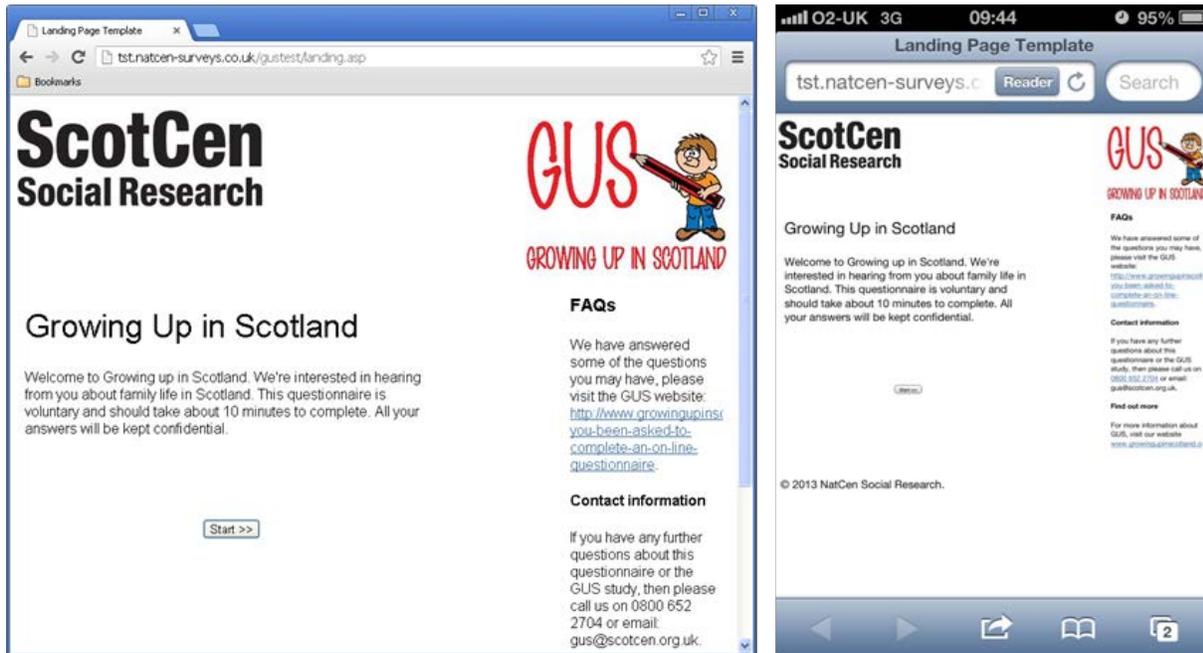


The only way that we were able to get this to work was to have set the tablet BIS file up so that it pointed to the web BDB file instead of the tab BDB file, and then this worked when we amended the location in the BOI file. This setup meant that respondents could access the questionnaire with one device, break off half way through, then reconnect with an alternative device and pick up from where they had previously got to.

During development it is very easy for installed versions to get out of sync, so the best thing to do is to make sure both datamodels are compiled, copy over any sample that is needed into the main web database, then install the web project, then the tablet project, then amend the BOI files.

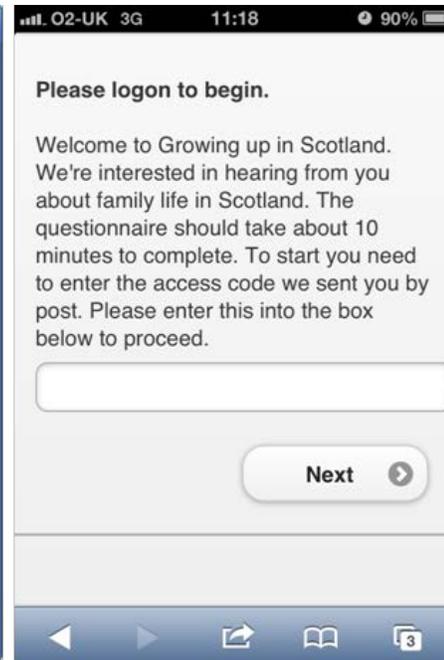
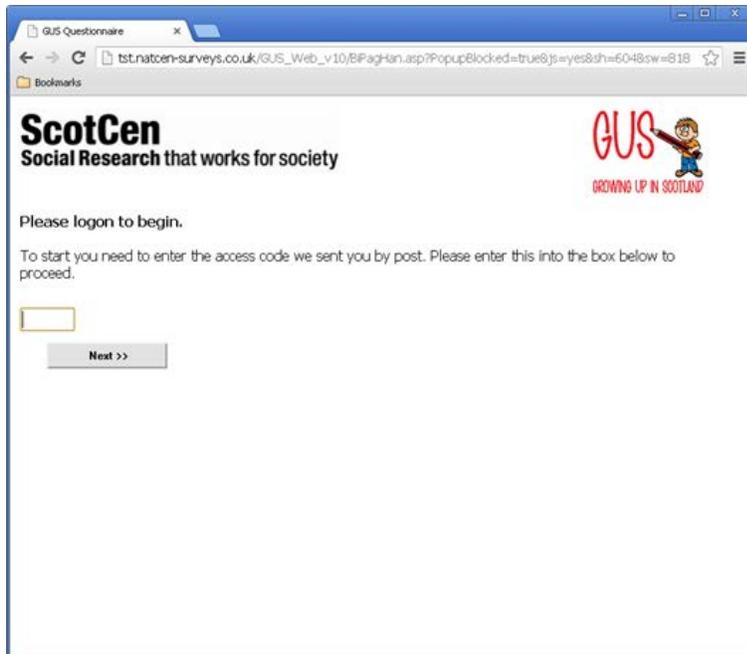
3.5 Landing page and re-direction

We decided that we would create one landing page for both device types which had a link to the GUS website and additional contact details. The screenshots below show how this displayed on a PC and iPhone screen respectively.



We added code behind the Start button which did the redirection to the appropriate device specific version of the questionnaire.

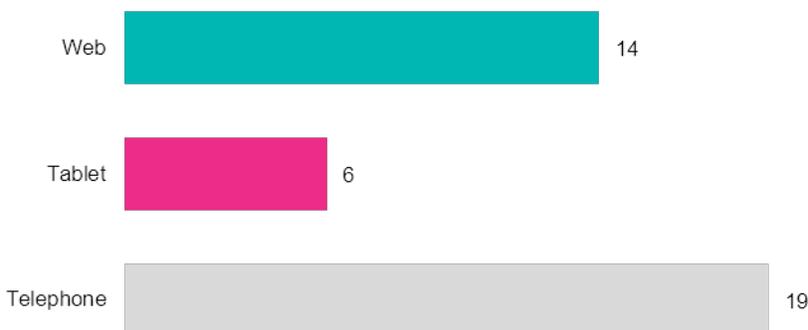
The access codes were entered as the first question in the questionnaire and hence the tablet version has the C-MoTo layout where the logos are stripped from the page, but we extended the text so that the first question wasn't too blunt.



4 Numbers using each mode

The web and tablet versions of the GUS questionnaire were set up for the pilot, so we only have a small issued sample, but we will be able to judge the impact of offering a tablet friendly version at the main stage in September 2013.

Even give the small sample size the results are encouraging. 19 respondents were chased by the telephone unit (this was the third mode that we have ignored in the bulk of this paper). Of those who completed a questionnaire themselves 6 out of 20 used a tablet while 14 out of 20 used a PC (or Mac).



It is not possible to estimate the impact offering a tablet version had on response, except that we know that 6 people would have been directed to a page telling them that the survey was unavailable on a tablet or smartphone and would have been asked to connect from a PC instead. If these proportions are replicated on the main stage, then that would be a large proportion of respondents who would initially be discouraged when they attempted to complete a questionnaire.

Using Basil for ACASI at Westat– From custom to COTS

G J Boris Allan, Peter Stegehuis, and Rick Dulaney
Westat,
Rockville, Maryland, USA

Audio Computer-Assisted Self Interviewing (ACASI) is an important data collection mode for many surveys, particularly when collecting very sensitive data. As the use of Basil for ACASI is fairly new, this paper focuses on the possibilities and capabilities of the product to be customized in various ways. We have been able to use Basil in CASI mode, ACASI mode, language-aware ACASI, and with text-to-speech (TTS). There are limitations inherent in the use of human voice recordings to read questions during an ACASI interview (“voice talent”), so we have investigated the use of TTS software and show the possibilities with TTS integration into Basil applications. We conclude some thoughts on lessons learned and next steps.

1. Basic Basil – CASI

Basil can be fully implemented without any additions as a commercial-off-the-shelf (COTS) software and apart from the specific graphics files would run on any computer with Blaise. Once prepared, all that is needed is `Manipula.exe` and command-line parameters. The program is CASI, not ACASI, because we have no audio (no sound files).

In programming Basil for CASI, two characteristics of Basil are of utmost importance as compared to Blaise – how information is displayed and how the RULES are executed:

- *Basil is page-based*, that is, screens are presented one page at a time, where the design of what appears on a page is dependent on
 - a template (an application definition) and
 - the questions that appear on that screen (the field definitions).
- *Basil is event-driven*, that is, though moving between items on a page/screen is by user choice, to move from one page/screen to the next or previous requires selection of a specific item (usually a button).

For example, a basic Basil screen looks like this:

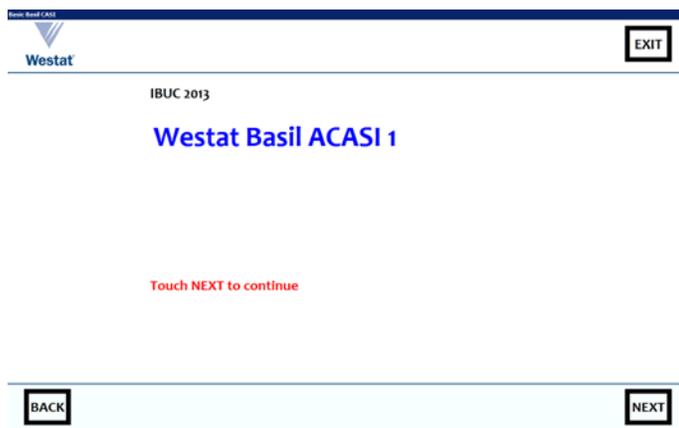


Figure 1-1: WBA 1, title screen

By following the “Touch NEXT to continue”, another screen appears (the NEXT page):

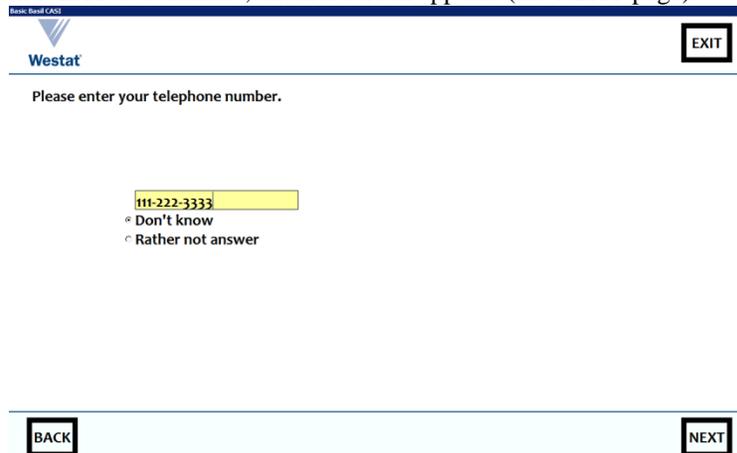


Figure 1-2: WBA 1, data-entry screen (text entry)

If you look at the two screens, there are certain common display features (from the logo to the NEXT button). These common features are defined in a Basil application section.

Here is the application section used to define the common features for the above displays (underlined text is for emphasis):

```
DATAMODEL BasicBasil

"
  <application name='app' title='Basic Basil CASI' canclose=false
windowstate=maximized width=1280 minimumwidth=1280 clientheight=800
minimumheight=800 resource='' icon='westat.ico' sizeable=false
fontface='Candara' fontbold=true fontsize=20 fontcolor=black
singleinstance=true>
  <panel name='overall' align=client color=white>
    <panel name='global' height=100 align=top>
      <rectangle left=1160 top=10 height=70 width=90 color=#000000>
        <img src='Westat_Standard_Vert.gif' left=34 top=4 width=92 height=81
stretch=true onclick='http://www.Westat.com' hint='Westat website'>
        <speedbutton name='finish' left=1170 top=20 height=50 width=70
color=#000000 caption='EXIT' onclick='blaise:save();blaise:quit()'
hint='Finish interview'>
        <line left=0 top=98 width=1280 height=2 color=#7E4500>
      </panel>
    <content-area horizontalscrollbar=false verticalscrollbar=false >
    <panel name='navigation' height=100 align=bottom color=#FFFFFF0>
      <line left=0 top=0 width=1280 height=2 color=#7E4500>
      <rectangle left=30 top=10 height=70 width=90 color=#000000>
      <rectangle left=1160 top=10 height=70 width=90 color=#000000>
      <speedbutton name='back' left=40 top=20 height=50 width=70
color=#000000 caption='BACK' onclick='blaise:previouspage()' hint='Previous
page'>
      <speedbutton name='forward' left=1170 top=20 height=50 width=70
color=#000000 caption='NEXT' onclick='blaise:save();blaise:nextpage();'
hint='Next page'>
    </panel>
  </panel>
</application>
"
```

```
LANGUAGES = BASIL "Basil"
```

Note that we have declared only one language, the *machine* language `BASIL` – we have not declared *spoken* languages such as English or Spanish.

Within the application section we define certain standard features such as font characteristics and colors. The screen is divided into three “panels”. One panel (`overall`) contains two other sub-panels (`global`, `navigation`) and a `content-area` that is used to display questions.

The `global` (top) sub-panel has two items:

- the Westat logo gif, and a link to the Westat web site
- an EXIT button that uses standard Basil commands to save the data, and exit the instrument

The `navigation` (bottom) sub-panel has two items, `BACK` and `NEXT` buttons that use standard Basil commands to save the data, and change the page (using `BACK` does not save the data, but we could modify `onclick` to add a `blaise:save()`).

Screen locations and other design aspects are given within sub-panels.

In the “content-area” we turn off scroll bars, and this screen area is where we show instrument fields. Basil provides for considerable flexibility in the development of screens. For example, the first screen (the description “WBA 1”) is defined by:

```
_StartField
  BASIL "<question>
    <label fontsize=20 halight=center width=1000 topmargin=20 text='IBUC 2013
<br><br> <font color=blue size=40>Westat Basil ACASI 1
</font><br><br><br><br><br><br><br> <font color=red size=20>Touch NEXT to
continue</font>'>
  </question>"
  : STRING[1], EMPTY
```

The `_StartField` field definition uses HTML-style formatting commands (from `<question>` to `</question>`). The label control has a `text` attribute which uses HTML-style formatting:

```
'IBUC 2013 <br><br> <font color=blue size=40>Westat Basil ACASI 1</font>
<br><br><br><br><br><br><br><br> <font color=red size=20>Touch NEXT to
continue</font>'>.
```

This is the formatted text that appears in the first screen (Figure 1-1).

The second screen (which asks for a telephone number) shows an example with an extra control (`input`) to indicate that a data value is expected (most fields will have an `input` control). The `input` control has various Basil-specific formatting commands (including an edit mask):

```
Quest41
  BASIL "<question>
    <label left=40 width=1200 topmargin=20 text='Please enter your telephone
number.'> <input visible=true halight=center left=200 top=200 width=300
height=100 editmask='000\ -000\ -0000' fontcolor=black focusedcolor=#99FFFF
showinputlineradiobutton=false fontcolor=black radiobuttonsize=40
showdontknow=true showrefusal=true showfocusrect=false>
  </question>
  "
  : STRING[10], DK, RF
```

Additional example screens show fills and rules.

Basic: Basil CASI

Westat

TELEPHONE NUMBER: 111-222-3333

Have you ever tried cigarette smoking, even 1 or 2 puffs?

Yes
 No
 Don't know
 Rather not answer

BACK NEXT

Figure 1-3: WBA 1, data-entry screen (enumeration)

```
Quest1
  BASIL "<question>
    <label height=600 width=1200 top=20 left=40 text='<font
color=blue>$Telephone</font>Have you ever tried cigarette smoking, even 1 or
2 puffs?'" <input visible=true left=200 top=200 width=400 fontcolor=black
radiobuttonsize=40 focusedcolor=#99FFFF showdontknow=true showrefusal=true
showfocusrect=false>
  </question>"
  : (Yes, No), RF, DK
```

The response categories are highlighted (#99FFFF) and we will look at the focus rectangle in the context of the next screen.

In the `text` attribute for the `label`, `$Telephone` is a fill (just as `^Telephone` would be a fill in ordinary Blaise). Basil has two types of fill (`$` and `%`) where the difference between them is not always clear in practice – the use of standard Blaise (`^`) fills is restricted to fills in type definitions.

The value of the `Telephone` field is given in the `RULES`, and depends on the value entered for the telephone number:

```
Quest41
IF (Quest41 = RESPONSE) THEN
  Telephone := 'TELEPHONE NUMBER: ' + SUBString(Quest41,1,3) + '-'
    + SUBString(Quest41,4,3) + '-' + SUBString(Quest41,7,4) + '<br><br>'
ELSEIF (Quest41 = REFUSAL) THEN
  Telephone := 'TELEPHONE NUMBER: Rather not say<br><br>'
ELSEIF (Quest41 = DONTKNOW) THEN
  Telephone := 'TELEPHONE NUMBER: Do not know<br><br>'
ELSE
  Telephone := ''
ENDIF
```

Basic: Basil CASI

Westat

EXIT

TELEPHONE NUMBER: 111-222-3333

Which of these products did you use?

(Please select all that you used.)

- Cigarettes
- Pipes
- Cigars
- Snuff or Chewing tobacco
- Nicotine patches, gum, or other nicotine product
- None of these products
- Don't know
- Rather not answer

BACK

NEXT

Figure 1-4: WBA 1, data entry screen (set)

```

Quest60
  BASIL "<question>
    <label left=40 width=1200 topmargin=20 text='<font
color=blue>$Telephone</font> Which of these products did you
use?<br><br>(Please select <u>all</u> that you used.)'>
    <input visible=true left=200 top=200 width=800 checkboxsize=40
focusedcolor=#99FFFF showdontknow=true showrefusal=true>
  </question>
  "
  : SET OF TCigTypes, DK, RF

```

In Quest60 (Figure 1-4) there is a focus rectangle around Cigars for illustrative reasons. There is no rectangle around Yes for Quest1 (Figure 1-3) because we have switched off the focus rectangle for that question (`showfocusrect=false`). The final screen:

Basic: Basil CASI

Westat

EXIT

Thank you. Touch the bar to finish

Touch here to finish

BACK

NEXT

Figure 1-5: WBA 1, confirmation screen (button)

The interview ends when the button is touched (`onclick`).

```
EndInstrument
  BASIL "<question>
    <label left=40 width=1200 topmargin=20 text='Thank you. Touch the bar to
finish'> <input type=button left=220 top=200 height=50 width=800
caption='Touch here to finish' onclick='blaise:save();blaise:quit()'>
    </question>"
  : (Yes), EMPTY
```

2. Basil CASI to ACASI with Maniplus added

The next enhancement to Basil is completed by adding sound to each input field (using WAV files) and activating the error-checking features of Basil. At this point Basil CASI becomes ACASI.

We make a small change to the beginning of the application definition:

```
"<application name='app' title='Basil ACASI -- Audio files' canclose=false
windowstate=maximized width=1280 minimumwidth=1280 clientheight=800
minimumheight=800 resource='' icon='westat.ico' sizeable=false
fontface='Candara' fontbold=true fontsize=20 fontcolor=black setups='WBA.msu
/Kmeta=WBA02;' singleinstance=true>
...
```

The only real difference is the use of a metadata-aware Maniplus executable named `WBA.msu` (**W**estat **B**asil **AC**ASI). This is the generic WBA Maniplus driver program, and individual projects can add additional programs for special purposes. The program is declared by `setups='WBA.msu /Kmeta=WBA02;'`, that is, there is a program `WBA.msu` that has a variable metadata definition, and in this case the metadata is that provided by `WBA02` (`WBA02.bmi`).

Another change is to the operation of the `NEXT` button, which actually uses a procedure from the Maniplus program declared in the `setups`:

```
<speedbutton name='forward' left=1170 top=20 height=50 width=70 color=#000000
caption='NEXT' onclick='WBA.ErrorCheck;blaise:save();blaise:nextpage();'
hint='Next page'>
```

The `onclick` attribute has an extra action, that is, the `ErrorCheck` procedure from `WBA.msu`.

The Maniplus program is aware of the current data and the current metadata of the instrument in memory by a file setting `INTERCHANGE = SHARED`.

The program performs the following actions:

- Check the rules (`CHECKRULES`) for the record in memory (a result of the setting `INTERCHANGE = SHARED`)
- Save the name of the currently active field for the record in memory to `theActive`
- Set the scope of error checking to include only those fields on the currently active page
- For the currently active page, get the number of Blaise/Basil errors and – if the count is greater than zero – for each error, get the type (`KIND`) of error

Much of the power of Basil in ACASI comes from the use of metadata-aware Maniplus (metadata such as `tempFile.GETERRORCOUNT`) even though we use only one procedure in this example.

Westat

TELEPHONE NUMBER: Rather not say

Which of these products did you use?

(Please select all that you used.)

- Cigarettes
- Pipes
- Cigars
- Snuff or Chewing tobacco
- Nicotine patches, gum, or other nicotine product
- None of these products
- Don't know
- Rather not answer

BACK NEXT

Figure 2-1: WBA 2, data-entry error (incompatible selections) – the question text and responses categories are spoken

One should not be able to answer “Cigarettes” and “None of these products”, and – when **NEXT** is touched – a defined error checking routine goes into operation:

WBA problem information

There is something wrong.

YOU CANNOT ANSWER NONE, AND ALSO CHOOSE OTHER PRODUCTS

RETURN TO QUESTION

Figure 2-2: WBA 2, error dialog box (incompatible selections) – the error text is not spoken

This is a Maniplus dialog box, and the displayed text is taken from a **CHECK** in the Basil/Blaise rules:

```
IF (None IN Quest60) AND (Quest60.CARDINAL > 1) THEN
  CHECK ERROR
  BASIL "YOU CANNOT ANSWER NONE, AND ALSO CHOOSE OTHER PRODUCTS"
ENDIF
```

Note that our Maniplus procedures prepend “There is something wrong” to the defined error text in the displayed text. If none of the answers are chosen (a common user action), we get a different dialog box, that is, the dialog box for a **ROUTE** error:

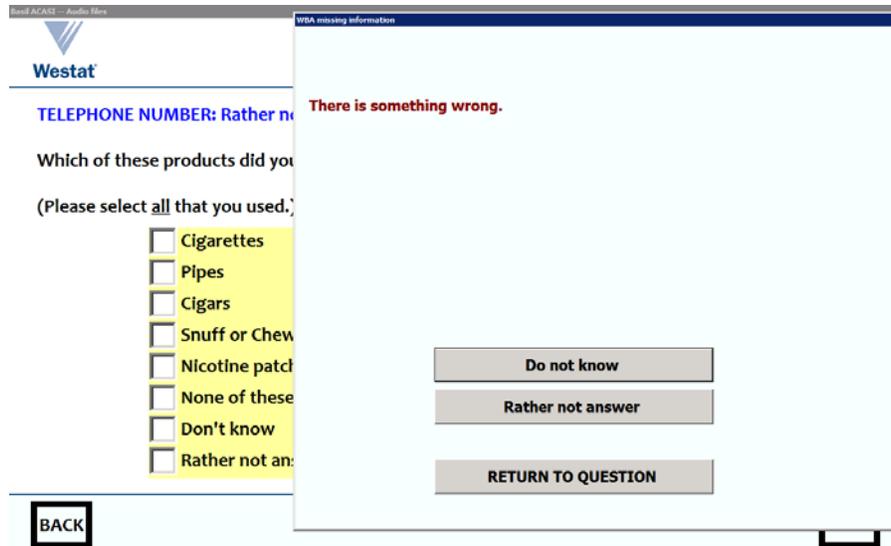


Figure 2-3: WBA 2, error dialog box (missing data) – the error text is not spoken

It is clear that we can change the text displayed, depending on the type of error (ROUTE SOFT HARD), and ErrorCheck has different dialog boxes for each type (we based our procedure on a sample provided with the Basil distribution).

We have also declared audio output as Quest60Audio, composed of many sound files in a folder Media\English. We have the main question file (Quest60.wav), files for each option, and files for DK and RF, plus a silence file. The sound files are played in a continual loop (including silence), and the audio loop is stopped when a key is pressed. We start the audio by attaching it to the <input ... onenter= ... > control/attribute.

3. Basil ACASI becomes language-aware

So far we still have only one language (BASIL) which we conveniently express in English. Many of our studies are multi-language, and so we need to be able to switch spoken languages. The Basil programming language always operates with one machine language (BASIL) with the question text being displayed by the <label ... text= ...> control/attribute in whatever spoken language is provided by the fill. At Westat, we have resolved this problem of changing spoken languages by using procedures defined in WBA.msu that fill the <label ... text= ...> control/attributes, and select folders, depending on the currently chosen spoken language.

A language option is added to the screen as shown below.



Figure 3-1: WBA 3, title screen (with language option)

After the user selects the language, the screen text is changed.

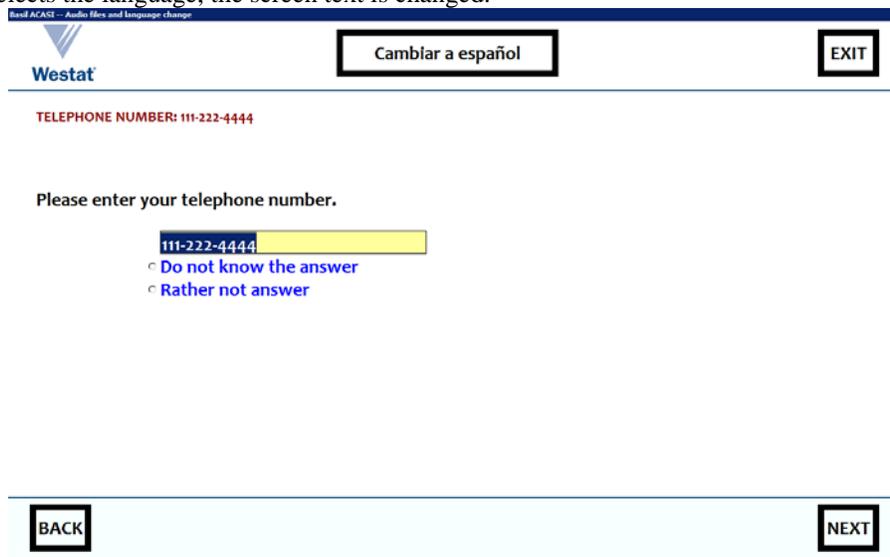


Figure 3-2: WBA 1, data-entry screen with English display (text entry)

Touch the button that changes to Spanish:

Basil ACASI -- Audio files and language change

Westat

Change to English

EXIT

TELEPHONE NUMBER: 111-222-4444

Por favor anote su número de teléfono.

111-222-4444

No sé la respuesta

Prefiero no contestar

BACK

NEXT

Figure 3-3 WBA 4, data-entry screen with Spanish display (text entry) – the question text is spoken in Spanish

Our question is now displayed in Spanish.

The field (Quest60) that asks about types of tobacco product has an attached type:

```
Type
  TCigTypes =
    ( Cigarettes (1)  BASIL "^TCigTypes_Cigarettes" ENG "Cigarettes" ESP
"Cigarillos",
    Pipes (2)  BASIL "^TCigTypes_Pipes" ENG "Pipes" ESP "Pipas ",
    Cigars (3)  BASIL "^TCigTypes_Cigars" ENG "Cigars" ESP "Cigarros",
    Snuff (4)  BASIL "^TCigTypes_Snuff" ENG "Snuff or Chewing tobacco"
    ESP "Rapé o tabaco",
    Other (5)  BASIL "^TCigTypes_Other"
    ENG "Nicotine patches, gum, or other nicotine product"
    ESP "Parches de nicotina, chicle, u otro producto de nicotina",
    None (9)  BASIL "^TCigTypes_None" ENG "None of these products"
    ESP "Ninguno de estos productos"
  )
  ENG "Which of these products did you use?<br><br>(Please select <u>all</u>
that you used.)"
  ESP "¿Cuál de los siguientes productos usó usted?<br><br>Por favor elija
<u>todos</u> los que usó."
  : SET OF TCigTypes, DK, RF
```

We were able to set it up so that , an error message for this field, in English and in Spanish, now shows the applicable question text as well as the error message:



Figure 3-4: WBA 3, error dialog box with question text fill, in English (incompatible selections) – the error text is not spoken

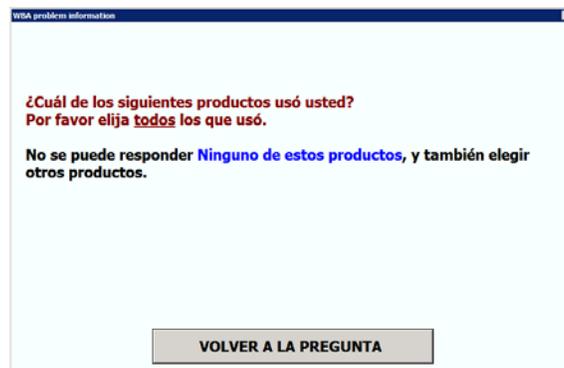


Figure 3-5: WBA 3, error dialog box with question text fill, in Spanish (incompatible selections) – the error text is not spoken

The CHECK is:

```
IF (None IN Quest60) AND (Quest60.CARDINAL > 1) THEN
  CHECK ERROR
  ENG "You cannot answer <font color=blue>None of these products</font>,
and also choose other products."
  ESP "No se puede responder <font color=blue>Ninguno de estos
productos</font>, y también elegir otros productos."
ENDIF
```

If we do not make any entries for this question, we get dialog boxes in English and in Spanish:

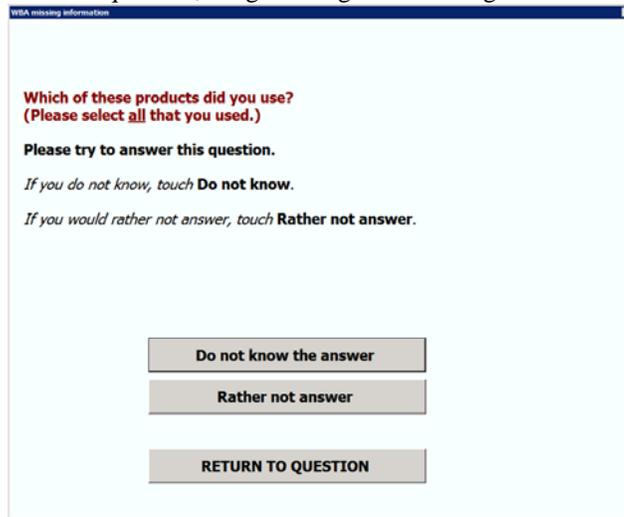


Figure 3-6: WBA 3, error dialog box with question text fill, in English (missing data) – the error text is not spoken

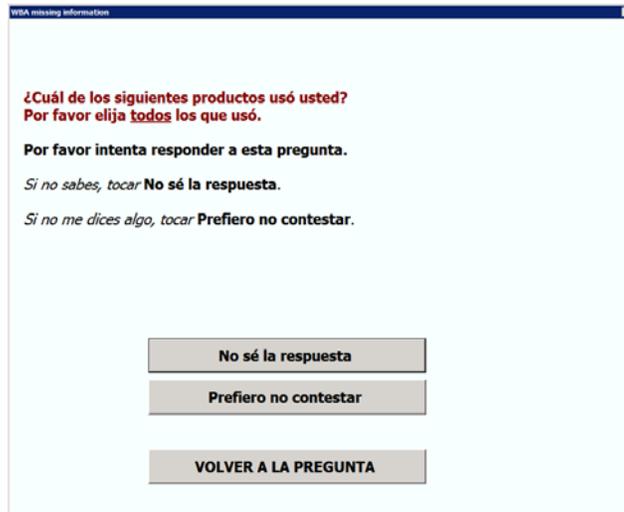


Figure 3-7: WBA 3, error dialog box with question text fill, in Spanish (missing data) – the error text is not spoken

There is no sound attached to these dialog boxes.

4. Adding text-to-speech to language-aware Basil

As a next step, we were able to add text-to-speech (TTS) to Basil. An example screen is shown below.

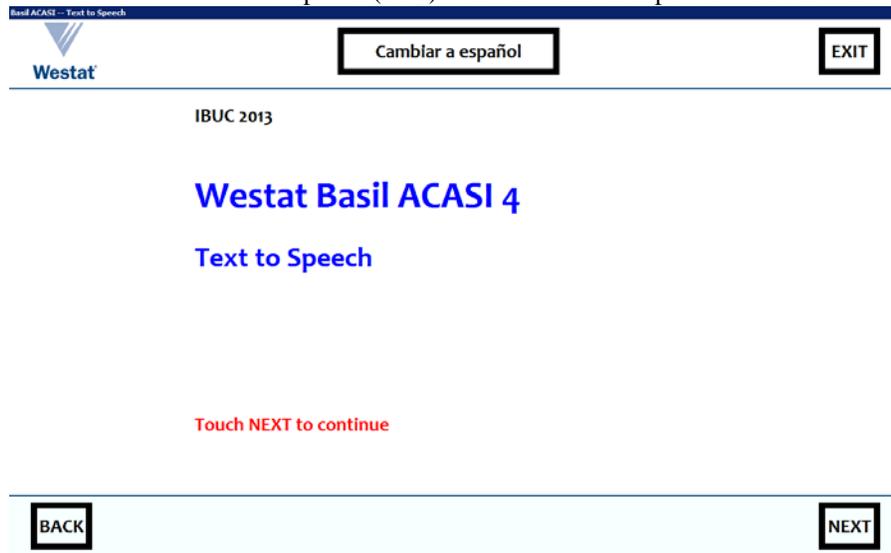


Figure 4-1: WBA 4, title screen (with language option) – “Touch NEXT to continue” is spoken

And you hear “Touch next to continue”. The definition of this field shows a new procedure, `WBA.Speak`, being called.

Because we have to be able to stop and start speech on demand, the application section refers to many TTS-related procedures:

```
DATAMODEL BasilACASITextToSpeech

"<application name='app' title='Basil ACASI -- Text to Speech'
cancelclose=false windowstate=maximized width=1280 minimumwidth=1280
clientheight=800 minimumheight=800 resource='' icon='westat.ico'
sizeable=false oncreate='WBA.ChangeLang(''ENG'');' fontface='Candara'
fontbold=true fontsize=20 fontcolor=black setups='WBA.msu /Kmeta=WBA04;'
singleinstance=true>
  <panel name='overall' align=client color=white>
    <panel name='language' height=100 align=top>
      <speedbutton left=480 top=20 height=50 width=300
caption='$__Language_Fill'
      onclick='WBA.StopSpeaking(""); WBA.GotoStart();
WBA.ChangeLang(''OTHER''); WBA.GotoOldActive();'
      hint='$__Language_Fill'>
      <rectangle left=470 top=10 height=70 width=320 color=#000000>
      <img src='Westat_Standard_Vert.gif' left=34 top=4 width=92 height=81
stretch=true
onclick='WBA.StopSpeaking("");http://www.Westat.com'
      hint='Westat website'>
      <speedbutton name='finish' left=1170 top=20 height=50 width=70
color=#000000
      caption='EXIT' onclick='WBA.StopSpeaking(""); blaise:save();
blaise:quit()' hint='Finish interview'>
      <rectangle left=1160 top=10 height=70 width=90 color=#000000>
      <line left=0 top=98 width=1280 height=2 color=#7E4500>
    </panel>
  <<content-area horizontalscrollbar=false verticalscrollbar=false>>
```

```

    <panel name='navigation' height=100 align=bottom color=#FFFFFF0>
      <line left=0 top=0 width=1280 height=2 color=#7E4500>
        <rectangle left=30 top=10 height=70 width=90 color=#000000>
        <rectangle left=1160 top=10 height=70 width=90 color=#000000>
        <speedbutton name='back' left=40 top=20 height=50 width=70
color=#000000
          caption='BACK' onclick='WBA.StopSpeaking(" ");
blaise:previouspage()'
          hint='Previous page'>
        <speedbutton name='forward' left=1170 top=20 height=50 width=70
color=#000000
          caption='NEXT' onclick='WBA.StopSpeaking(" "); WBA.ErrorCheck;
blaise:save();blaise:nextpage();' hint='Next page'>
      </panel>
    </panel>
  </application>
"

```

You will notice that we make great use of `WBA.StopSpeaking(" ")`, otherwise we are liable to continue speaking old text when we move on to new text, because you have asynchronous speech when using alien Visual BASIC procedures.

Basil can handle complex field definitions and rules execution paths. We found that Basil gives an impressive inventory of features for customizing instruments.

5. The potential of Basil for ACASI

ACASI provides a viable new methodology for self-interviewing, and – especially with text-to-speech – great flexibility in what users hear. Blaise Basil has shown to be adaptable as an ACASI platform and to enable Blaise users to develop sophisticated custom applications to meet study needs. Some observations from the process include:

- Apart from the general look and feel defined in the application section, the only instructions for displaying a field in a Basil instrument are those given as `BASIL "<question> ... </question>"` where the actual text displayed is given in the `text=` attribute. Other interview languages (say, `ENG ESP`) do not affect Basil.
- Though other interview languages are not recognized by Basil, we can use metadata-aware Maniplus to look at those other languages, and that is the way we can change the content of a `BASIL text=` attribute.
- Because Basil has most of the flexibility of Blaise, with some useful extra features, appearance is very customizable and we can react to the desires of clients and methodologists.
- If we use Basil for ACASI, then there are cost savings for text-to-speech as opposed to using pre-recorded sound files (sometimes known as “voice talent”) – there are valuable time savings in the development cycle by not having to record all questions, answers and fills, especially when using multiple languages. It is also simpler to use TTS with dialog boxes.
- With Basil/Maniplus we control layout and can respond to events within one control environment and do not need to use other features such as the alien router or special DLLs to complete these operations.
- With TTS technology improving at a rapid pace its use for ACASI will become ever more sensible and cost-effective. While there are still some remaining challenges with regard to attaining the full flexibility we would like to achieve with integration of TTS within Basil ACASI, we believe that Basil/Maniplus already allows for a very satisfactory level of control over TTS events .

¹ We would like to thank Joseph Allen and Bill Copeland of Westat for their great help in the preparation of Westat Basil ACASI. The Blaise team were always exceedingly helpful and prompt in their assistance – thanks to Lon Hofman and Roger Linssen in particular.

CARI Recorder Component Application in Michigan

Youhong Liu, Gina-Qian Cheung Survey Research Center, University of Michigan

1. Introduction

The Computer Audio Recorded Interviewing (CARI) Recorder Component is a new function offered by Blaise in version 4.8.4. It can be utilized when the built-in audio capabilities are inadequate for recording. In this paper, we will discuss the application of this function at the University of Michigan. We will also discuss how the output of this function can be used to conduct quality control tasks.

2. Background

CARI is a technique that records the conversation between the interviewer and the respondent on the computer during an interview. The Blaise Data Entry Program (DEP) began to support CARI in version 4.8.2. The built-in audio function provides sound recording and screen capturing during the data entry process. In Michigan, we have utilized a system developed in-house, the Digitally Recorded Interviewing (DRI) system, to record video files since 2007. While the DRI system works well to meet our needs, some inadequacies exist:

- The Blaise Alien Router used for DRI has some inconsistencies – it may miss some fields in recording, while on the other hand it adds extra field to the output;
- The Alien Router may slows down the interview process for some datamodels
- The log file provided by the system does not provide any linkage to Blaise data model fields thus; it (the log file) cannot be used by the evaluation system for quality control purposes.

We started to investigate the usage of CARI when Blaise 4.8.2 was released. We chose not to incorporate it in our environment at that time because, at that point, Blaise CARI was only capable of producing audio and image files but not video files. In Michigan, we utilize the video files to view the interview process primarily for quality control purposes, as well as for other reasons.

In November 2011, several organizations gathered at Michigan to discuss all aspects of Blaise CARI/DRI functions. During the meeting, Michigan requested that Blaise extend its CARI functions to be more flexible in order to meet our requirements. At the beginning of 2012, Blaise Beta version 4.8.4 added a new Recorder Component that can be used to control a COM-interface. This COM-interface can be used when the built-in audio capabilities are inadequate, e.g., when you need to record interviews conducted via voice-over internet protocol (VOIP), control a video camera or any other devices. In our case, we would need this COM-interface to control an external program, - Camtasia, for video recording the computer screen during the interview. Michigan worked closely with Blaise developers to define the system requirements and to help with testing.

During 2012, one of the major studies in Michigan – the Panel Study of Income Dynamic (PSID) was in its development stage for its 2013 panel survey. We decided to implement this new Blaise CARI function to record interviews. The project launched the production interview at beginning of 2013, and it is still on-going. We found the new Blaise CARI system performed very well and are very pleased with its outcomes.

3. About CARI COM-interface

The Blaise CARI COM-interface defines a set of methods in which your own recorder object can be supported without dictating anything about the implementation. COM is a language-neutral way of implementing objects. It can be used with almost any programming language including all .NET languages through .NET COM. With this new interface, we were able to drive the Camtasia recording software to start and stop recording interview questions. The language used is C#.NET.

4. Michigan CARI Recorder Component

4.1. Start/Stop Camtasia in Sample Management System

The existing DRI system developed at the University of Michigan utilizes the Sample Management System (SMS) to launch the Camtasia recording, handle video file renaming, detect the rendering process, and copy video files to their final location. Based on the project requirement changes, the SMS can also implement different algorithms to select cases to be recorded, and to update its database status upon exiting the Blaise DEP. The decision was made to continue to implement these steps outside of Blaise CARI.

4.2. Create an ActiveX Library (DLL)

The ActiveX Library is called by the Blaise CARI Setting Interface (.BCI) file. Below are the methods that need to be implemented by the ActiveX Library:

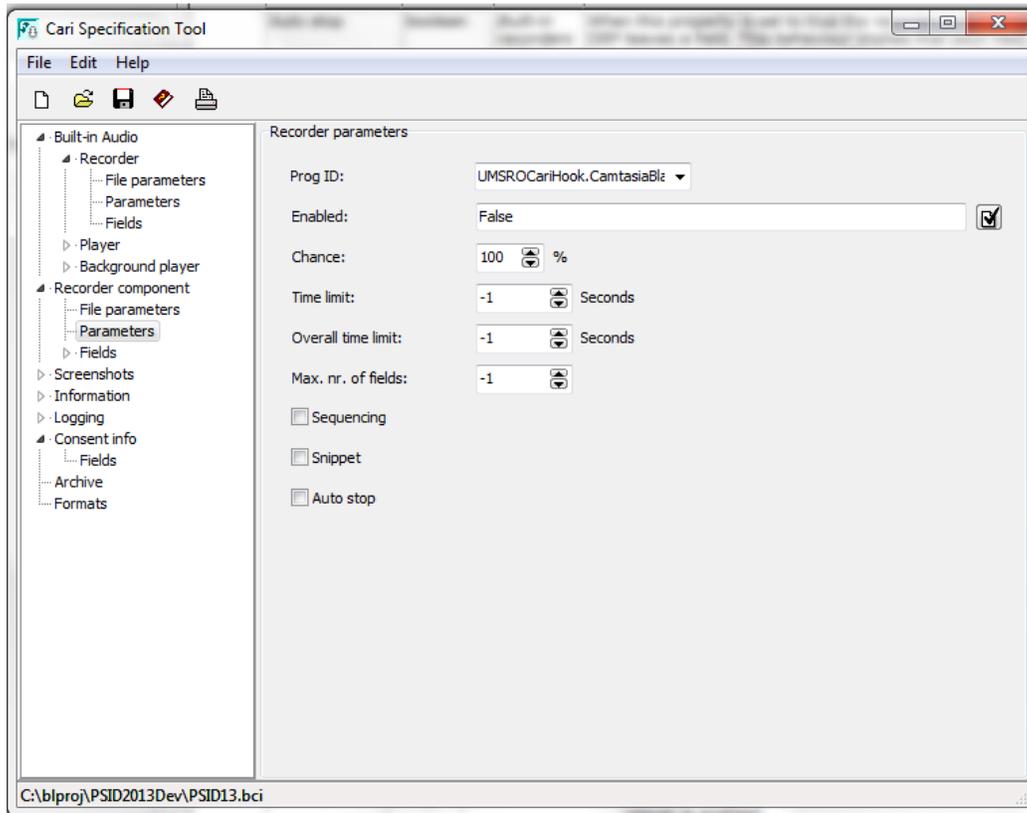
- **InitRecorder:**
A procedure called when the DEP is started.
- **EnterForm:**
A procedure called when the DEP enters the form.
- **Start (Filename):**
A procedure called when the DEP arrives at a field and the recorder is enabled. The filename passed to this procedure is a string that consists of the current fieldname and file settings defined in the CARI-specification file, e.g., extension and temporary path name.
- **TimeElapsed as integer:**
The function TimeElapsed returns the number of seconds the recorder is running.
- **Status as TStatus:**
Status returns the status tsIdle (=0) when the recorder idles, tsPaused (=1) when the recorder is paused, or tsPlaying (=2) when the recorder is running.
- **Stop:**
A procedure called to stop recording.
- **LeaveForm:**
A procedure called when the DEP leaves the form.
- **ExitRecorder:**
A procedure called when the DEP exits.

Since most of the Camtasia recording programming is handled outside of Blaise, the Michigan DLL implementation was very concise. The codes that needed to be added to the system were limited to three methods: Start () to start or resume recording, Stop () to pause recording to Camtasia Recording, and ExitRecorder () to stop Camtasia Recording when the DEP exits.

4.3. Set Up the BCI File

Next, a significant amount of effort was required to investigate how to set up a Blaise CARI Setting Interface (.BCI) file to work with the DLL to capture the video files based on project requirements.

4.3.1 Parameters

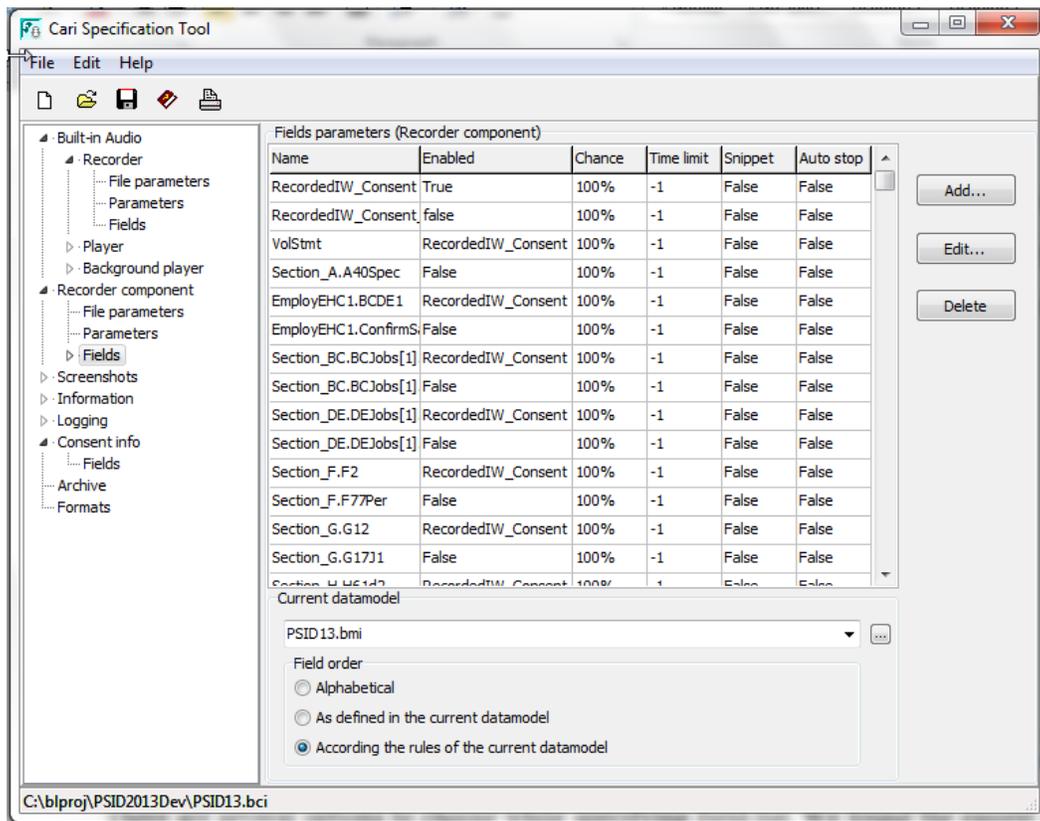


- **Prog ID:** The first step to enable Recorder Component is setting its Prog-ID. By default, the components' prog-ID is left blank, meaning that no COM component is used. In order to use our own recording component, click the Prog-ID's dropdown box to select the Prog-ID 'UMSROCariHook.CamtasiaBlaiseRecord'.
- **Enabled Status:** A Boolean expression that evaluates true or false. If true, the recorder is enabled. If false, we can enable/disable the recorder at the field list level.
- **Time Limit:** Use this property to set the recording duration (in seconds). The value is set to be -1 so the recorder will continue to record until it is stopped for any other reason.

- Overall Time Limit: Recording stops when the overall recording time for a case exceeds the specified limit in seconds. In our situation, the time limit and overall time limit have the same effect.
- Max. nr of fields: Recording stops when the recordings per case exceed the specified amount. It is set to -1 since we want to record an unlimited number of fields.

4.3.2 Fields

In most cases, we do not want to record the entire interview. As in the Built-in Recorder, a list of fields to record can be specified. For each field identified by its name, you can overrule the recording time limit and a recording chance or even enable or disable the Recorder Component.



There are several options to choose from while specifying the field list. We found the easiest way is to record ranges of consecutive fields, resulting in limited fields added to the list. To record a range of fields, we add fields to the list that enable or disable the recorder upon arrival in these fields. These fields work as a toggle. Once the recorder is enabled, it will stay enabled until it is disabled. Below are a list of summary points regarding the field list:

- We found it is the best to choose “According the rules of the current datamodel” option while defining the field list. This will ensure that the field list is defined in the right order..
- The first field is the consent question and it is always enabled. The other fields enabled statuses are dependent on the Consent question, e. g., RecordedIW_Consent = Consent. The CARI’s “Consent Info” feature can also be used for the same purpose.

- The “Auto stop” settings are set to False for all fields on the list. This means that the interview will be recorded in one file during a Blaise session. For example, if an interview is completed in two sessions, there will be two recordings associated with the interview. We think this is the best setting for video recording in order to minimize the number of files and the storage size.

4.3.3 Log Files

The Recorder can keep track of all its actions in two different log files - a *Main* log file and a *Case* log file. In our implementation, we primarily use the main log file. The log files are very useful in several aspects:

- They can be used to verify the CARI recording system to make sure the recording files contain the fields we defined in the .BCI file and vice versa.
- The timestamps and other start/stop information can be used later for quality control purposes, i.e. to jump to recorded field locations.

5. Developing a Quality Control System for CARI

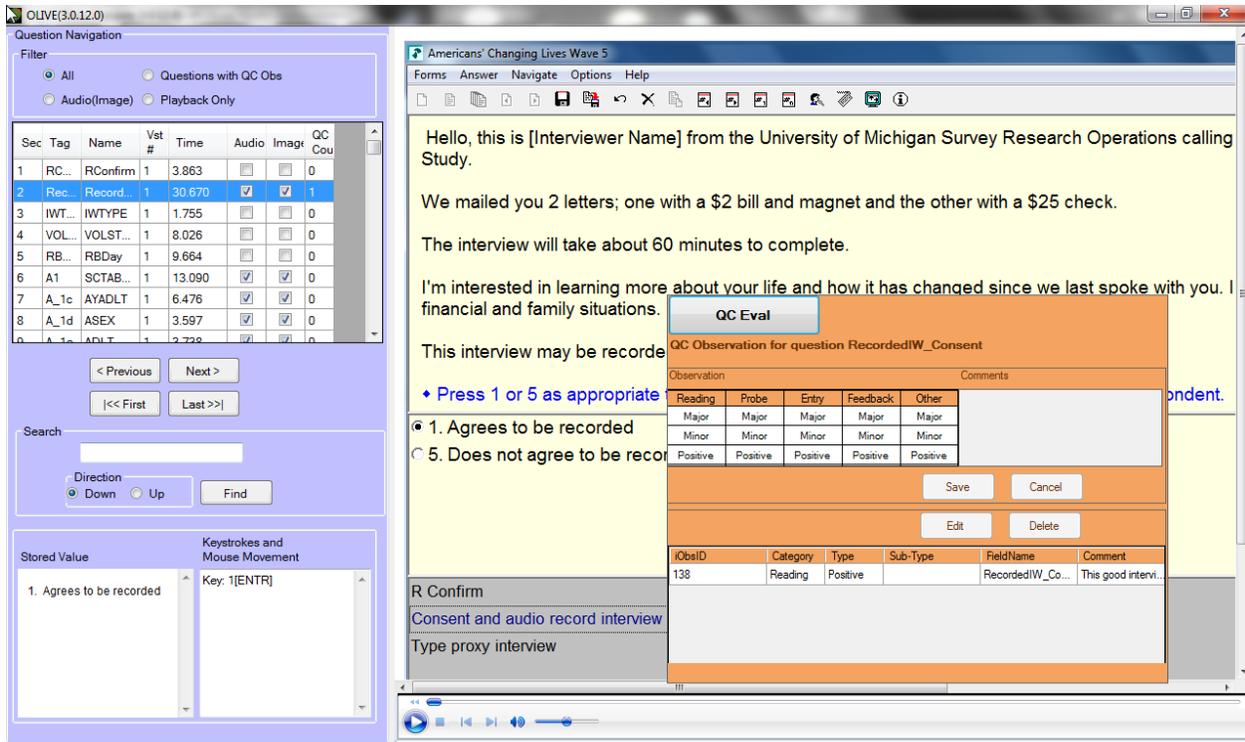
In Michigan, we have been utilizing a quality control (QC) system to evaluate interviews recorded in the existing DRI system. Because the video files produced by the new CARI system are similar, the same quality control system can be used for both CARI and DRI files. Since the file produced by the CARI system is more closely related to other Blaise files, e.g., the data model and audit trails, we think a better system can be built to facilitate the QC Process.

Some early work has been conducted recently. In July this year, a lab study implemented CARI’s built-in audio recording option. A system is being built to quickly to evaluate audio output for this study. Although the current program can only work with audio and image files, we believe that we can add the video evaluation component relatively easily because both types of output share similar features.

Below is a screenshot of the new Michigan QC system. On the left, the case adt file is used to build the field navigation list. When a staff member reviewing the file for quality control purposes lands on a field that has both audio recording and image files, the right window will show the image file and the audio file will be played back. If no recording file exists for that field, the system will show the question text and answers based on the data model and adt files. The quality control staff are able to enter comments in the pop-up box.

It is extremely helpful to be able to link the adt file to the recording. The system knows exactly which question is under review. This field information is saved in the QC database along with comments for future reporting purpose. The keystroke information from the adt is also displayed to aid QC evaluations.

At this point, the built-in audio outputs one file per field, so we do not need to use the CARI log file to find the field location. When the video function is implemented for the system later, the CARI timestamp in the log file should be able to guide the system to jump to specific field locations in a video.



6. Conclusion

PSID is the first project that implemented this new CARI system. We have already seen many advantages over the previous system. We have a few applications developed for verification of CARI output files, and have found the output video files are accurate. Several Alien Router interfaces were programmed in the PSID data model to help collect special data, for example, the Employment Event History calendar. With the CARI Recorder Component, all user actions are recorded. This is not only useful for interviewer evaluation and quality control purposes, but it is also beneficial as it can aid in detecting usability issues..

7. References

Statistic Netherland Blaise Guide

Rebecca Gatward, Gina-Qian Cheung, Patty Maher, DRI/CARI Explorations, International Blaise User Conference, 2011.

SESSION 5 PAPER 3 THE APPLICATION OF CARI IN VERIFICATION SYSTEMS

Implementing Medical Care Surveys Using DEP and Manipula

Roberto Picha, Curtis Ziesing, US Census Bureau

1. Introduction

The National Center for Health Statistics (NCHS) sponsors the National Hospital Ambulatory Medical Care Survey (NHAMCS) and the National Ambulatory Medical Care Survey (NAMCS). These surveys collect information about the health problems of ambulatory patients and the treatment given to them. The NHAMCS collects this data from hospital emergency rooms and outpatient departments, and the NAMCS collects the data from office-based physicians.

Both the NAMCS and the NHAMCS have two phases of data collection. The first phase of data collection involves an "induction" interview that collects information about a physician's practice in the NAMCS and the services offered by the various departments of a hospital in the NHAMCS. In the induction phase, the instruments sample either physician's practices for NAMCS or the various eligible departments for NHAMCS. The second phase of data collection is the "abstraction" interview which collects patient information in the Patient Record Form (PRF) section of the abstraction instruments. The abstraction interview can collect up to one hundred individual PRFs with each PRF spanning multiple pages within the instrument.

Both surveys collect similar data and have similar functionality. This paper discusses three key features that the NAMCS/NHAMCS instruments have in common and how they were implemented when the surveys migrated from paper data collection. These are:

- The instrument's extensive use of Manipula to aid data collection in different areas.
- The customization of the Blaise Mode Library layout to mimic the paper PRFs.
- The requirements for navigation within and between the PRFs and having the forms look and feel the same across the two surveys.

2. Approach to Design and Development

2.1 Using Manipula in the data Collection

One of the requirements for the NAMCS and NHAMCS instruments is that the FR (i.e., the Field Representative who conducts the interview) be able to update certain information within the instruments at any point during the data collection either via a function key or as part of the normal routing within the instrument. Using simple parallel blocks displayed as tabs was one possible approach that would have allowed the FR to jump and collect information such as office schedules and office contact information. Considering the flexibility required by NCHS, the Authoring team proposed collecting this data via Manipula dialog boxes. The advantage of the "Manipula dialog box" approach is that the instrument can also automatically bring the questions on route at the time they are needed based on the routing instructions and via a function key. There were advantages, disadvantages, and some challenges to implementing the required functionality in this manner.

Advantages of using the Manipula Dialog box:

- This approach required using only one set of fields in the instrument. This means that there would not be the problem of keeping multiple sets of fields in sync with the same data.
- The dialog boxes are launched either via a function key or by using an event key handler.
- The dialog boxes offered some additional flexibility. For example, the sponsors did not want all the fields in the dialog box to be editable when the dialog box was launched from the normal routing within the instrument. The sponsor wanted the dialog box to be context sensitive and allow an FR to edit only the field that pertained to the lead in question. This was handled by the “enabled dialog element” for both controls and buttons allowing the authors to make the fields read-only or editable using a conditional statement. Doing this allowed the use of the same dialog box without having to creating multiple dialog boxes for all the different situations. Figure 1 shows how the dialog box used to enter Staff Contact information.

Update Staff Contact Info			
Staff Name 1:	Jane Doe	Phone 1: (111)333-3333 x 3333	Type: Work
Title:	Head Nurse	Phone 2: () - x	Type:
		Phone 3: () - x	Type:
Staff Name 2:	Bread John	Phone 1: (999)999-9999 x	Type: Main
Title:	IT dude	Phone 2: (123)444-4444 x 4444	Type:
		Phone 3: () - x	Type:
Staff Name 3:		Phone 1:	Type:
Title:		Phone 2:	Type:
		Phone 3:	Type:

Figure 1 - Manipula dialog box collecting Contact information for NAMCS

Disadvantages with using the Manipula Dialog Box:

- The layout for the dialog boxes took time to develop. It took some trial and error before the look and feel of the dialog boxes appealed the user. This trial and error was the result of defining all the positions via pixels.
- Another disadvantage is that the data cannot be capture in the case Audit Trail file.

Challenges with implementing the Manipula Dialog Box:

Part of the data collected in the dialog box is the state abbreviation. Since the instruments are already using a state lookup, the dialog box had to interact with this external in the same manner. This created its own set of challenges. For instance, use of a text box to invoke the lookup table was not possible in the Manipula dialog box; the Authoring team implemented a button to trigger the lookup. After an FR selects the state, the information is displayed next to the button.

- Another challenge involved collecting the “type” of phone number entered (see figure 2). This field uses a drop-down box with up to ten options. The challenge here was that if an FR had entered an answer in the phone type field after realizing the phone number was entered erroneously, then the FR could not empty out the phone type field. The solution was to add code associated with the dialog

box that runs a procedure when the FR pressed the "Update" button. This code checks for data in the phone field and will empty out the phone type dropdown field if the phone field was empty.

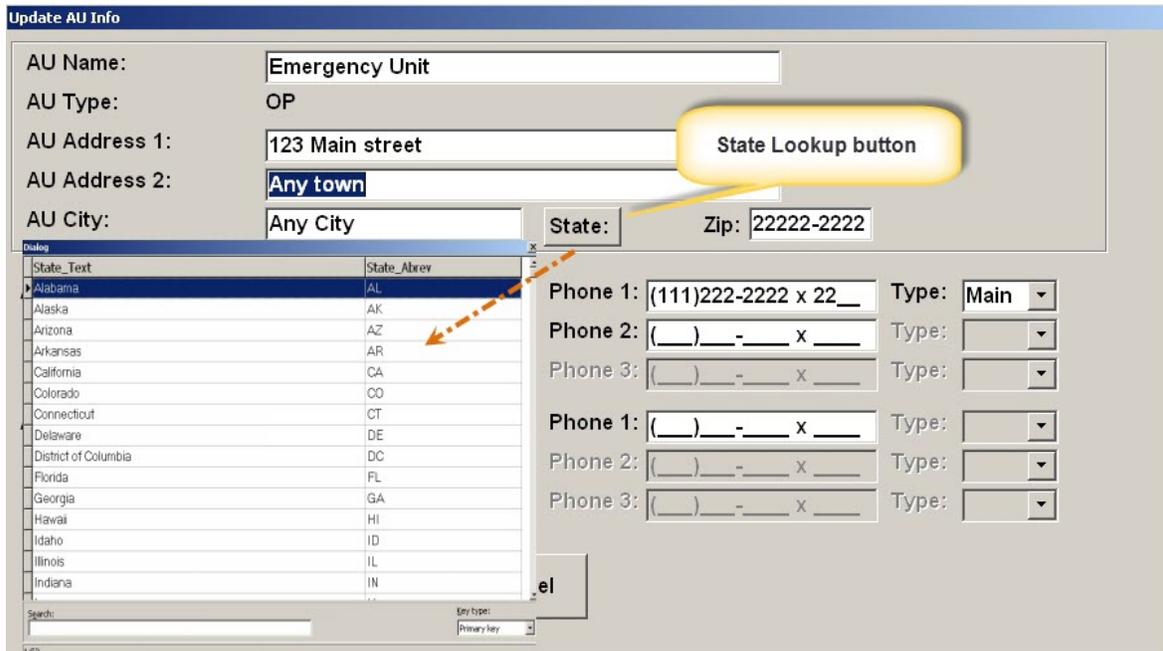


Figure 2 - Dialog box showing several types of entries

- Another requirement was the need to collect office schedules. While implementing this requirement, we found a situation similar to the one in the dropdown field for the phone types. However, this situation was observed in an enumerated answer list displayed as radio buttons (see figure 3). Note that while the schedule information can be entered into the Textbox, FRs can also select one of the radio buttons to quickly identify certain types of office scheduling (i.e., office open 24 hours, office not open, office has variable hours). Originally, there were only the three-labeled choices, and this caused a problem when FRs selected one of the radio buttons by mistake. There was no way for them to de-select the erroneous entry. Consequently, we added the first (unlabeled) column and made it the default setting. This provided the FRs with a way to remove an erroneous entry if necessary (see figure 3).

Day(s)	From	Time	To	Open 24 hours	Not open	Hours vary
Monday	8:00AM		5:00PM	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Tuesday	1:00PM		5:00PM	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Wednesday	1:00PM		5:00PM	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Thursday	8:00AM		5:00PM	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Friday	8:00AM		5:00PM	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Saturday				<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Sunday				<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Figure 3 - shows a dialog box with office schedule.

2.2 The customization of Blaise Mode Library layout to mimic the paper PRFs.

Except for the Patient Record Form (PRF) sections of the NAMCS and NHAMCS instruments, the screen layouts in the instruments were straightforward. That is, the screen consisted of the usual default Infopane with the question text, the answer list, and the Formpane where the FR enters the data. The PRFs, however, presented a challenge. The goal in implementing the PRF sections of the instrument was to mimic the paper form as much as possible. After all, the abstraction activity (i.e., collection of the patient record data) was a heads down data collection task and not an interactive interview. Mimicking the paper form would provide much of the same flexibility that the paper form provided in terms of navigation and moving around in the form. It would also reduce the FR's learning curve when migrating from the paper operation to the automated system.

In order to obtain the desired form layout for the PRFs, the Authoring team combined the GRID and INFOPANE producing an "electronic paper form" with columns and rows. The team then focused on customizing the FIELDPANE used throughout the PRF in the abstraction part of the instruments.

- The GRID specifies the columns and rows where the question items are displayed as well as the vertical or horizontal navigation. Since an FR may enter data in any order and not necessarily as displayed in the form, free navigation had to be set for all data items (see figure 4).

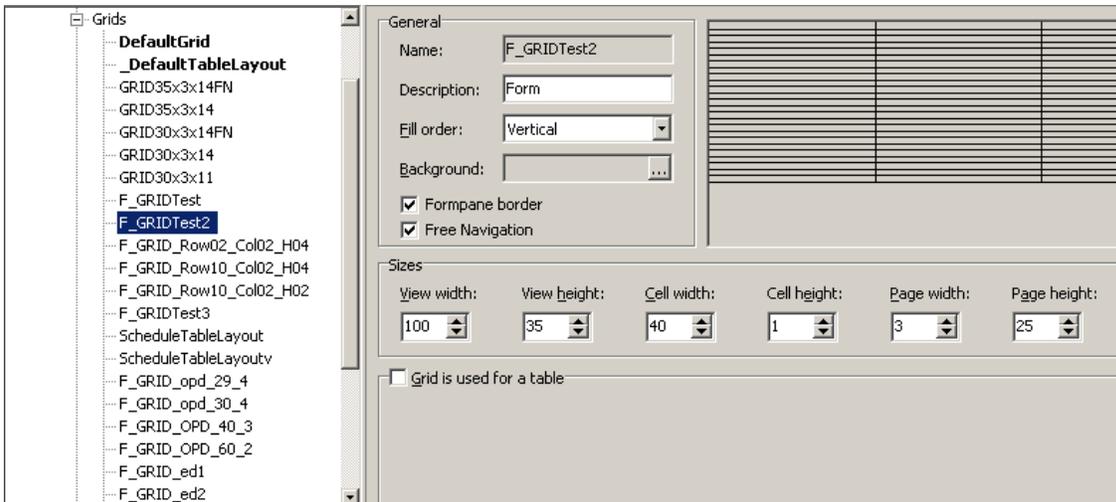


Figure 4 - Mode Library setting for the GRID showing 3 columns and 25 rows

- The INFOPANE acted as a placeholder. The settings for this object are shown in the figure below. Note that the INFOPANE is sized to '0', and the Field Text, Answer List and Answer Info are not set to Visible. This allowed the Authoring team to customize individual FIELDPANES for each question item (see figure 5).

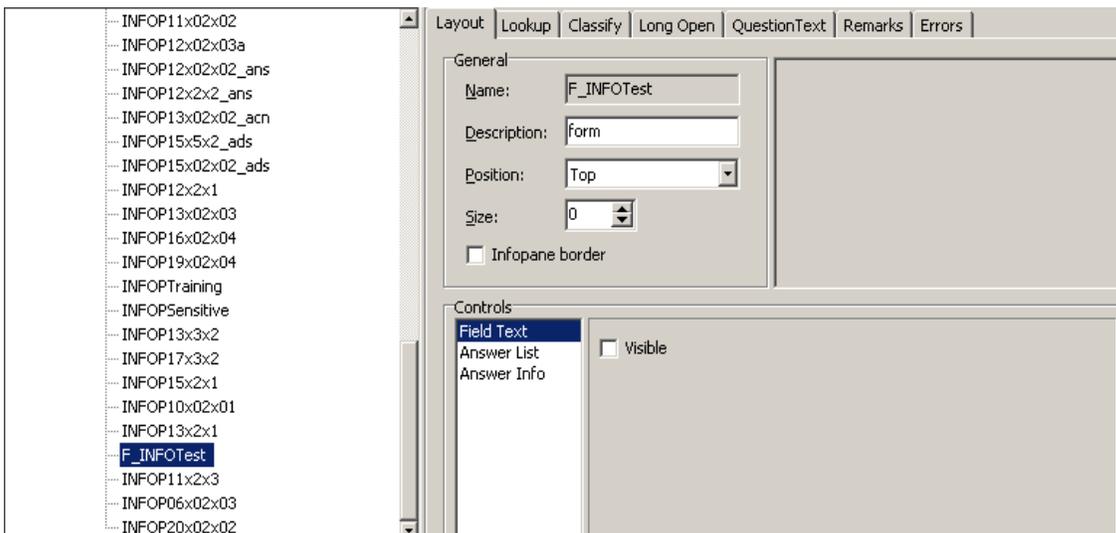


Figure 5 - Example of the INFOPANE settings used in the PRF forms

- The FIELDPANE is the element that was key to achieving the desired expected results. For the FIELDPANE, the Authoring team only used specific controls such as the Field Text, Input Line, Remark Icon, and Answer List (see figure 6). Throughout the PRF section the layout changes drastically. Since the PRF is a heads down data collection activity and the FR does not have to read question text from the PRF out loud, there is no need for extensive text description. Much of the time the question items in the PRF were just FR instructions on how or what to collect. FR instructions are displayed in Bright Blue text. This was one of the few U. S. Census Bureau screen standards the

authors adhered when implementing the PRF section. The authors had to put aside most of the other screen standards for the PRF because the standards primarily address interactive interviewing which did not apply to the PRFs.

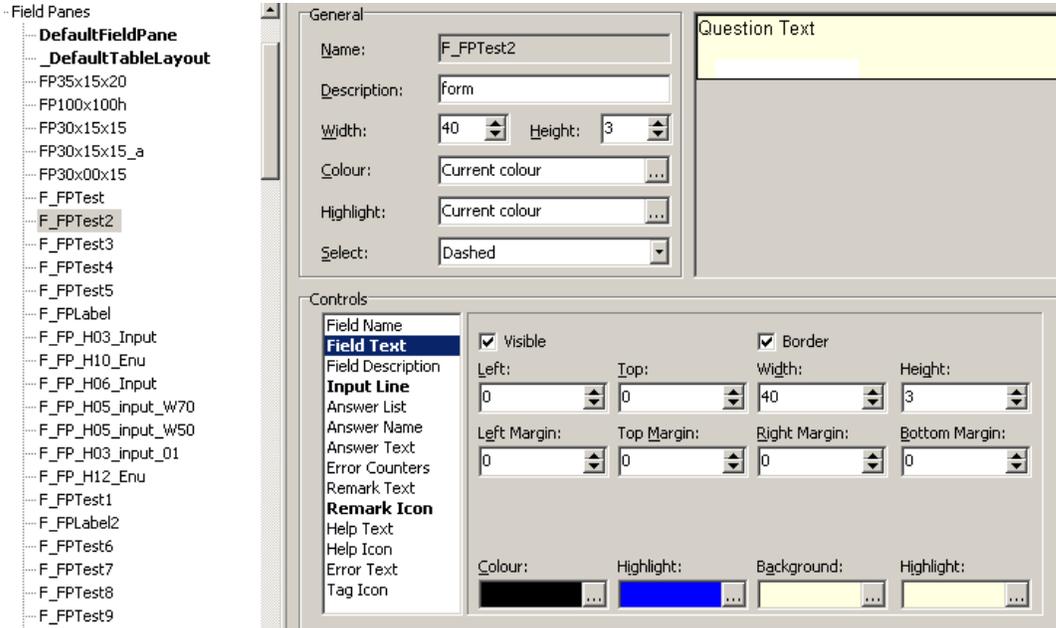


Figure 6 – Example of a more elaborate setting of the FIELDPANE setting used in the PRF

One very specific example of the appearance of a FIELDPANE used in the PRF is shown in Figure 7. The blue text represents an instruction for the FR to enter the medical record number for a patient’s visit. This question item used the default layouts for entering alphanumeric information.

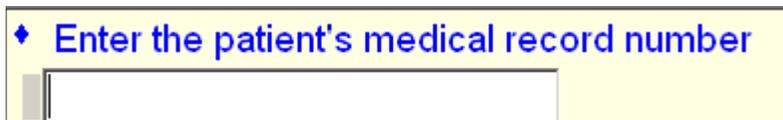


Figure 7 - shows FIELDPANE used for alphanumeric entry.

A more complete example of the PRF display is shown in Figure 8 on the next page. Here some of the question items have embedded in them the Answer List. The Authoring team could have disabled the input field for the enumerated type questions, but we did not do so. We discovered that the FR could select an entry type via the mouse and move to the next question successfully. However, there was no way for the FR to deselect an enumerated response entered by mistake. Furthermore, if we hid the input field for enumerated types, the FR would not be able to correct the entry causing issues in data processing. Since most of the questions in the PRF were not enforced by the rules, there would be an increased likelihood for getting erroneous data in the output processing.

The authors also considered using the auto-entry setting in the Mode library. Initial testing proved successfully. However, there was some resistance to the use of this feature. One of the concerns was that the FRs should be able to see the data as they enter a value and press the <Enter> key as means to validate the entry.

1 First 2 Prev 3 Next 4 Last 5 Add 6 Delete 7 Exit		NHAMCS FAQ Exit/F10 Patient Information Diagnosis Conditions Procedures Medications Anesthesia Disposition	
1 of 1 PRF's MRN: ABCDEFG NHAMCS-100(ASC) PATIENT INFORMATION			
Enter the patient's medical record number <input type="text" value="ABCDEFG"/>		Date/Time into operating room <input type="text" value="08/20/2013"/> <input type="text" value="2:00PM"/>	
Date of visit (Format MM/DD/YYYY) <input type="text" value="8/10/2013"/>		Date/Time surgery began <input type="text" value="08/20/2013"/> <input type="text" value="3:00PM"/>	
Patient's 5-digit zip code. (Enter "1" if homeless) <input type="text" value="22222"/>		Date/Time surgery ended <input type="text" value="08/20/2013"/> <input type="text" value="4:00PM"/>	
Date of birth <input type="text" value="1/3/1970"/>		Date/Time out of operating room <input type="text" value="08/20/2013"/> <input type="text" value="4:30PM"/>	
Age <input type="text" value="43"/>		Date/Time into postoperative care <input type="text" value="08/20/2013"/> <input type="text" value=""/>	
Enter time <input type="radio"/> 1. Years <input type="radio"/> 2. Months <input type="radio"/> 3. Days <input type="text" value="1"/> period		Date/Time out of postoperative care <input type="text" value="08/21/2013"/> <input type="text" value=""/>	
Sex <input type="radio"/> 1. Female <input checked="" type="radio"/> 2. Male		Expected source(s) of payment for THIS VISIT. <input checked="" type="checkbox"/> 1. Private Insurance <input checked="" type="checkbox"/> 2. Medicare <input type="checkbox"/> 3. Medicaid or CHIP or other state-based program <input type="checkbox"/> 4. Workers' compensation <input type="checkbox"/> 5. Self-pay <input type="checkbox"/> 6. No charge /Charity <input type="checkbox"/> 7. Other <input type="checkbox"/> 8. Unknown	
Ethnicity <input type="radio"/> 1. Hispanic or Latino <input checked="" type="radio"/> 2. Not Hispanic or Latino <input type="text" value="2"/>		<input type="text" value="1,2,3"/>	

Figure 8 – Example of a more elaborate use of the FIELDPANES used in the NHAMCS instrument

Another example of a PRF display created using the layout from the Mode Library appears in Figure 9. This example is from the PRF for an Emergency Department.

1 First 2 Prev 3 Next 4 Last 5 Add 6 Delete 7 Exit		NHAMCS FAQ Exit/F10 Patient Information Triage Reason Injury Diagnosis Diagnostics Meds Disposition	
1 of 1 PRF's MRN: ABCDEFG NHAMCS-100(ED) PATIENT INFORMATION			
Enter the patient's medical record number <input type="text" value="ABCDEFG"/>		Date (MM/DD/YYYY) Time (HH:MM AM/PM/ML) (1) Date of Arrival <input type="text" value="8/20/2013"/> <input type="text" value="6:00AM"/> (2) Seen by MD/DO/PANP ? [F1] <input type="text" value="8/20/2013"/> <input type="text" value="8:00AM"/> (3) Date of ED departure, if released or transferred ? [F1] <input type="text" value="8/21/2013"/> <input type="text" value="2:00PM"/>	
Patient Residence <input type="radio"/> 1. Private residence <input checked="" type="radio"/> 2. Nursing home <input type="radio"/> 3. Homeless <input type="radio"/> 4. Other <input type="radio"/> 5. Unknown <input type="text" value="2"/>		Patient's 5-digit zip code. (Enter "1" if homeless) <input type="text" value="22222"/>	
Date of birth <input type="text" value="3/15/1970"/>		Date of birth <input type="text" value="3/15/1970"/>	
Age <input type="text" value="43"/>		Enter time period <input type="radio"/> 1. Years <input type="radio"/> 2. Months <input type="radio"/> 3. Days <input type="text" value="1"/>	
Sex <input type="radio"/> 1. Female <input checked="" type="radio"/> 2. Male		Ethnicity <input type="radio"/> 1. Hispanic or Latino <input checked="" type="radio"/> 2. Not Hispanic or Latino <input type="text" value="2"/>	
Race <input type="text" value="1,3,2"/>		Race <input checked="" type="checkbox"/> 1. White <input type="checkbox"/> 2. Black or African American <input checked="" type="checkbox"/> 3. Asian <input type="checkbox"/> 4. Native Hawaiian or Other Pacific Islander <input type="checkbox"/> 5. American Indian or Alaska Native	
Arrival by ambulance <input checked="" type="radio"/> 1. Yes <input type="radio"/> 2. No <input type="radio"/> 3. Unknown <input type="text" value="1"/>		Expected source(s) of payment for THIS VISIT. <input checked="" type="checkbox"/> 1. Private Insurance <input type="checkbox"/> 2. Medicare <input type="checkbox"/> 3. Medicaid or CHIP or other state-based program <input type="checkbox"/> 4. Workers' compensation <input type="checkbox"/> 5. Self-pay /Charity <input type="checkbox"/> 6. No charge <input type="checkbox"/> 7. Other <input type="checkbox"/> 8. Unknown <input type="text" value="1"/>	

Figure 9 – Example of a Form for the NAMCS instrument

2.3 Navigation of the Abstraction Instrument

The NAMCS and NHAMCS projects had requirements for navigation within and between the PRFs and having the forms look and feel the same across the two surveys. As a result, the authors had to address several navigation issues relating to Parallel Tabs, free form navigation in tables, buttons, and shortcuts. The authors defined Tabs to allow for data collection of more than one hundred PRFs. The Tabs were defined using temporary Auxfields. The authors then passed data into the Auxfields from the Blaise database array for editing. The authors could not call all the PRFs from the main data model level because that would place several PRFs on route at the same time. Moving from one PRF to another would have been slow and cumbersome. We needed to have one PRF at a time and we needed to be able to move from one to another for rapid data collection.

2.3.1 Parallel tabs navigation

Navigating with parallel Tabs has some interesting considerations. The instrument needed to allow movement between fields on the screen and provide movement to the next logical group of questions within the PRF (i.e., the next parallel block). The user could click on a Tab to activate those questions or use the Ctrl-Tab key combination to move from Tab to Tab in a forward direction or use Ctrl-Shift-Tab to move in a backward direction. The authors used the paper forms of the questionnaire to group questions that were similar for each Tab. Some Tabs have one screen page and others have multiple screen pages. The authors showed the number of potential pages on the status bar at the bottom of the instrument. When you are on a Tab only the pages that are valid with that Tab are counted.

To implement the parallel Tabs we needed to segment the code into blocks. We did this by grouping several questions together into blocks and defined each block as a separate Tab. In figure 10 note the dot notation for each Tab defined.

```
PARALLEL
mASLs = ASLs
FAQs = FAQ
Appointment = F10
ASC_P1 = PV_PRF.tempASC_PRF.Page1
ASC_P2 = PV_PRF.tempASC_PRF.Page2
ASC_P2b = PV_PRF.tempASC_PRF.Page2b
ASC_P3 = PV_PRF.tempASC_PRF.Page3
ASC_P4 = PV_PRF.tempASC_PRF.Page4
ASC_P5 = PV_PRF.tempASC_PRF.Page5
ASC_P6 = PV_PRF.tempASC_PRF.Page6

OPD_P1 = PV_PRF.tempOPD_PRF.Page1
OPD_P5 = PV_PRF.tempOPD_PRF.Page5
OPD_P2 = PV_PRF.tempOPD_PRF.Page2.Reason_for_Visit
OPD_P2Injury = PV_PRF.tempOPD_PRF.Page2.Injury
OPD_P3 = PV_PRF.tempOPD_PRF.Page3
OPD_P4 = PV_PRF.tempOPD_PRF.Page4
OPD_P6 = PV_PRF.tempOPD_PRF.Page6
OPD_P9 = PV_PRF.tempOPD_PRF.Page9
OPD_P10 = PV_PRF.tempOPD_PRF.Page10
OPD_P11 = PV_PRF.tempOPD_PRF.Page11
```

Figure 10 - Parallel Tab definitions

Each block defined as Page1 through Page n... was designated as a group of logical questions. We used Page1 as the basic patient information and PRF identification. An example of Page1 can be seen in Figure 11.

The screenshot displays the 'National Ambulatory Medical Care Survey (NAMCS) - Ver 3.02 7/08/2013' application window. The interface is organized into a grid of input fields and sections. At the top, a red header bar contains the text '1 of 1 PRF's MRN: NAMCS PATIENT INFORMATION'. Below this, the form is divided into several columns and rows of data entry areas:

- Left Column:** Contains fields for 'Enter Office Number' (with a dropdown menu showing '1. 104 Ocean View Circle'), 'Enter the patient's medical record number', 'Date of visit (Format MM/DD/YYYY)', 'Patient's 5-digit zip code. (Enter "1" if homeless)', and 'Date of birth (Format MM/DD/YYYY)'.
- Middle Column:** Contains fields for 'Age', 'Enter time period' (with radio buttons for '1. Years' and '3. Days', and a dropdown for '2. Months'), 'Sex' (with radio buttons for '1. Female' and '2. Male'), 'Is patient pregnant?' (with radio buttons for '1. Yes' and '2. No'), 'Specify Gestation - Gestation week refers to the number of weeks plus 2 that the offspring has spent developing in the uterus', and 'Last menstrual period - Month/Day/Year'.
- Right Column:** Contains sections for 'Race' (with checkboxes for '1. White', '2. Black or African American', '3. Asian', '4. Native Hawaiian or Other Pacific Islander', and '5. American Indian or Alaska Native'), 'Expected source(s) of payment for THIS VISIT.' (with checkboxes for '1. Private Insurance', '2. Medicare', '3. Medicaid or CHIP or other state-based program', '4. Workers' compensation', '5. Self-pay', '6. No charge /Charity', '7. Other', and '8. Unknown'), and 'Tobacco Use' (with radio buttons for '1. Not current, never', '2. Not current, former', '3. Current', and '4. Unknown').

The interface also features a top menu bar with options like 'Forms', 'Answer', 'Navigate', 'Options', and 'Help', and a toolbar with navigation buttons such as '1 First', '2 Prev', '3 Next', '4 Last', '5 Add', '6 Delete', and 'Z Exit'.

Figure 11 – Example of the ASC patient information and PRF identification

To allow the FR to quickly identify where in the abstraction process they were, the authors used an Auxfield to show some text with important information. As noted in the Blaise help, an important use of the Auxfield is to place a label in the page of the interviewing screen. This kind of label can be used to announce a section and to give the interviewer a landmark that can be used for page-based navigation.

Navigating from one Tab to the next was accomplished by using the Action from the "At end" property. Each Tab allows for an event, an action that defines what to do at the end of the parallel Tab. The authors used that event to activate the next Tab. Moving from the first Tab to the next both activates the specific Tab and places the focus on the first field. On the last active question of the last parallel Tab, the Manipula event evaluates the maximum number of rows allowed using a Manipula procedure. Depending on the number of active PRFs, it either goes to the first PRF or adds a new PRF and then sets the focus on the first active question for the active Tab. The "At end:" property with the Action option is used to provide a method to move to the next parallel block and land on the first question of the next screen. In figure 12 we show how to select a parallel Tab and provide a Manipula procedure to activate at the end of all questions on the Tab. Navigating from one Tab to another is done sequentially while moving from field question to the next field question. If the navigation needs to skip field questions then the user can click on the Tab with the specific data required. This way data from the office folders can be collected in any order and skipping around the PRF is quick and convenient.

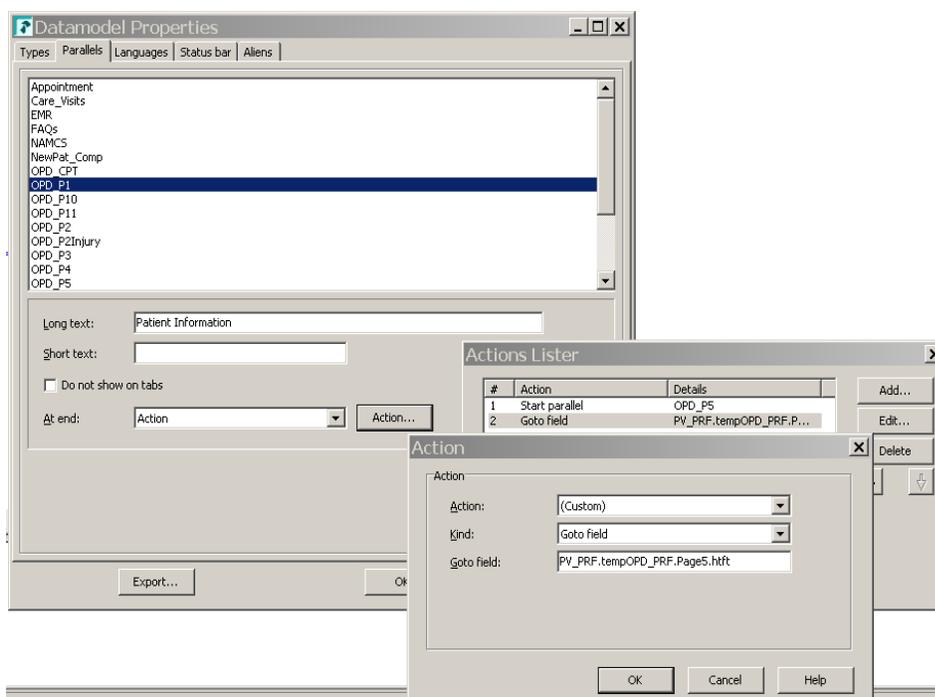


Figure 12 – Set the parallel Tab action with Manipula procedure

The organization of the Tab pages are as follows.

- Patient Information – Visit date, Birth date, zip code, age, Race, etc.
- Vital Signs – Height, Weight, temperature, blood pressure
- Reason for visit – 5 user input and 5 lookup reasons plus the major reason question
- Injury – the injury and/or cause of the patient visit
- Continuity of care – what happens after the patient leaves the facility
- Diagnosis – 5 user input and 5 lookup diagnosis plus additional general diagnosis
- Services – exams, tests ordered, treatments
- Medications – user input and lookup medications list
- Disposition – final patient outcome after visit
- Tests – Cholesterol, HDL, LDL, TGs, HbA1c, FBG, and Serum creatinine
- CPT – current procedural terminology lookup

2.3.2 Free form navigation

Another requirement for the PRF section in the abstraction instrument is to provide the FR with the ability to move around freely to record patient data. This is necessary during the abstraction process because the information in a physician's and a clinic's records does not necessarily appear in the order in which the instrument collects it. As a result, an FR might encounter a data item in a patient's medical record and then have to navigate to the correct field in order to enter that bit of information.

Free form navigation was implemented with the use of the arrow keys and the page up and page down keys. The instrument allows movement from one field to another without requiring the FR to enter data. Questions can be left blank and then answered later in the abstraction process. The instrument performs data validation at the end of each group of questions (i.e. parallel block).

This type of navigation allows movement of field questions on a page and movement from one Tab to another. Jumping around is allowed to facilitate quick data entry for the PRFs. When moving from the first Tab to another and then back to the first Tab again it will allow the active focus to remain on the field question that was left off on.

For a table, the authors specified the direction of the ENTER / TAB key when free navigation was active. To accomplish this, we used the property 'Fill order' for the grid in the Mode Library. Note the grid in figure 13 shown below uses the horizontal fill to allow for data collection across the screen.

The screenshot shows the 'Tests' section of the NAMCS software. At the top, a red header bar displays '6 of 6 PRF's', 'MRN: 6', and 'NAMCS Tests'. Below this, a question asks: 'Was blood for the following laboratory tests drawn on the day of the sampled visit or during the 12 months prior to the visit?'. A reference time range of '4/24/2010 - 4/24/2011' is provided. A radio button option '1. Enter 1 to Continue' is visible. The main data collection area is a table with three columns: 'Most recent result' and 'Date of Test'. The table contains seven rows of laboratory tests, each with a question, a 'yes/no' response field, a result field, and a date dropdown menu.

	Most recent result	Date of Test
? [F1] Was blood for the following laboratory tests drawn on the day of the sampled visit or during the 12 months prior to the visit? Reference Time: 4/24/2010 - 4/24/2011 <input type="checkbox"/> 1. Enter 1 to Continue		
? [F1] Total cholesterol? (1 = yes 2 = none found)	Total cholesterol mg/dL	
? [F1] High density lipoprotein (HDL)? (1 = yes 2 = none found)	HDL mg/dL	
? [F1] Low density lipoprotein (LDL)? (1 = yes 2 = none found)	LDL mg/dL	
? [F1] Triglycerides (TGs) ? (1 = yes 2 = none found)	TGs mg/dL	
? [F1] HbA1c Glycohemoglobin ? (1 = yes 2 = none found)	A1C %	
? [F1] Fasting blood glucose (FBG) ? (1 = yes 2 = none found)	Blood glucose mg/dL	
? [F1] Serum creatinine? (1 = yes 2 = none found)	Serum creatinine mg/dL	

Figure 13 - Data collection using horizontal grid

2.3.3 Button navigation

As noted earlier, an FR can collect over one hundred separate PRFs within an abstraction instrument. As a result, it was necessary to provide the FRs with a way to move between the PRFs.

To implement this type of navigation, the authors used Blaise menus to create a button panel. The button panel is only active and visible when in the PRF section of the instrument. We assigned the "Alt-1" through "Alt-7" function key combinations to the buttons in the panel. This satisfied the requirement that all instrument functionality is keyboard accessible. Buttons not needed in a particular context are deactivated and greyed out, but they remain visible. For example, if you are on the first PRF then the "previous" button would be greyed out. If you were on the last PRF then the "next" button would be greyed out. The "exit" button was implemented to finalize the PRF data collection and complete the case.

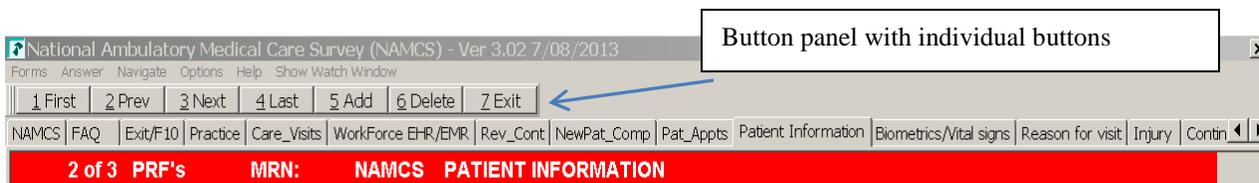


Figure 14 - Button bar for PRF navigation

The button panel used in the PRF section is shown in figure 14 above. The functions for the seven buttons in the panel are specified in the following list.

- First – moves to the first PRF unless on the first PRF
- Previous – moves to the previous PRF unless on the first PRF
- Next – moves to the next PRF unless on the last PRF
- Last – moves to the last PRF unless on the last PRF
- Add – adds an additional PRF unless maximum has been reached (set to 150 PRF's at this time)
- Delete –deletes a PRF and moves all PRF's up unless on the last PRF
- Exit – allows an FR to exit the PRF section

For the buttons to be active, some logical criteria had to be set to "True" in the "Enabled" criteria box (see figure 15). Using the panel design, the authors added the buttons and associated Manipula procedures with the buttons to implement the necessary movement.

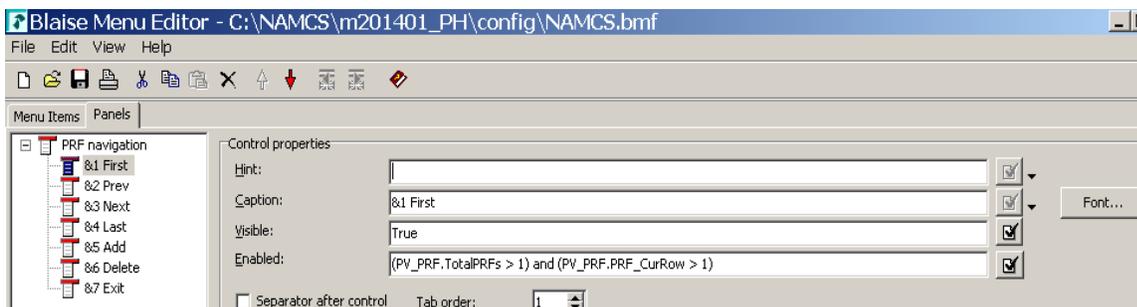


Figure 15 - Button definition

In figure 16 note how the "/Q" option is used to launch the Manipula script in the quiet mode. This ensures that there was no screen blinking when navigating between PRFs. In addition, the authors also found there was no time lag when the procedure was called. The instrument moved quickly among the PRFs.

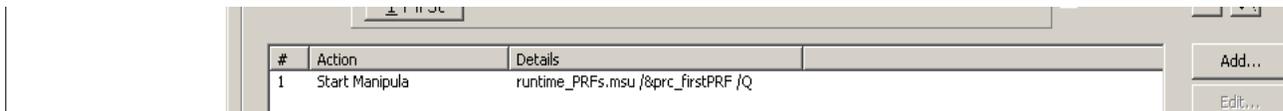


Figure 16 - Button action using Manipula procedure

2.3.4 Navigation shortcuts

Using the Blaise menu options there are several Manipula boxes that allow the user to enter data about the survey. Contact information, Office schedule, Interview status, Web setup information, and PRF status. Using the PRF status Manipula box we can view the PRFs and get details about a specific PRF. Because there could be over one hundred PRF's we wanted to be able to move to a specific PRF with confidence the correct one was selected. Each PRF number is defined by array row. Each PRF status is displayed by parallel block as a group of questions. Using this method the user can review a PRF plus pick a PRF to be edited. See figure 17.

PRF #	Medical Record Number	Visit Date	Patient Info	Reason for Visit	Care	Diagnosis	Vital Signs	Diagnostic	Meds	Disposition
1	1	4-20-2011	Yes	No	No	No	No	No	Yes	No
2	2	4-21-2011	No	No	No	No	No	No	No	No
3	3	4-22-2011	No	No	No	No	No	No	No	No
4	4	4-22-2011	No	No	No	No	No	No	No	No
5	5	4-23-2011	No	No	No	No	No	No	No	No
6	6	4-24-2011	No	No	No	No	No	No	No	No

Select PRF
Details
Close

Figure 17 - PRF selection and section status

When the Details button is activated, the Page layout for the Manipula box provides status on each of the parallel blocks as a group of questions. Note the use of color where red shows critical data missing. See figure 18.

Show PRFs Status for PRF's 1
Total PRFs = 2

PRF #	Medical Re	Patient Information	Vital Signs	Meds	Disposition
1	1	Patient Information Patient Zip: 1 Birthdate/Age: 01/01/1970 / 41 Years Sex: Female Pregnancy: No Ethnic: Not_Hisp Race: Black Payment Source: Medicare Tobacco Use: Current	Vital Signs Height: 5 ft 5 in Weight: 120 lbs 0 ozs Temperature: 100 Fahrenheit Blood Pressure: 1 / 1	Meds Medications:	Yes
2	2	Reason for Visit Injury: Yes, injury/trauma Reason for Visit: 2	Diagnostic Screening Services: NO SERVICES	Disposition Providers Seen: Physician Duration: 1 minutes Visit Disposition: Critical data missing	No
		Continuity of Care Prime Care: Yes Referred: Seen Before: Yes, established patient Past Visits: 1 Major reason for visit: New problem (Lookback Tests Cholesterol Test: HDL Test: LDL Test: TGS Test: A1c Test: FBG:		No

Select PRF
Previous
Next
Close

Figure 18 - PRF details Manipula box with buttons

Using the Manipula boxes Contact information, Office schedule, and Interview status from anywhere in the instrument allows the user to view case status and case level data. The contact information can be updated. The information is provided for both names and phone numbers. The Office schedule is provided as a tool for data collection of office times and places. The interview status is used to identify

when a case needs to be completed and where any data elements are missing. This can be very helpful when jumping around in the instrument, something can be skipped and the user would need to be reminded where to data needs to be completed.

3. Conclusions

The Authoring team is very familiar with Manipula and Maniplus. Nevertheless, the extensive use of Manipula and Maniplus in the NAMCS and NHAMCS instruments turned out to be a challenge and provided us with the opportunity to expand on their capabilities on a large scale. In the end, Manipula/Maniplus proved to be robust and reliable tools for implementing many of the requirements for the two medical surveys. Manipula was a great aid. Using it allowed the team to quickly resolve a number of challenges using alternative and innovative methods.

Working with the Blaise Mode Library was challenging in the beginning. Once the pattern was established however, the Authoring team did not have to invest a great deal of time in creating Fieldpanes for each item because the Fieldpanes were used for multiple questions. In the end, we found great flexibility in the layout which enabled us to mimic the paper forms successfully.

This approach to the layout provided some feedback on the use of the auto-enter option in the Mode Library. While this setting was not used for the surveys because it is a global setting, it might have been useful if it could have been implemented for the Patient Record Forms only.

Another realization while working on this project was that the layout of the PRF's could easily be used as a Web Form.

4. Acknowledgements

Many individuals have worked and continue to work on the NAMCS and NHAMCS to ensure their successful operation. We would like to include grateful acknowledgement to our colleagues from the U.S. Census Bureau whose work directly contributed to the instrument development discussed in this paper. We thank Johanna Rupp and Michael Johnson for their work and efforts in the design, programming and testing of the instruments for this project.

5. Disclaimer

The views expressed on (statistical, methodological, technical, or operational) issues are those of the author(s) and not necessarily those of the U. S. Census Bureau.

DEP Unchained: Using Manipula on the Current Record in Memory

Peter Stegehuis and Rick Dulaney, Westat, Inc

INTRODUCTION

One longstanding feature of Blaise is the very tight integration between code, presentation and data storage: what you code is what you see, and what you see is what's stored in the database. The descriptive nature of Blaise datamodels also provides a framework for the Blaise selective checking mechanism, which is generally regarded as one of the strengths of the Blaise system.

However, these advantages come with two significant potential pitfalls. First is that Blaise datamodels require fixed arrays. This requirement is fine to a point, and accommodates many surveys, but can be problematic when arrays are nested or combined. Take the simple example in Listing 1, in which a list of persons are each asked about a list of cars:

Listing 1. Collecting Data about Persons and their Cars

```
DATAMODEL CarPeople

BLOCK BCars
  FIELDS
    MakeModel "What's the make and the model of the car?": STRING[20], EMPTY
    LikeIt "Did you like this car": (Yes, No, SoSo)
  RULES
    MakeModel
    IF MakeModel <> EMPTY THEN
      LikeIt
    ENDIF
ENDBLOCK {BCars}

BLOCK BPerson
  LOCALS
    J: INTEGER

  FIELDS
    FirstName "First Name": STRING[20], EMPTY
    LastName "Last Name" : STRING[30]
    Cars: ARRAY[1..500] OF BCars

  RULES
    FirstName
    IF FirstName <> EMPTY THEN
      LastName
      FOR J:= 1 TO 500 DO
        IF J=1 OR Cars[J-1].MakeModel <> EMPTY THEN
          Cars[J]
        ENDIF
      ENDDO
    ENDIF
ENDBLOCK {BPerson}

LOCALS
  I: INTEGER

FIELDS
  Person: ARRAY[1..500] OF BPerson

RULES
  FOR I:=1 TO 500 DO
    IF I = 1 OR Person[I-1].FirstName <> EMPTY THEN
      Person[I]
    ENDIF
  ENDDO
ENDMODEL {CarPeople}
```

When prepared, this datamodel will yield an error (“The maximum number of allowed pages has been reached”), and the normal course of action is to compromise on the size of the arrays until it prepares successfully. This is just one example, created to trigger this error, but we regularly run into similar

problems in large establishment surveys with hundreds of observations nested at multiple levels, and even in complex household studies, particularly those that track populations and rostered information over time.

The second potential pitfall is the datamodel-wide scope of the selective checking mechanism itself, which checks the entire datamodel at each opportunity, including checks and downstream computations that may not be intended. When the interviewer leaves the first field in a ten-item datamodel, Blaise checks and potentially performs computations after the tenth item as well. This leads to awkward IF...THEN blocks to “protect” code from running before the programmer intended it, and may lead to dangerous uses of the powerful KEEP statement.

NEW CAPABILITIES

Blaise 4.8 introduced a powerful new capability: the Blaise DEP may shell out to other programs, which can perform additional computations, and even write directly to the record in memory from outside the DEP. Because Manipula has so many broad uses within the Blaise system, we focus in this paper on the use of external Manipula procedures running outside the DEP. However, any executable recognized on your system will run in the same fashion.

External programs may be associated with several different actions in the DEP, such as a button press or a category selection. Buttons on a panel may become visible or active based on conditional logic using the interview data. Figure 1 shows one simple method: open Project->Datamodel Properties, select the type, and click the Event button to specify the program and procedure name – in this case, we will run the Manipula setup CarPeople and start at the procedure named StartCarsProc.

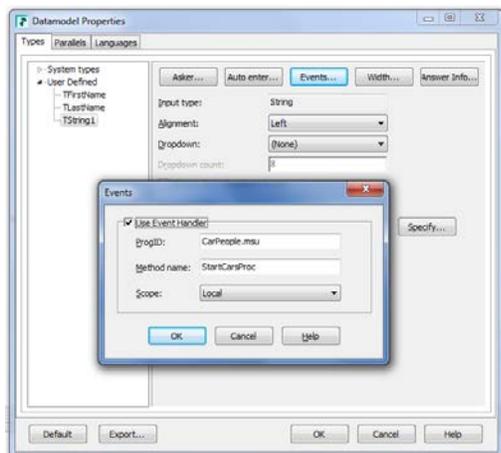


Figure 1. Associating an external program with a field type.

DISTRIBUTED DATAMODELS

One of the immediate impacts of this new capability is the ability to accommodate large datamodels, in a way that resembles traditional relational database operations. Returning to the example above, the natural way to model this data is to have one table for persons having a one-to-many relationship with a table storing information about each car that a person owns. We can now collect store these data using two datamodels and use Manipula to move between them. The People datamodel has a table with one row per person:

Listing 2: People datamodel

```
DATAMODEL People
  PRIMARY
    CaseNumber

  TYPE
    TFirstName = STRING[20]
    TLastName  = STRING[30]
    TString1   = STRING[1], EMPTY

  FIELDS
    CaseNumber: 1..1000
    CurrentPersonID: 1..500

  TABLE BallPeople
    LOCALS
      I: INTEGER

    BLOCK BPerson
      FIELDS
        PersonID: 1..500
        FirstName "First Name": TFirstName, EMPTY
        LastName  "Last Name" : TLastName
        AddCars   "Press Enter to add cars for ^FirstName ^LastName ": TString1, EMPTY

      RULES
        PersonID:= I
        PersonID.SHOW
        FirstName
        IF FirstName <> EMPTY THEN
          LastName
          AddCars
        ENDIF
      ENDBLOCK {BPerson}

    FIELDS
      Person: ARRAY[1..500] OF BPerson

    RULES
      FOR I:=1 TO 500 DO
        IF I = 1 OR Person[I-1].FirstName <> EMPTY THEN
          CurrentPersonID:= I
          Person[I]
        ENDIF
      ENDDO
    ENDTABLE {BallPeople}

  FIELDS
    AllPeople: BALLPeople
```

The AddCars column in the person table uses the TString1 format above to call Manipula (recall that the method name is StartCarsProc):

Listing 3. CarPeople Manipulus Setup.

```
PROCESS CarPeople

USES
  People
  Cars

UPDATEFILE
  CarsData: Cars ('Cars', BLAISE)

TEMPORARYFILE PeopleInMemory: People
  SETTINGS
```

```

INTERCHANGE = SHARED
CONNECT = NO
CHECKRULES = YES

AUXFIELDS
  Rslt: INTEGER
  AuxStrPeopleCaseID, AuxStrCurrPersID: STRING[4]

PROCEDURE StartCarsProc
  IF (ROUTERSTATUS= BLRSPPOSTEDIT) THEN
    AuxStrPeopleCaseID:= STR(PeopleInMemory.CaseNumber)
    AuxStrCurrPersID:= STR(PeopleInMemory.CurrentPersonID-1)
    Rslt:= CarsData.EDIT('/G /K'+ AuxStrPeopleCaseID + ';' + AuxStrCurrPersID + ' /X')
  ENDIF
ENDPROCEDURE {StartCarsProc}

```

Note that the current record in memory is designated `PeopleInMemory` and uses the `INTERCHANGE = SHARED` keyword to enable writing from Manipula and subsequent datamodels. The one-to-many relationship between `People` and `Cars` is implemented by constructing the `Cars` key using the case ID from `People` plus the ID of the current person and calling the `Cars` datamodel to collect make/model and consumer attitude:

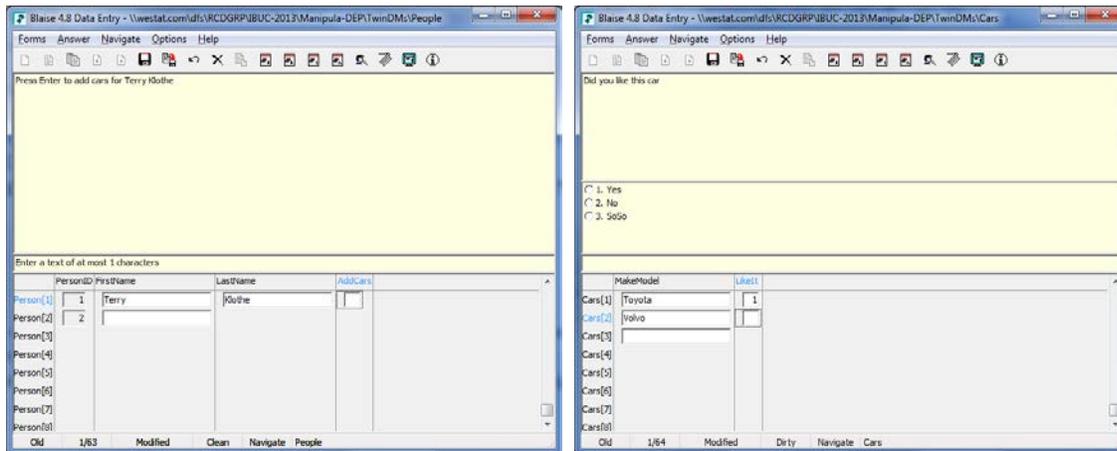


Figure 2. Calling the `Cars` datamodel while the `People` datamodel is active.

Note in the title bar on the left that the `People` datamodel is running to collect a table of persons. Pressing enter in the `AddCars` column goes through Manipula and presents the `Cars` datamodel on the right. Pressing Enter in on a blank line will return to the `People` datamodel to collect the next person and his or her `Cars`. For this example, we preloaded records in the databases, but this is normally implemented using one or more foreign keys. Many-to-many relationships may use a separate datamodel to store the linkages and other attributes of the relationship. There is no logical limit to the number of datamodels or Manipula setups that may be invoked.

The ability to extend Blaise capabilities at run-time offers significant advantages. Of course, once Manipula has started, one can show dialogs, open additional datamodels, or perform computations involving any number of databases – including the one currently opened by the original datamodel. Of particular importance is that Manipula code called in this way can write directly to the DEP record in memory. This has significant impact for large or complex surveys that threaten to exceed the storage capabilities of a single datamodel, or have complex navigation requirements. Some examples of activities that may run while the DEP is active include:

- Sampling units for subsequent interviewing in the DEP
- Various lookup activities, with outcomes stored in the current record in memory
- Roster management for lists currently in use in the DEP, including add-delete-update operations

NAVIGATION

Navigation generally poses challenges in large instruments. To facilitate non-linear navigation, such as moving back to a specific field, developers often resort to techniques such as emptying a field on or near the destination. The next time the selective checking mechanism fires, the interviewer will be taken back to that now-empty field. This solution requires that the field not have the EMPTY attribute and that the programmer be very careful to avoid unintended navigation. In Blaise 4.8 developers may use the SETACTIVEFIELD keyword to send the interviewer to any location in the current record, regardless of whether the field is required or whether there is indeed data in the field.

This capability is extraordinarily powerful when combined with the ability to distribute data among different datamodels. In the simple example above, at the end of adding cars for one person, you automatically return to the People datamodel, because that's where you left off. In a more elaborate example you could more actively determine where to 'put the cursor' by using the keyword SETACTIVEFIELD to return control to the original datamodel and continue data collection using the DEP. ACTIVEFIELD is the corresponding property, so programmers can save the active field at any point and return to it later on. The related keywords ACTIVEPARALLEL and SETACTIVEPARALLEL are also very useful; one can interrupt the current processing using a parallel block, shell out to Manipula to perform other activities, and then return to the parallel block or to the original datamodel.

CASE STUDY

There are many potential uses for these capabilities. We show here a specific example of a last-minute requirement that would have been difficult or impossible to implement within a single datamodel.

The National Hospital Care Study is a large and complex study, one portion of which requires data collectors to abstract medical records from hospitals. The data collection begins with a hospital level induction instrument, which collects general information and enumerates ambulatory units (AUs). The system then creates a case for each AU and the data collector opens the AU instrument. The AU instrument samples patient visits to that AU during a reference period and collects data onto a Patient Record Form for that visit; the PRF has many individual parts, as shown here:

Figure 3. The main Patient Record Form screen in the AU instrument.

In Figure 3, a single PRF is collected using multiple tabs. Data collectors may use buttons 1-7 on the panel to move through collected PRFs as well as to add, delete or edit a PRF.

In order to increase the number of cases meeting a specific condition, late in the development cycle the client provided sample specifications that allowed us to create an external file of hospital visit data as reported on a specific claims form called the UB04. Unfortunately, the external UB04 data was at the hospital level, not the AU level. We added a button – number 8, above – to enable interviewers to examine the external file, sort it as necessary, add comments or a disposition if appropriate, and select a UB04 which is then written back to the current PRF record in memory.

Figure 4 shows the screen after pressing the “8 – UB04 List” button. We have called a Manipula setup which reads in the external data and presents it to the data collector. Notice that the first record displays a disposition (“Incomp”), and the data collector is entering a comment on the second record. When the data collector then pressed the “Move Case to PRF” button on the third record, the data were written to the PRF record as shown in Figure 5.

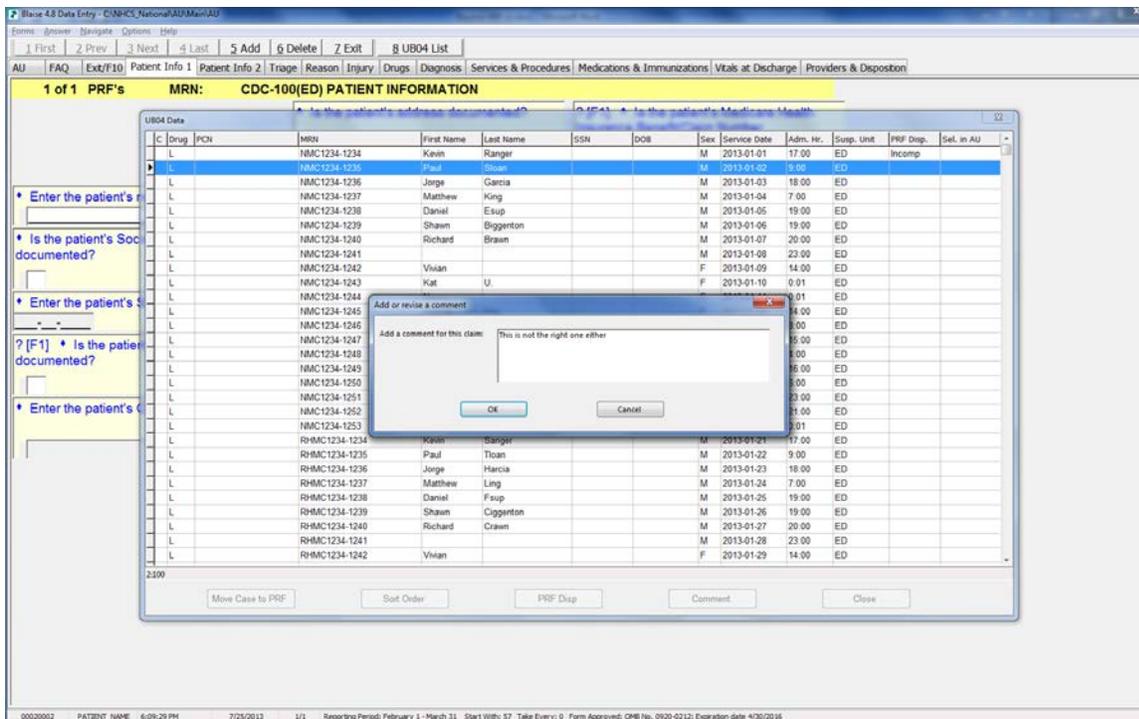


Figure 4. Updating disposition and adding comments in the external file.



Figure 5. Data from external file written back to DEP record in memory.

In this case, when we copy UB04 data to the Blaise record in memory, we also indicate this on the external file, as shown by the Blaise case number in the rightmost column in Figure 6.

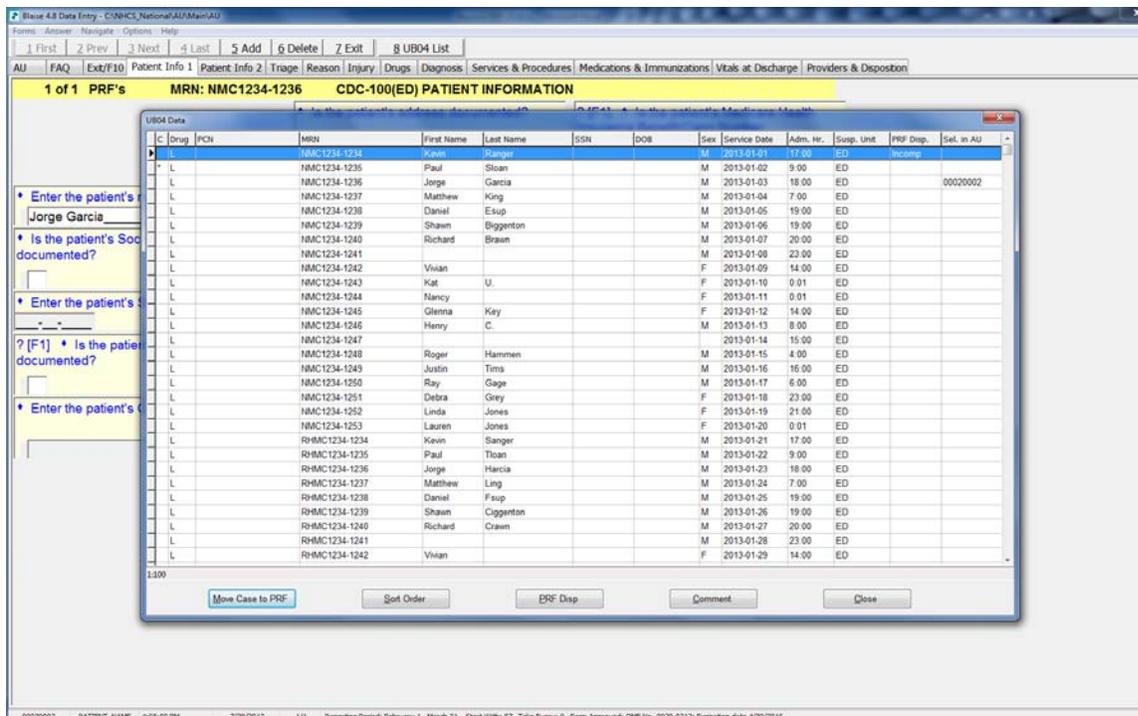


Figure 6. Storing the Blaise case ID on the external file.

The details of the NHCS are considerably more complex, and attempting to build this capability purely within the DEP would have been extremely challenging. The techniques added in Blaise 4.8 and described here allowed us to implement this solution within the project schedule.

SUMMARY

Blaise 4.8 allows developers to move within and between datamodels during data collection, organizing data into entities most appropriate for their projects. The full range of Manipula functionality is available at run-time during DEP. This movement can be interviewer-initiated through use of a button or question response, or can be initiated through code. The power of this technique has broad application across our most complex surveys.

Using Blaise for Implementing a Complex Sampling Algorithm

Linda Gowen and Ed Dolbow, Westat Inc.

Introduction

In 2012 Westat was awarded a contract to conduct a very large household survey. The survey included a roster of all household members. The roster collected demographics data along with many other study specific variables. These data from the roster were used in a complicated selection algorithm to choose household members to complete extended questionnaire instruments. The goal of the survey designers was to conduct “extended” interviews for no more than 2 adults, 2 children ages 12-17 and one child aged 9-11. In addition, they wanted to select persons with particular characteristics. In cases where there were multiple members, with these similar characteristics, they wanted a random selection applied.

Since the roster instrument and the extended interviews were being developed in Blaise, the system architects wanted to explore using Blaise to implement this complex algorithm. They were avoiding installs of additional software to the interviewer tablets that would have to be maintained and licensed. The algorithm required generating random numbers for implementing various sampling rates and sorting. In addition to developing these functions, we needed to be able to load test data to verify the system would correctly yield the sampling rates. We were not sure if Blaise was the “perfect” tool for these functions, but definitely thought it was worth a try.

This paper will explain our Blaise approach. It will describe how we used the Blaise Random Function to meet the requirements for applying specific sampling rates. Also, we will discuss how we programmed a bubble sort in Blaise code to conduct a sort and then finally we will discuss our test methods and the results.

Random Number Generator

The first function we discovered to be very helpful was the RANDOM function. It's simple to use. Here is the screen shot from Blaise help.

Purpose
Returns a random number.

Syntax
`RANDOM [(N)]`

N is an integer expression. Real expressions will be rounded to integer values.

If you specify **N**, the result **R** is an integer random number in the range 0 .. **N** - 1 (0 <= **R** < **N**). If you do not specify **N**, **R** is a real number in the range 0 .. 1 (0 <= **R** < 1).

Remarks
By default, each time a program is started, a new seed value will be determined for the random function. Because of this, the result of the random function differs systematically from one run to another. Only by using the `SETRANDOMSEED` instruction the same result from one run to another can be reproduced.

Example
`X:= RANDOM ((A**2 + B**2) / C**2 + LEN ('1'))`
`Y:= RANDOM (5)`
`Z:= RANDOM`

We did not seed the RANDOM function. We simply defined the receiving field, like the example below, to generate a uniform distribution between 0.001 and 1.000"

RAND1 (RAND1)
{English text}
"Person Random Number 1 generated following
a uniform distribution between 0 and 1."
: 0.001..1.000,EMPTY

We conducted an analysis in SAS on a dataset of 15,800 records generated by our Blaise program. We wanted to see if the random numbers were evenly distributed.

Mean	0.500607	Std Deviation	0.28960
Median	0.500000	Variance	0.08387
Mode	0.146000	Range	1.00000
		Interquartile Range	0.50400

The results prove the distribution of the random number was very good and reliable for using it for the purpose of random selection.

Quantile	Estimate	
Max	1.000	100%
99%	0.991	
95%	0.951	
90%	0.901	
75% Q3	0.752	
50% Median	0.500	
25% Q1	0.248	
10%	0.101	
5%	0.049	
1%	0.010	
0% Min	0.000	

Duplicate Random Numbers

In order to have a true random ordering of household members, we needed unique random numbers for each person. We realized early on, it was quite possible there would be duplicate random numbers among persons within a household. The odds are 1 in 1000 with the range .001 to 1.000 for one additional person in the household. The odds increase with each person in the household. So, we came up with the solution to check for duplicates every time a random number was generated after the first person. We designed the program to regenerate a random number up to 3 times if a duplicate was found among household members. In our test of 5700 households with 15,800 people, there were no duplicates. Our test included households with up to 12 people in them. We did not keep track of how often a number was regenerated. It would be interesting to know. Also, we could probably have increased the precision (places to the right of the decimal), to decrease the incident of duplicates. But, I think we still would have looked for duplicated to guarantee there were none.

Bubble Sort

The requirements for our household member selection were to generate a random number and sort the members by the random number generated. We decided that a bubble sort would work well in Blaise.

We programmed the simplest bubble sort algorithm modeled after the following:

```
For I:=1 TO 10 DO {simple sorting algorithm}
  For j:=1 to 9 DO
    IF SortedArray[J] > SortedArray[j+1] THEN
      k:= SortedArray[J]
      SortedArray[J]:= SortedArray[j+1]
      SortedArray[j+1]:=k
    ENDIF
  ENDDO
ENDDO
```

We replaced the upper bound of 10 for the outside “I” loop to be the number of members in the household. We replaced the upper bound of the inside “j” loop with the number household members minus 1. We didn’t see a performance issue with this sort, so we went no further with trying to optimize it. There are many ways to optimize a bubble sort by checking for swaps, but we felt this would complicate the code when it wasn’t necessary.

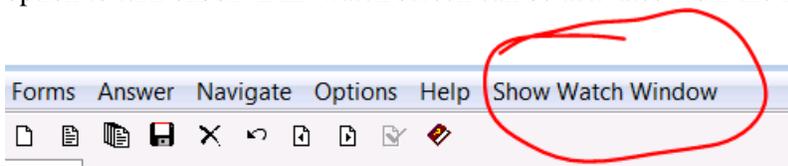
Testing

We conducted 3 levels of testing. (1) Unit Testing, (2) Bulk Load Testing, and (3) End to End Testing.

Unit Testing consisted of creating a testing environment with just the sampling algorithm code where we could control the many different parameters. We used the DEP Watch window extensively. Below is a screen shot from our unit testing. This screen shot shows 5 persons in RAND1PersArray, the random number generated for each of the persons stored in RAND1SortArray and the person numbers sorted in the Pnum1 array.

```
*** Selected fields ***
BHHR1.Rand1PersArray[1]: 1
BHHR1.Rand1PersArray[2]: 2
BHHR1.Rand1PersArray[3]: 3
BHHR1.Rand1PersArray[4]: 4
BHHR1.Rand1PersArray[5]: 5
BHHR1.Rand1PersArray[6]: <empty>
BHHR1.Rand1PersArray[7]: <empty>
BHHR1.Rand1SortArray[1]: 0.021
BHHR1.Rand1SortArray[2]: 0.822
BHHR1.Rand1SortArray[3]: 0.453
BHHR1.Rand1SortArray[4]: 0.460
BHHR1.Rand1SortArray[5]: 0.461
BHHR1.Rand1SortArray[6]: <empty>
BHHR1.Rand1SortArray[7]: <empty>
BHHR2.pnum1[1]: 1
BHHR2.pnum1[2]: 3
BHHR2.pnum1[3]: 4
BHHR2.pnum1[4]: 5
BHHR2.pnum1[5]: 2
BHHR2.pnum1[6]: <empty>
BHHR2.pnum1[7]: <empty>
```

To activate the DEP Watch Window, the option “/!” has to be set when calling the DEP program. The option to turn off/on DEP watch screen can be activated from the Blaise Data Entry Screen.



The second testing we conducted is **Bulk Load Testing**. We took a set of 570 cases with fields set to values representing test scenarios and duplicated them 100 times. We created Manipula programs to set the values in fields and to duplicate the cases 100 times. Once the manipula programs loaded the data and the Blaise program automatically generated the random numbers for each case were ready to run our systematic analysis of the results. We conducted our analysis on the large numbers cases to see if the program was truly yielding the selections at the targeted sampling rates.

Final testing is **End to End Testing**. We integrate our sampling algorithm into the larger system which includes a (1) study management system (SMS) calling a (2) fully implemented instrument which collects the roster and then calls the (3) sampling algorithm. There are lots of data moving between the components of the system. There are data passed as parameters to the instrument from the SMS and data generated by the instrument collecting the roster. Both the data from the SMS parameters and the data collected by the instrument are used by the sampling algorithm. All of this data have to work together and can be complex. We have to make sure all of the components are using correct fields in the correct places with correct field types. Once all three types of testing are completed we can feel confident that our complex sampling algorithm will work in the field.

Conclusion

Everyone involved with the project was pleased at how well the sampling algorithm was implemented in Blaise both in performance and results. It was a huge advantage to stay with the same technology used by the data collection instruments. This way we avoided the additional complexities of maintaining and integrating different software and managing more licenses.

Making the most out of Manipula-Maniplus

Fred Wensing, Independent IT contractor, Australia

1. Introduction

Most people who are introduced to Blaise soon become aware of the questionnaire language and its capabilities. They start to develop questionnaires and hope to start collecting data. Before long they realise that they will need to draw on the other components of the Blaise software to help them load the input data, extract the output data and in many cases help them to manage and run the survey.

If they have their own survey infrastructure they may look at the Blaise API and its features. But not every organisation has the infrastructure or skilled staff to work with the Blaise API. That is usually the first time when they think that Manipula may help, so they may turn to the Online Assistant or the Manipula Setup Wizard.

For many people using the Blaise software, the Manipula Setup Wizard is their first exposure to the Manipula language and capability. However, the Wizard only handles some simple data conversions. There is much more to Manipula than is found in the Wizard. In particular there is a whole suite of statements and functionality that has been added to Manipula, called Maniplus.

Maniplus has been a feature of Blaise since version III and it has been extended with every release since then. Maniplus now provides comprehensive functionality which can enable full systems to be developed with little reliance on other software. This paper will highlight some of the features of Manipula-Maniplus that make it an excellent tool for developing systems.

This paper is not aimed at replacing or competing with the comprehensive documentation in the Blaise Online Assistant but will just highlight useful features that some people may not be aware of to encourage you to give it a go.

2. Basic description of Manipula-Maniplus

Manipula is a processing language that can be used to carry out manipulations of Blaise and other data files (ASCII, XML, OLEDB). It can combine data, derive new data, sort records, define filters, and perform complex computations.

A basic Manipula program consists of a list of settings, datamodels and file definitions followed by some processing logic which is contained within a MANIPULATE step which contains the main program logic. The program logic can be extended with a PROLOGUE, SORT and/or a second MANIPULATE step for text file processing. Additional functionality can be obtained using PROCEDURE sections which are called from either of the MANIPULATE steps.

The Maniplus language encompasses almost all of the Manipula language with extended statements that provide for interaction with the operator (through dialogs or menus) as well as the ability to call other programs, file management and other special functionality.

A Maniplus program commences with the keyword PROCESS and the main processing logic is contained within the MANIPULATE step. Flexibility is obtained within Maniplus through the use of PROCEDURES or calls to other programs.

3. Blaise datamodel awareness

The overriding advantage of using Manipula or Maniplus is the ability to interact directly with Blaise datamodels. To access any Blaise file you just need to mention the relevant datamodel in the USES section and associate it with the corresponding file (INPUTFILE, OUTPUTFILE, UPDATEFILE etc). The file can be a native Blaise data file or one that can be accessed using OLEDB Information file (BOI file). The latter capability is part of the Datalink features of Blaise Components (licensed separately).

Text files (ASCII, ASCIIRELATIONAL, FIXED, XML) can also be accessed using a Blaise datamodel but that datamodel can be defined locally (within the program) rather than separately, if so desired.

4. Some auto-execute features of Manipula

Manipula was developed to make some basic processing easy to do. For that reason there are some auto-execute features of Manipula. I will call them AUTOREad, AUTOCopy and AUTOWrite. The first two of these are settings which you can change to YES or NO. The last is not a setting but a behaviour outcome if you leave out the MANIPULATE step.

These auto-execute features can make the programming of Manipula quite easy, provided that you know they are active (or not). They can, however, be a source of confusion for inexperienced programmers which is why I suggest that the AUTOREAD and AUTOCOPY settings should be specified with their relevant setting at the top of the program, even if you are using the default settings.

In Maniplus AUTOREAD is always set to No, so file reading is your responsibility. That can readily be handled using READNEXT and the REPEAT UNTIL logic:

```
Infile.OPEN
REPEAT
  Infile.READNEXT
  <other statements>
UNTIL Infile.LASTRECORD
```

AUTOCOPY is a great feature that copies the input fields automatically to the output fields with matching names, immediately after reading the record (by any method). This means that most data transfer can happen “automatically”. Blaise does this by connecting the matched fields. Individual Blaise files can be withheld from the auto-copy feature by turning off by changing the CONNECT setting to NO.

5. Sort and summarise

Before I mention the features of Maniplus, I want to mention a useful feature of Manipula which is the ability to sort and summarise. This feature only applies to the first ASCII output file in a Manipula Setup. Sorting enables you to output your text file in whatever order you like. By adding summary keywords such as SUM, MEAN, MAXIMUM, MINIMUM, STDDEV, VARIANCE and MEDIAN to your sort

variables you can readily obtain some summary data that can be used for reporting. While the Blaise software no longer includes a tabulation component this feature of Manipula allows for some basic summary data to be obtained. If you output the data to CSV or Fixed format you can produce some simple reports for a system.

6. Dialogs – the visible interactive part of Maniplus

The most useful aspect of Maniplus is the addition of Dialog boxes which can be used to interact with the operator. Early versions of these were basic but the flexibility has improved greatly over the years.

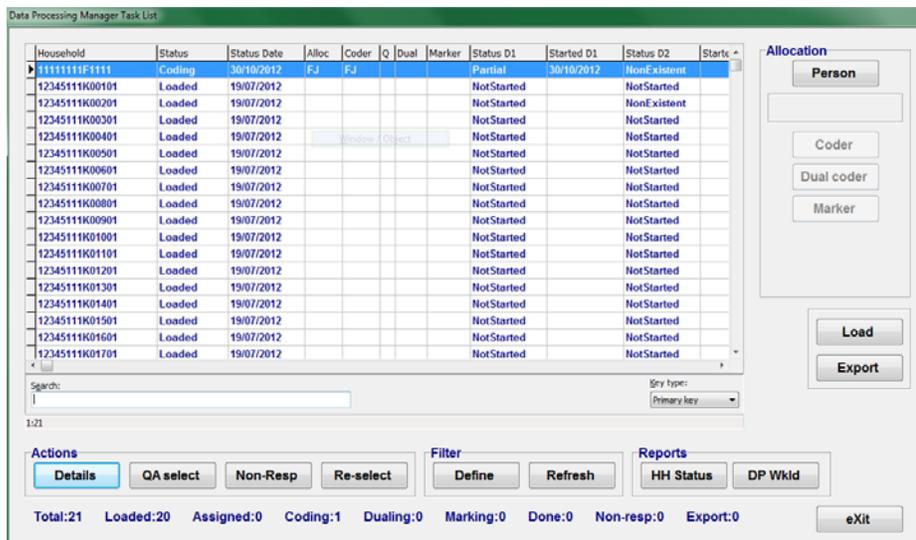
The dialog boxes can show a list of records, content of fields, text elements and buttons to enable actions to be initiated. The elements can be made visible and/or enabled based on Boolean expressions. The placement of elements on the screen is done using x,y coordinates that are measured in pixels.

With Dialog boxes it is possible to create a comprehensive interface that can provide the user with access to lots of functionality.

Dialog boxes can also contain multiple lookups which can be synchronised by using ‘on change’ instructions to refresh the lookup entries.

Some examples are shown here.

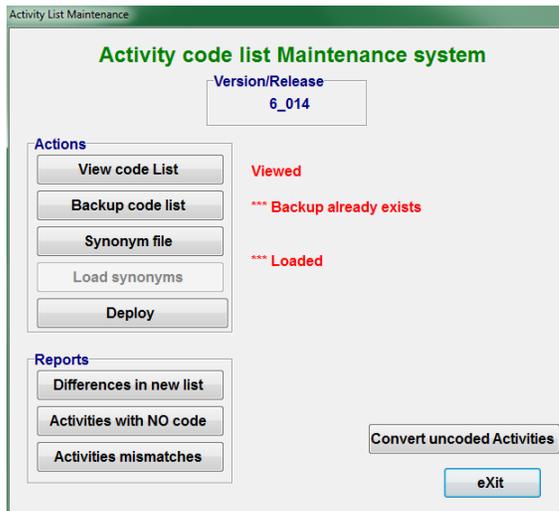
Example 1. Typical screen to manage a survey list with action buttons, filters and record counts.



Buttons and objects are made visible or enabled using Boolean conditions attached to the button:

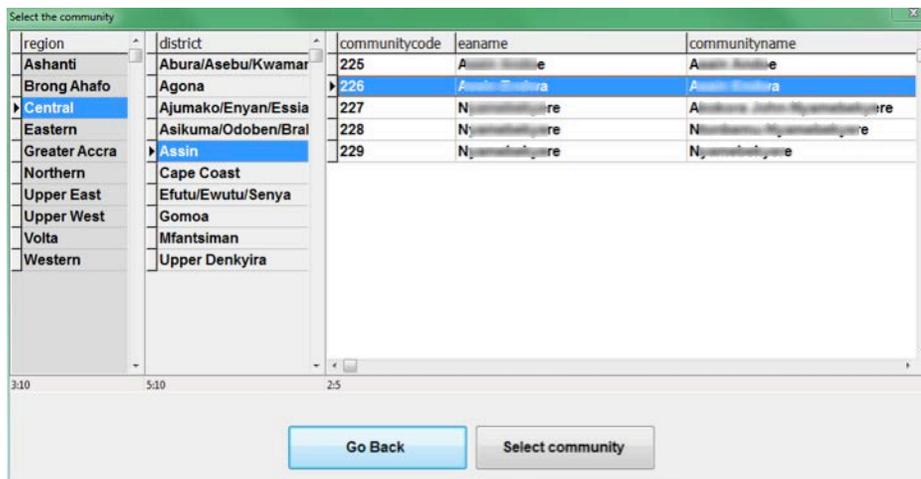
```
BUTTON aButton1 VALUE=bAllocA
    CAPTION=' Coder '
    ENABLED=((aRole<>EMPTY) AND (aOneCoder<>EMPTY))
BUTTON aButton1 VALUE=bCHStatus
    VISIBLE=(aTestMode='ON')
```

Example 2. Small utility screen for code list maintenance.



Buttons are intended to be used in sequence and outcomes are reported back to the screen.

Example 3. Selector screen showing three synchronised lookups.



Changing the selection in lookup columns 1 and 2 will cause immediate change in the value of the other lookup columns. This is done using temporary files for each lookup and reloading them using ONCHANGE to activate a procedure to do the reload:

```
lookup tmpR
  size=(150,520)
  onchange=prc_districts
lookup tmpD
  size=(200,520)
  position=(150,0)
  onchange=prc_Communities
lookup tmpC
  size=(674,520)
  position=(350,0)
```

Example 4. Screen showing buttons that are color coded for progress. These buttons are displayed using a set of graphic images.



The graphic images are stored in a resource folder and activated using APPEARANCE and SRC options. The references to the images are changed by the program so they will differ depending on progress through a section:

```
BUTTON aTryButton SIZE=(175,75) POSITION=(70,50)
  APPEARANCE = IMAGEBUTTON
  SRC = "^trybuttonfile[7]"
  VALUE = bu_s02Land
  CAPTION = ' '
```

7. Menus – less visible but functional

Another useful aspect of Maniplus is the provision of Menu facilities. These are activated by applying the MENU method to a file containing the menu specifications. Selecting from the menu effectively selects one of the records from the file. The selection can be tested and actions or procedures activated.

The attributes of each record in the menu file define the action that will happen.

The Blaise Online Assistant and sample files can show you how to implement this feature.

Use of menus is a little limiting because you cannot have them on the same screen them together with action buttons.

I must admit that I prefer the Dialog box functionality over the Menus when it comes to building systems for inexperienced users because dialogs are more visible and interactive.

8. Parameters

The operations of any Manipula program can be tailored through the use of parameters. This is particularly useful for passing information from other systems or between programs.

The use of parameters makes it easier to break down your system into manageable parts. It enables those parts to be developed and tested separately, and sometimes run stand-alone, before integration into a system.

Parameters are passed into a Manipula or Maniplus program via one of the command line options (/P). Multiple parameters are separated by the ';' character.

Parameters are accessed using the PARAMETER function with a reference to the parameter number. It is useful to set some default values in case the parameters are not invoked. Code like this can be put at the top of the MANIPULATE step:

```
{Check parameters for system settings}
IF PARAMETER(4)<>' ' THEN
    aTestMode := PARAMETER(4)
ELSE
    aTestMode := 'OFF'
ENDIF
```

Aside from the user defined parameters, Manipula also has a system PARAMETER(0) which returns the name of the program and path which is executing the Manipula/Maniplus Setup. For example:

```
aManiProgram:=PARAMETER(0)
```

on my computer this returns:

```
C:\Program Files\StatNeth\Blaise 4.8 Enterprise\Bin\Manipula.exe
```

This result can conveniently be used to extract the path to enable Maniplus to call other Blaise executables like the B4CPars.EXE or CAMELEON.EXE.

9. Command Line options and Command Line options File (BCF)

In addition to parameters the command line can also be used to affect changes to the way that the Manipula or Maniplus program operates.

You can supply the names of files to be used (using /I and /O or /N), the names of working folders (/W) or search paths (/E) and other operational behaviour. In this way the same program can be used to operate on different files. For example:

```
CALL ( 'myprog.MSU /CC:\Surveys\Config\General.miw'
      + ' /EC:\Surveys\External /HC:\Surveys\MySurvey'
      + ' /WC:\Surveys\MySurvey' )
```

When the command line options become numerous it can be easier to put the options into a small text file and reference that file using the command line option of @<filename>. The format of such a file follows the style of an INI file with the structure

```
Option_name=value(s)
```

The full list of Manipula command line options (and others) can be found in the Blaise Online Assistant under “Miscellaneous / Command line options”.

A typical use of this feature may be to set the working folder, configuration files and meta search path. For example, the above command line settings would look like:

```
[ManipulaCmd]
ConfigFile=C:\Surveys\Config\General.miw
ExternalSearchPath=C:\Surveys\External
MetaSearchPath=C:\Surveys\MySurvey
WorkFolder=C:\Surveys\MySurvey
```

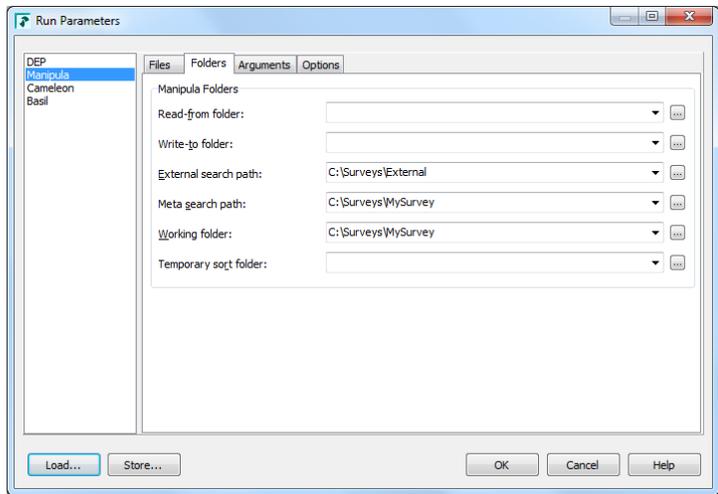
The first line of this file indicates that the settings/options apply to Manipula/Maniplus.

When the Manipula is invoked this file of settings is passed in using @<filename> in the Call to Manipula, namely:

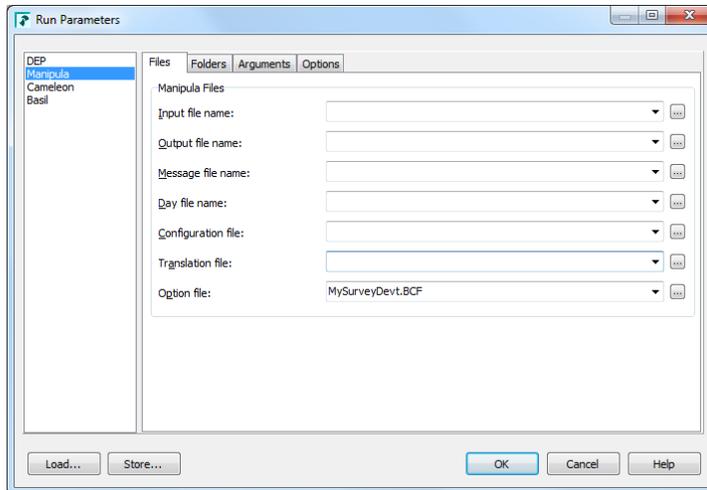
```
CALL ( 'myprog.MSU @Myprog.BCF' )
```

Blaise Command line options file (BCF) goes one step further than parameters or Command lines in that they contain the command line and environment settings and/or run parameters for multiple applications (eg. for Manipula and DEP at the same time).

A BCF can be created from within a Blaise project by using the Store button at the end of updating the Run / Parameters. Of course you can also create one manually.



The BCF can also be used in conjunction with a Blaise Project File to ensure that the relevant settings are applied while developing or testing your project. The BCF can be entered into the Option file entry in the Run Parameters screen.



Using a BCF means that all the settings are stored in the one file. That BCF can be used in a Windows shortcut or in any call to a Manipula program.

Of particular use in these BCF files are the path (or folder) entries which avoid the need to have full path references in the names of data files or data models. By setting up different BCF files for Development, Test and Production environments you can simply move the application between environments without the need to recompile it each time.

Although the BCF file can be created using the Store button the file that it creates is just a text file and additional entries can be added if needed.

10. Using an INI file

If you have a Manipula system then parameters can also be passed in at the start but that can make it difficult for an operator who doesn't know what parameter settings to use. This can be resolved by setting defaults within the program or by providing a small initialisation text file (*.INI) containing various settings that control the system.

An INI file is a text file that contains a list of settings in the form:

Setting_name=value

Of course the Manipula program will need to read this file to obtain the settings. That can be done with a simple file definition and some logic to read the settings.

For example. The datamodel, file definition and procedure to read such a file could look like:

```

USES
  DATAMODEL mSettings
    FIELDS
      ItemName : STRING[20]
      ItemValue : STRING[200]
    ENDMODEL
  INPUTFILE
    fSettings : mSettings ('',ASCII)
  SETTINGS
    SEPARATOR=' ' OPEN=NO

```

```

PROCEDURE proc_settings
  fSettings.OPEN ('TUSDiary_System.ini')
  fSettings.READNEXT
  REPEAT
    IF UPPERCASE(ItemName)='VERSION' THEN
      aVersion:=ItemValue
    ELSEIF UPPERCASE(ItemName)='INSTALLER' THEN
      aInstaller:=UPPERCASE(ItemValue)
    ELSEIF UPPERCASE(ItemName)='SYSFOLDER' THEN
      aSysfolder:=ItemValue
    ELSEIF UPPERCASE(ItemName)='CODERLOCATION' THEN
      aCoderLocation:=ItemValue
    ENDIF
  fSettings.READNEXT
  UNTIL fSettings.EOF
  fSettings.CLOSE
ENDPROCEDURE

```

11. Controlling the start-up screen

When Manipula or Maniplus start-up the system brings up a splash screen. It is possible to change the splash screen colours, logo icon and default font through the MANIPULA.MIW file. The Manipula.MIW file is based on standard INI file structure.

The following is an example of a typical MIW file that uses full screen, shows a logo and sets the default font to bold 12 and has a splash colour gradient from purple to teal:

```

[Maniplus]
Maximize=1
OwnLogo=1
LogoName=AgencyLogo.bmp
FontSize=12
Bold=1
ColorStart=Purple
ColorEnd=Teal
FontName=Arial

```

This is the first way that you can customise your Manipula/Maniplus processes.

To find out more lookup “How to use... / Maniplus / Customising a Maniplus Setup” in the Blaise Online Assistant.

12. Running other programs

Maniplus can run other programs, either through the CALL instructions (for Manipula/Maniplus programs) or the RUN instruction which can execute any other type of program. And, of course a Maniplus program can be used to call the Data Entry Program (DEP) through the EDIT method.

Manipula/Maniplus components can be developed separately and connected to the main program by having it CALL those parts.

If non-Blaise programs are needed (eg. STATA or SAS) then they can be called using the RUN command.

These features give a Maniplus system lots of flexibility. When the programs you call or run are Manipula/Maniplus programs then the system is more homogeneous. More importantly the Blaise files that you use or create in one part are directly accessible to the other part of your system.

With the file handling extensions to the RUN function even some of the file management can be handled in Maniplus (see next section).

13. File handling

In the early implementation of Maniplus the only way to create folders and copy or delete files was to use the various command line statements to be executed by the CMD.EXE Windows command line interpreter.

From version 4.6 onwards, however, the RUN instruction handles a range of file management commands making it much smoother to handle file management tasks. Some of these commands include:

```
RUN('SETREADONLYFLAG FileName')
RUN('REMOVEREADONLYFLAG FileName')
RUN('DELETEFILE FileName')
RUN('MOVEFILE SourceName TargetName')
RUN('COPYFILE SourceName TargetName')
RUN('CREATEDIRECTORY DirName')
RUN('REMOVEDIRECTORY DirName')
RUN('DELETEBLAISEFILE FileName')
RUN('MOVEBLAISEFILE SourceName TargetName')
RUN('COPYBLAISEFILE SourceName TargetName')
RUN('RENAMEBLAISEFILE SourceName TargetName')
RUN('DISCONNECTBLAISEFILE FileName [ IGNORESESSIONS ]')
RUN('DISCONNECTDICTIONARY FileName')
```

Of particular use are the instructions to copy or delete Blaise files because these are aware of all the file types which make up a Blaise file. The Blaise datamodel is unaffected when some of these file management commands are applied to the data.

```
{Close the data file before copying}
fMaster.CLOSE
fMaster.RELEASE
{copy the main Blaise data files to another location and name}
fMaster.COPYDATAFILE(C:\Surveys\Data_Local_'+aIDNumber)
```

In this example the COPYDATAFILE copies the Blaise data to the new location C:\Surveys with the name Data_Local_<aIDNumber>.

14. Zip utilities

Zip functionality was added to Maniplus with version 4.8 of Blaise. Zip and Unzip operations can now be performed seamlessly within a Maniplus application.

For example the following statements Zip the main data file to a backup folder with a date-specific name:

```
{set up date for zip file names}
aZipDate:=DATETOSTR(SYSDATE,YYMMDD)
{set up name of Zip file}
aFilenameOut:='Backup_Main_'+aZipDate

{set name of HH file to be zipped}
aFilename:='MasterHH'
{Prepare exclusion list for Zip files}
IF NOT (FILEEXISTS('Exclude_zip1.txt')) THEN
    Proc_CreateExcludes (aFilename,'Exclude_zip1.txt')
ENDIF
{Zip main HH file to backup using an exclusion list}
aResult := ZIPFILES ("backup\'+aFilenameOut+'.zip"
    "'aFilename+'.*' /X@Exclude_Zip1.txt')
```

This example uses a small procedure called Proc_CreateExcludes to create a list of files to be excluded from the ZIP process and that file is invoked using the /X@<filename> option. The procedure is listed here for information:

```
{Create a file with the list of filetypes which don't need to be put into
the Zip file}
PROCEDURE Proc_CreateExcludes
    PARAMETERS
        Datafile : STRING
        Outfile : STRING
    INSTRUCTIONS
        fSetup1.OPEN (Outfile)
        fSetup1.TextData := Datafile+'.bat' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bpf' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bla' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bdm' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bmi' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bxi' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bdv' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bdp' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.bww' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.buf' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.aif' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.adt' fSetup1.WRITE
        fSetup1.TextData := Datafile+'.~lk' fSetup1.WRITE
        fSetup1.CLOSE
    ENDPROCEDURE
```

15. Late binding (or generic setups)

One issue that can be a problem is that Manipula and Maniplus programs are usually compiled with the datamodels that they need to access the various Blaise data files. This means that if you change one of the datamodels in any way you will need to recompile the program before you can use it.

Since version 4.8 of Blaise, however, Manipula can be prepared with the name of the datamodel specified as a variable provided on the command line at the time of execution. This feature is known as late binding which enables the program to be compiled without knowledge of the datamodel. A program which uses this feature is considered to be a generic setup because it can be used with any datamodel (provided it is compatible with the statements).

You will be restricted in the kind of statements that can be executed in such a program but it does provide the flexibility for creating stand-alone utilities that do not need to be recompiled every time you wish to use it.

To use this technique the USES statement will have the VAR option, for example:

```
USES
  {instrument metadata is passed as a parameter}
  mData (VAR) 'xxx'
```

Then when the program is executed the name of the metadata is passed in using the /K command line option:

```
MakeZipData.MSU /KmData=C:\Surveys\Mysurvey\MasterHH.BMI
```

With late binding the statements in the Manipula program you can only access fields from those files which use the variable meta using GETVALUE / PUTVALUE / GETREMARK / PUTREMARK. These statements do not produce an error if the field being accessed does not exist in the datamodel.

For example the statements here read data from a few fields and write values to other fields:

```
{check the dates and copy status}
aCopyFlag:=Survey.GETVALUE('xCopyFlag',UNFORMATTED)
aSurveyDate:=VAL(Survey.GETVALUE('xDateChanged',UNFORMATTED))

{other statements}

IF aCopyFlag='1' THEN {not copied yet}
  {update the xCopyFlag on the main survey file}
  Survey.PUTVALUE ('xCopyFlag','2') {copied}
  Survey.WRITE
  {save the OfficeData and update xCopyFlag}
  OfficeData.WRITE
  OfficeData.PUTVALUE ('xCopyFlag','2') {copied}
ENDIF
```

16. Metadata access

A very useful extension of Maniplus in version 4.8 is the ability to access the metadata of a datamodel through a series of GET methods. Some of them are:

```
GETFIRSTFIELDNAME
GETNEXTFIELDNAME
GETNEXTROUTEFIELDNAME
GETMETAINFO
GETFIELDINFO
GETTYPEINFO
```

With these it is possible to use Maniplus to generate reports or construct other programs (in Manipula or other software language) to process Blaise files.

This new feature will eliminate the need to use Cameleon which is good because Cameleon will not be available in Blaise 5.

Recently a utility was completed that uses the Metadata extensions of Maniplus to a great extent. It is called the Statistical Script Generator and was produced by a collaborative team from the Blaise developers (Statistics Netherlands) and some Blaise users. I understand that the utility will be available in the next release of Blaise 4.8.4.

This utility can be used to produce a SAS, SPSS or Stata script to read exported Blaise data. It also has the functionality to do this for a selection of blocks/fields of a datamodel.

The main screen of this utility is shown here:

Generate script for Statistical Package

Package: SAS SPSS Stata [Settings...](#)

Meta file: ...

Language: ▾

Output path: ...

Input path: ...

Input name: ...

Script name:

Field selection: ... [Edit](#)

Blaise data: ...

Create ASCII file for import in SAS

[Run](#) [Cancel](#)

17. Other features

Some other extensions of Manipula/Maniplus features which I will mention but I don't have time to describe are:

- It is now possible to define functions using the FUNCTION specification
- The INTERCHANGE file setting allows sharing of files with the calling process (DEP or Maniplus)
- Maniplus can now interact with the active DayBatch files to add, delete or modify cases through various DAYBATCH file methods

18. Concluding remarks

Manipula / Maniplus has many features only some of which are described in this paper. Its main strength is its ability to handle Blaise files and Blaise metadata, but the functionality has been extended considerably over the years.

Manipula/Maniplus is well worth a second look if you are planning to build a system.

19. Acknowledgements

I would like to acknowledge that some of the examples used in this paper came from Lon Hofman (Statistics Netherlands) and Elana Cohen-Khani (ISSER - University of Ghana).

Experiences and Lessons Learned Converting to Blaise 5

*Richard Frey, Karen Robbins, Kathleen O'Reagan, and Nikki Brown
Westat, Inc.*

Abstract: Blaise 5 offers new features for multimode data collection. This paper discusses our approach to converting an existing survey running in Blaise 4.8 in CATI and CAWI modes to Blaise 5 and distributing the survey on mobile data capture devices (iPad and smartphone). We discuss the issues encountered during the code conversion process, efficiencies introduced by converting to Blaise 5, experience with templates and layouts, and lessons learned in approaching conversion of an ongoing survey.

I. INTRODUCTION

The initial focus for the adoption of Blaise 5 by many organizations will be the conversion of current Blaise 4.x applications to Blaise 5. These applications will vary in size, scope and mode, requiring organizations to develop Blaise 5 conversion strategies. These strategies will include areas of code conversion, modifications of converted code and the determination of target platforms; all of which will most likely be decided on an application by application basis.

Since the development of Blaise is ongoing this paper is a high level discussion of the conversion of a Blaise 4.8 application using the beta Blaise 5.0.1.553. We found some bugs but chose not to discuss them in this paper, the assumption being they will be fixed by the production release. The target mode for this conversion is web-based surveys (or CAWI) including a discussion of new features that provide mobile-app based access for iPads and iPhones.

II. KEY DESIGN CAPABILITIES

As one moves from Blaise 4 to Blaise 5 you see dramatic changes and none more so than in the Blaise Control Centre. The Blaise 5 Control Centre provides a development environment that dynamically integrates design elements with the application code. Also known as the integrated development environment (IDE), the Control Centre contains the Source Editor, the Layout Designer and the Resource Editor.

- The Source Editor is used to author and maintain the application code.
- The Resource Editor allows for the customization of graphical elements such as fonts, texts, media, and most importantly Templates and Resource Sets which are all contained in the Resource Database.
- The Layout Designer integrates the application code with the design elements of the Resource Database.

The layout of each question in a questionnaire is made up of one or more Templates. Templates are constructed from elements such as Grids, Panels, Data Values and Response Values, to get the desired question display. Blaise 5 provides a set of predefined Templates to handle numerous common question layouts.

A Resource Set is a user defined collection of graphical elements and Templates that are used to create standard layouts within an organization. Resource Sets can be defined to provide baseline support for related categories of surveys, e.g. device types, survey modes, etc

The combination of the application code for a survey and a given Resource Set comprise a Layout Set. In this manner, a single survey's application code can be combined with one or several Resource Sets (e.g., one for a web browser and one for an iPad).

This new feature within Blaise 5, the separation of the survey application code from the presentation and formatting specification, provides new flexibility for deploying a survey for use by different devices and modes with minimum recoding and ease of maintenance.

III. CONVERSION FROM BLAISE 4 TO BLAISE 5

The application we chose to convert was a multimodal (CATI/CAWI), household survey. We concentrated on the CAWI modifications necessary to produce a working Blaise 5 application since the target platform was the web.

A. General Steps

The Blaise 4 to 5 Source Converter tool was the starting point for the code conversion of our application. All that was needed were a few simple parameters entered into the Blaise 4 to 5 Converter dialog, Figure 1

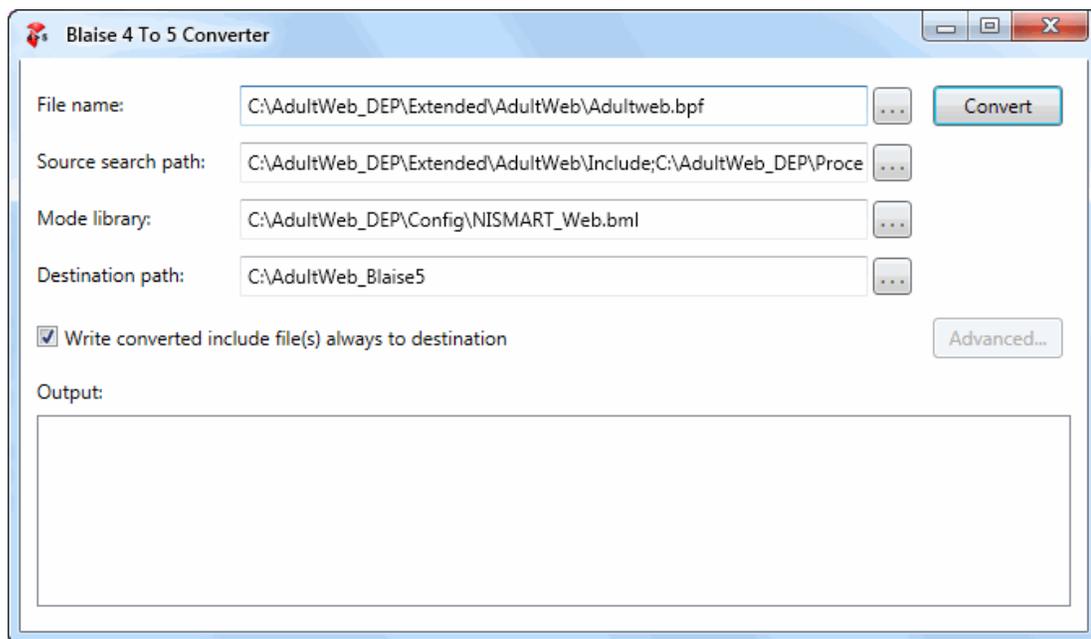


Figure 1

For the 'File name:' we chose to use the .bpf although we could have used the .bla. In the 'Source search path:' we selected folders containing include files, type libraries, procedures and configuration files.

The converter will then make the changes necessary to run the application under Blaise 5. We found that some Blaise 4 keywords such as NEWPAGE, NEWLINE and NEWCOLUMN were commented out in the new Blaise 5 code and identified as such in the Output of the Source Converter. Other Blaise 4

keywords such a LAYOUT and related keywords such as BEFORE, AT, GRID and FIELDPANE, were not commented out and were preserved. The LAYOUT and related keywords are not listed in the Output of the Source Converter. In addition to the Output of the Source Converter, we found a Blaise4to5.log file containing additional warnings and listing of created Blaise 5 files from Blaise 4 files with the new Blaise 5 file extensions.

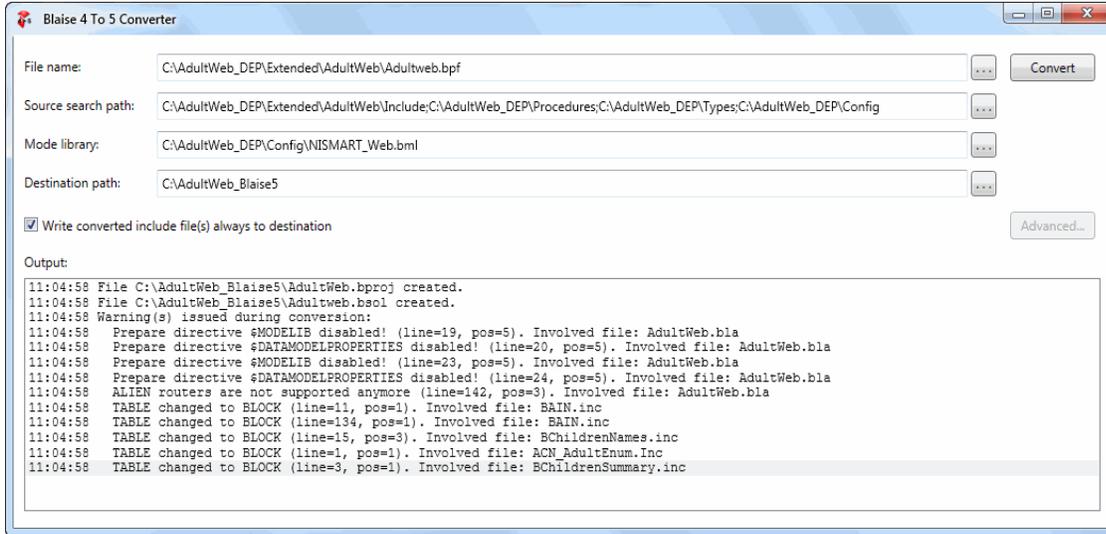


Figure 2

As you can see in Figure 2 the solution and project files were created. The Modelib and Datamodel Properties prepare directives were disabled and commented out in the source; while the Alien router and calls to it remain in the source. Lastly, the Blaise 4 keyword TABLE is replaced in the source with the BLOCK keyword.

Our application used prepare directives, \$MESSAGE, \$IFDEF, \$IFNDEF, \$ELSE and \$ENDIF, to distinguish between CATI and CAWI code and these directives were converted. In addition the conditional define from the Blaise 4 Project Options was carried over to the Blaise 5 source.

B. Lessons Learned

Once the conversion process was completed we had a Blaise 5 program that could have run without further intervention. However, after preparing the application in Blaise 5, we found that a number of warning messages were generated reflecting the keywords ignored by the parser.

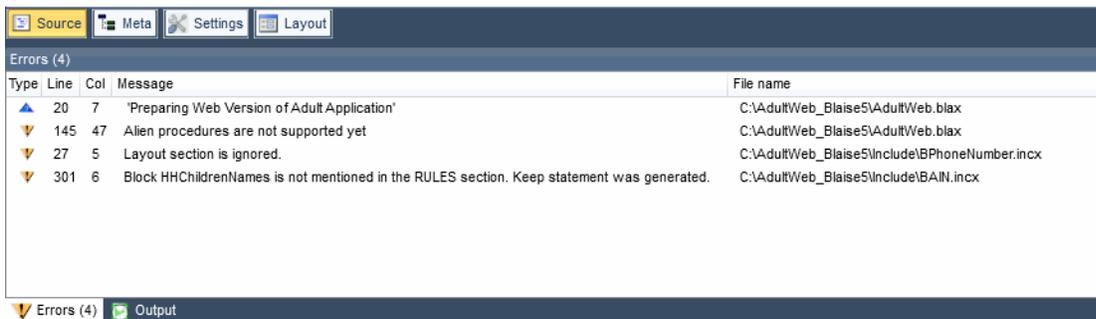


Figure 3

The warning messages, as shown in Figure 3, for items such as the Layout and alien procedures and their calls can be removed. Be aware that removing items that cause warning messages is an iterative process. If you prefer a cleaner set of code, we recommend you remove those items from the Blaise 4 code which are ignored by the parser or commented out by the converter.

IV. SUBSEQUENT MODIFICATIONS

During conversion Blaise 5 will assign a layout to each question based on the Modelib. It will also use defaults for the number of items on a page, the number of displayed columns for a response, style settings, master page and Resource Set.

The remainder of this section will describe some of the subsequent modification we made to our application.

A. Layouts

Blaise 5 provides the flexibility to create customized views through the use of layout instructions which Blaise 4 did not offer.

After running the application, as it was initially converted, we noted a number of different field layouts we needed to change. The first item we changed was the number of columns displayed for enumerations. The default is two columns, as shown in Figure 4, and we wanted our default to be one column.

Adult Web Instrument (2.21.307.012)

Welcome to the National Study about Children. You have been sent the link to this survey because we mailed a short questionnaire to your address and your household answered and gave your name.

Continue

According to your household's answers to our questionnaire, children age 18 or younger live or have lived in your household in the past 12 months, that is, since ^gp0. Is this correct?

Yes No

According to your household's answers in our questionnaire, ^gp1 takes care of the children most of the time. Is this correct?

Yes No

Are you ^gp1?

Yes No

Next

Figure 4

To set the global default number of columns to one we changed the 'Arrangement' property in the Resource Database to one column using the Resource Editor, Figure 5.

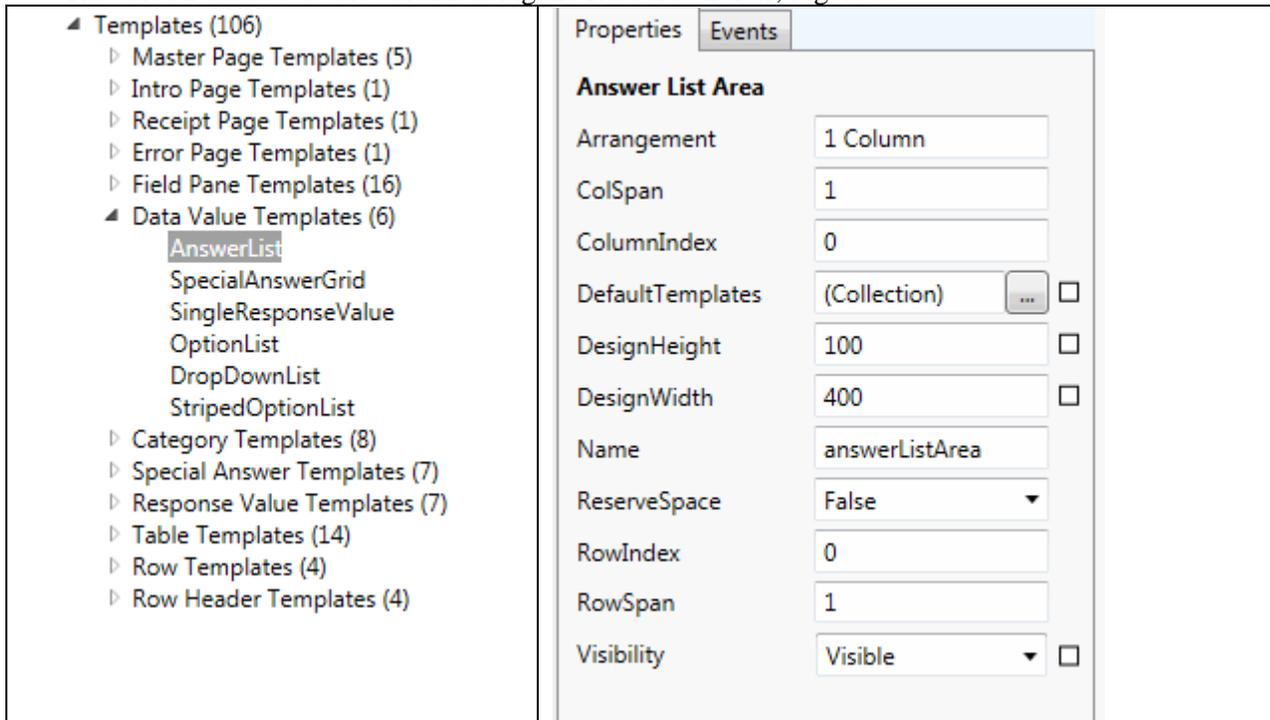


Figure 5

The result of the change is shown in Figure 6.

Adult Web Instrument (2.21.307.012)

Welcome to the National Study about Children. You have been sent the link to this survey because we mailed a short questionnaire to your address and your household answered and gave your name.

Continue

According to your household's answers to our questionnaire, children age 18 or younger live or have lived in your household in the past 12 months, that is, since ^gp0. Is this correct?

Yes
 No

According to your household's answers in our questionnaire, ^gp1 takes care of the children most of the time. Is this correct?

Yes
 No

Are you ^gp1?

Yes
 No

Next

Figure 6

The next change is to questions that are used for informational purposes such as welcome screen, interviewer instructions and closing text. Blaise 5 displays as its default not only the question text but also the response. As shown in Figure 6, there is a 'Continue' response associated with the welcome statement.

We wanted to remove the 'Continue' response because the question is used for informational purposes only and to reduce keystrokes. To remove the 'Continue' we changed the FieldPane from 'Vertical' to 'QuestionTextOnly' as shown in Figure 7.

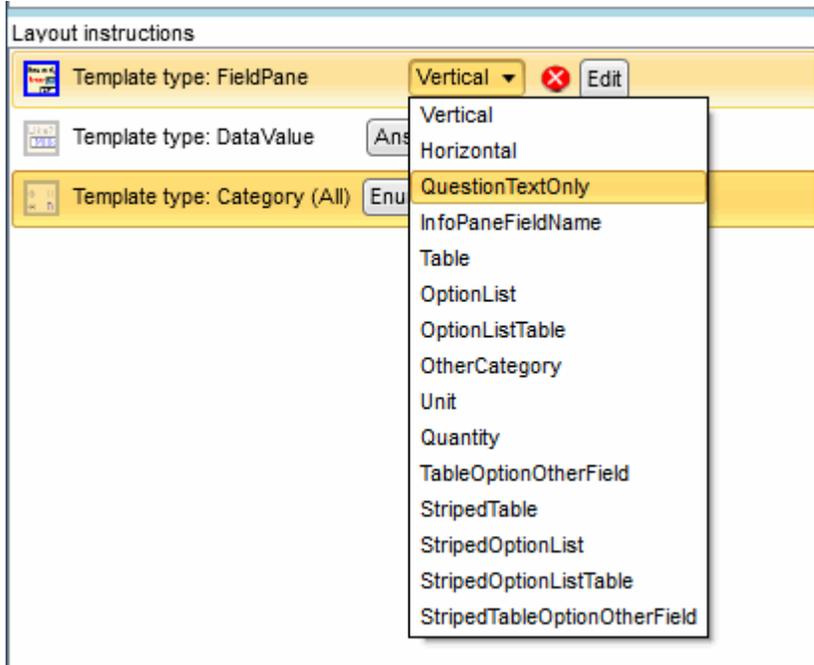


Figure 7

This question’s corresponding structure is pictured in Figure 8. The icon next to the field name AIN.auxWebIntroduction denotes the assigned ‘QuestionTextOnly’ template.

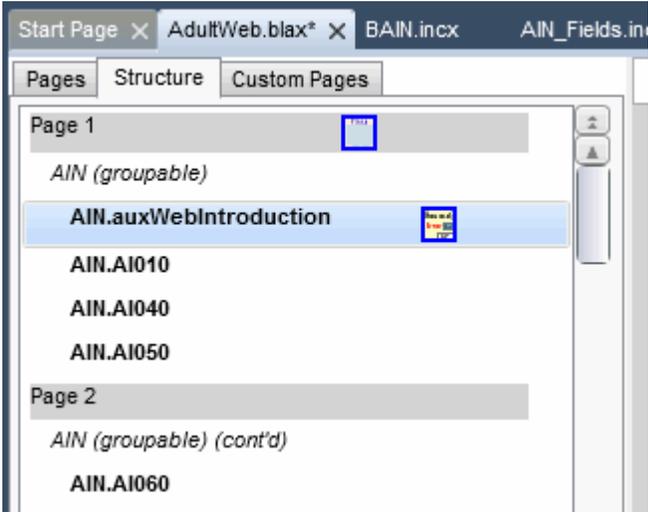


Figure 8

This resulted in the 'Continue' response no longer being displayed as shown in Figure 9.

Adult Web Instrument (2.21.307.012)

Welcome to the National Study about Children. You have been sent the link to this survey because we mailed a short questionnaire to your address and your household answered and gave your name.

According to your household's answers to our questionnaire, children age 18 or younger live or have lived in your household in the past 12 months, that is, since ^gp0. Is this correct?

Yes
 No

According to your household's answers in our questionnaire, ^gp1 takes care of the children most of the time. Is this correct?

Yes
 No

Are you ^gp1?

Yes
 No

Next

Figure 9

Another change we made was to make an enumerated list into a dropdown list. In this example, Figure 10, we wanted to change the month of a date of birth.

Adult Web Instrument (2.21.307.012)

Girl
 Don't know
 Rather not answer

What is ^gp0[pChildPointer]'s birthdate?

ENTER MONTH

JANUARY
 FEBRUARY
 MARCH
 APRIL
 MAY
 JUNE
 JULY
 AUGUST
 SEPTEMBER
 OCTOBER
 NOVEMBER
 DECEMBER
 Don't know
 Rather not answer

Previous **Next**

Figure 10

To create the dropdown we changed the DataValue Template from the default 'AnswerList' to 'DropDownList' as shown in Figure 11.

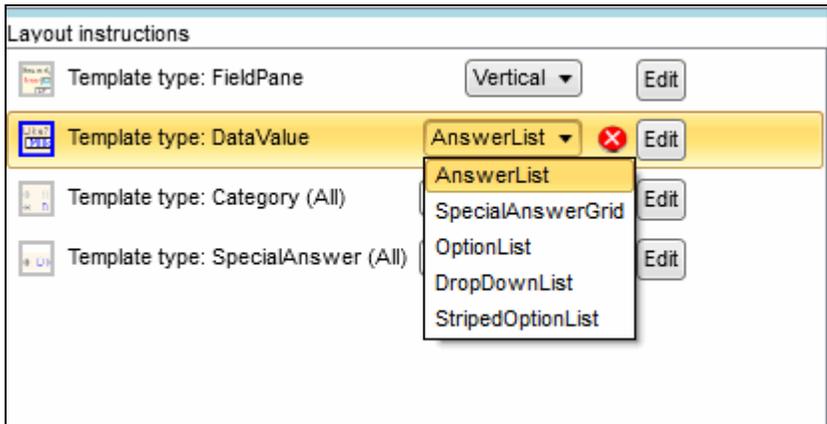


Figure 11

This resulted in the enumerated list being changed to a dropdown list, Figure 12.

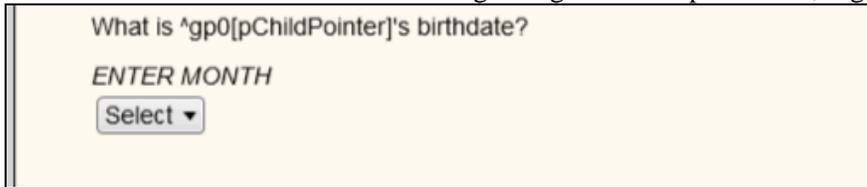


Figure 12

Blaise 5 gives you the ability to change the look of a question by modifying any of the “standard” Templates. In the Resource Editor, you can make a copy of the standard template you want to modify and use this as a starting point. How to make the very detailed formatting modifications is beyond the scope of this paper but can be found in the Blaise 5 Help.

B. Tables

The concept of tables is to place fields on a page in a structured view. Depending on the instrument’s structure some fields (arrayed blocks) lend themselves to be a table. After the conversion you’ll notice in the Layout Structure, blocks and arrays will have the term ‘(groupable)’ denoting that they are eligible to be displayed in a table format. In our case an example of this type of table would be a child roster as shown in Figure 13.

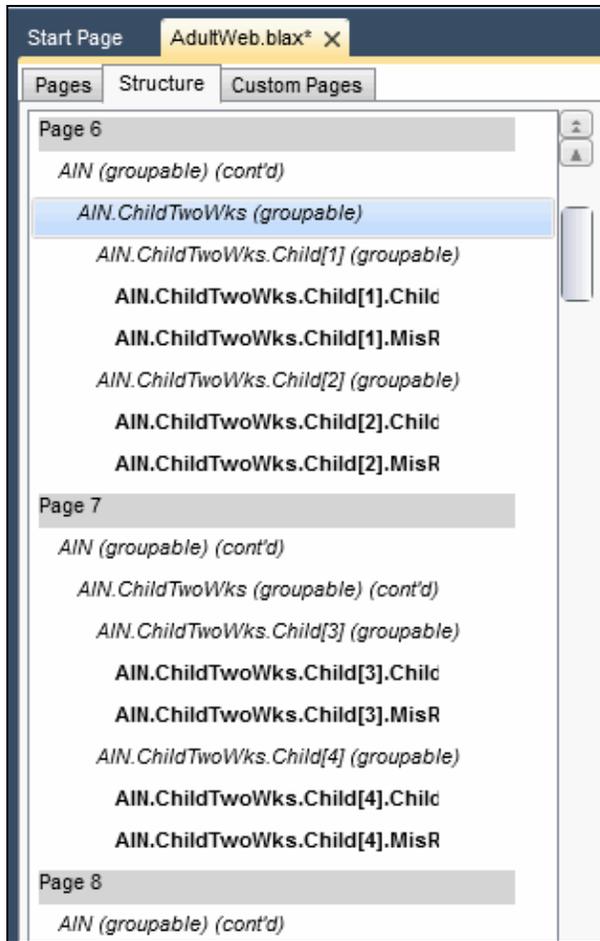


Figure 13

Figure 14 shows the default page layout before assigning any layout instructions to the Structure of Figure 13. You'll notice that the page layout displays the questions vertically with the question over the response field. This is the default layout for all questions in Blaise 5.

Adult Web Instrument (2.21.307.012)

Please provide a name, initials, or a nickname for all the children 18 years old or younger who have lived in your household for at least 2 weeks in a row since ^gp0.

ENTER ^gp1 CHILD'S NAME:

If you want to delete a child from the table check the delete row box corresponding to the child you want to delete.

^gp3

Please provide a name, initials, or a nickname for all the children 18 years old or younger who have lived in your household for at least 2 weeks in a row since ^gp0.

ENTER ^gp1 CHILD'S NAME:

If you want to delete a child from the table check the delete row box corresponding to the child you want to delete.

^gp3

Figure 14

To create the child roster table we first highlighted the 'AIN.ChildTwoWks (groupable)', Figure 13, in the Structure and then selected the 'Table: Default' grouping, Figure 15.

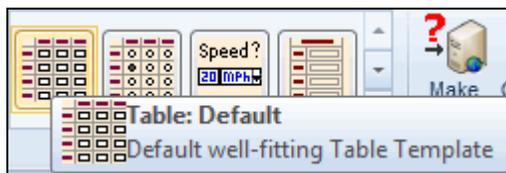


Figure 15

These actions resulted in a change to the Layout Structure, Figure 16. The Layout Structure now has the notation of 'TableGroup' and each member of the table has the notation 'TableRow'.

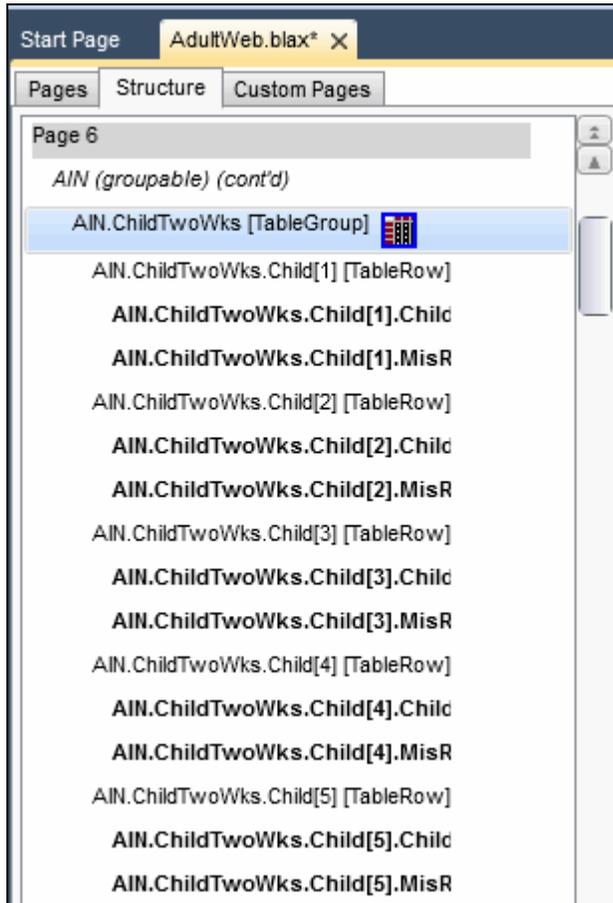


Figure 16

The result of adding the layout instruction, 'Table: Default', is we now see the child table displayed in a roster format, Figure 17.

Adult Web Instrument (2.21.307.012)

	Who Lived in Home Two Weeks in Row	Delete Child?
AIN.ChildTwoWks.Child[1]	<input type="text"/>	<input type="checkbox"/> ^gp3
AIN.ChildTwoWks.Child[2]	<input type="text"/>	<input type="checkbox"/> ^gp3
AIN.ChildTwoWks.Child[3]	<input type="text"/>	<input type="checkbox"/> ^gp3
AIN.ChildTwoWks.Child[4]	<input type="text"/>	<input type="checkbox"/> ^gp3
AIN.ChildTwoWks.Child[5]	<input type="text"/>	<input type="checkbox"/> ^gp3

^DispChildChildrenListed there any other children 18 years old or younger who live or have lived in your household part-time under a shared custody arrangement, but did not live in your household for 2 weeks in a row?

Yes
 No
 Don't know
 Rather not answer

How many children live or have lived in your household part-time under a shared custody arrangement, but did not live in your household for 2 weeks in a row?

Figure 17

When working with tables we need to distinguish between 'Groupable' and the Group instruction. 'Groupable' refers to the contents of blocks and arrays being put into a table format; as opposed to using the Group instruction to manually group related fields which will then be displayed in a horizontal or table like fashion.

C. Grouping

The concept of Grouping is new in Blaise 5. The Blaise 5 Group instruction allows you to tie together related fields in order to allow a horizontal or tabular display and/or a special behavior on the screen. Phone number, full name and date of birth are prime examples of fields that would be grouped.

In our example the first grouping we did was for the child's date of birth. In the source we grouped the date of birth fields, month, day and year and their applicable rules using the Group instruction, Figure 18.

```

GROUP ChildDOBGroup "What is ^HHChildrenNames.ChildrenNames.Person[pChildPointer].ChildFName's
birthdate?"
  FIELDS
  ChildBirthMonth (AIN2460)
  {What is ^HHChildrenNames.ChildrenNames.Person[pChildPointer].ChildFName's birthdate?}
  "<NEWLINE><I>ENTER MONTH</I>"
  : TMonth,DK,RF
  ChildBirthDay (AIN2480)
  "<I>ENTER DAY</I>"
  : TDay,DK,RF {T11_31}
  ChildBirthYear (AIN2500)
  "<I>ENTER YEAR</I>"
  : TYear,DK,RF {T11990_2015}
  RULES
  ChildBirthMonth
  IF ChildBirthMonth = RESPONSE AND ChildBirthMonth <> EMPTY THEN
    DoBMonthInteger := ORD(ChildBirthMonth)
    IF (ChildBirthMonth.ORD = 4) OR (ChildBirthMonth.ORD = 6) OR (ChildBirthMonth.ORD = 9) OR
(ChildBirthMonth.ORD = 11) THEN
      ValidDay29 := '29'
      ValidDay30 := '30'
      ValidDay31 := ""
    ELSEIF (ChildBirthMonth.ORD <> 2) THEN
      ValidDay29 := '29'
      ValidDay30 := '30'
      ValidDay31 := '31'
    ELSE
      ValidDay29 := '29'
      ValidDay30 := ""
      ValidDay31 := ""
    ENDIF
  ENDIF
  ChildBirthDay
  ChildBirthYear
ENDGROUP

```

Figure 18

In our application we wanted the DOB question text to appear across all three fields. In order to this we moved the question text from the month field to the Text component in the Group instruction.

The rules in the Group instruction were taken from the Rules section in the application. Inserted in place of the rules in the Rules section of the application is the Group Identifier, ChildDOBGroup.

As a result of grouping the month, day and year fields, they are now termed a 'Group' in the Layout Structure, Figure 19.

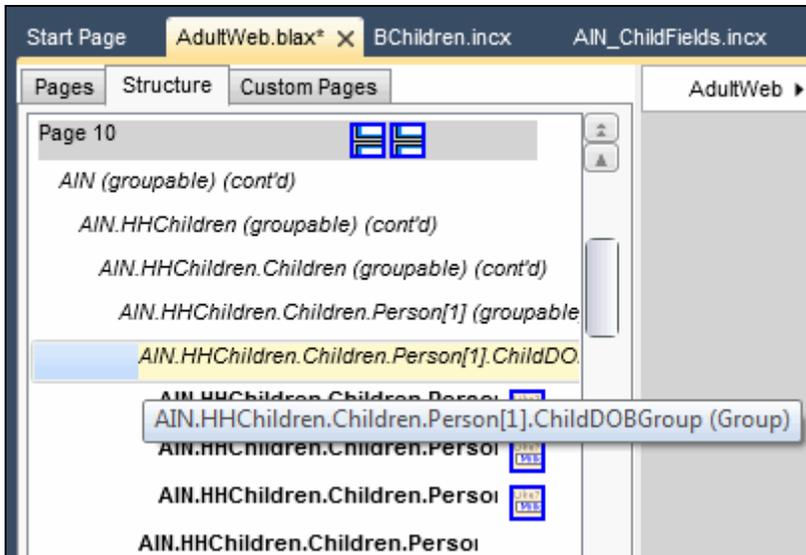


Figure 19

In order to get the fields to display horizontally we needed to apply a template to the group. We selected the 'Table: Abreast' from the Grouping templates in the ribbon which resulted in the three DOB fields being aligned horizontally, Figure 20.

The screenshot shows a web instrument form titled 'Adult Web Instrument (2.21.307.012)'. The form contains a question: 'What is 'gp9[pChildPointer]'s birthdate?'. Below the question are three dropdown menus labeled 'ENTER MONTH', 'ENTER DAY', and 'ENTER YEAR'. Each dropdown menu has a 'Select' button. Below the dropdowns is another question: 'So that would make 'gp9[pChildPointer]' *DispChildsCalcAge old?'. Below this question are four radio button options: 'Yes', 'No', 'Don't know', and 'Rather not answer'. At the bottom of the form are two buttons: 'Previous' and 'Next'.

Figure 20

Our final step was to replicate the DOB layout across the other DOB instances in the array. This was done by “Promoting” the DOB Group. We first used the ‘Promote instruction scope to block’ to promote the month, day and year fields. Next we promoted the Group using the ‘Promote instruction scope to block’ to replicate the layout across instances, Figure 21.

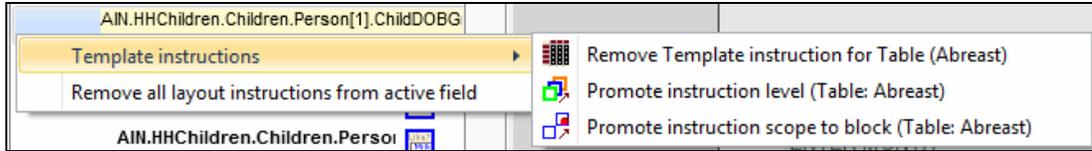


Figure 21

As another example of grouping we grouped alike questions to form a table. We grouped the questions similar to the date of birth questions but without a Rules section. Then using the ‘Table: OptionList’ from the Groupings on the Ribbon, the table was created, Figure 22.

Adult Web Instrument (2.21.307.012)

Has ^{*}ChildIdentifier lived in any of the following places for at least 2 weeks in a row in the last 12 months?

	Yes	No	Don't know	Rather not answer
a. Camp	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
b. Boarding School	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
c. Juvenile Detention Center	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
d. Mental Health Facility	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
e. Hospital or Medical Facility	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
f. Foster Care	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
g. Any other place?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

What other place has ^{*}ChildIdentifier lived for at least 2 weeks in a row in the last 12 months?

Please describe:

Previous
Next

Figure 22

D. Lessons Learned

1. We found that the Source Converter will automatically insert line breaks (<newline>) at the beginning of the line being new lined. However, if you need to add subsequent line breaks be sure to place them at the beginning of the new line. Adding the break at the end of the previous line instead of at the beginning of the new line will result in a blank space as the first character in the new line.
2. While testing our application we observed that questions displayed contrary to their conditional statements and that fill strings did not fill until we selected the next page. Answers to a question can influence the visibility or the value of subsequent questions. If these questions are on the same page it is desirable that the page gets refreshed as soon as a new value is entered. Therefore, the fields must be designated as a 'critical' field.

In our application we needed to designate the month of the child's date of birth to be critical so that the appropriate dropdown display of days matched the month selected. For instance April required a display of 30 days, January required 31 and February required 29. We found that we also needed to designate birth year to be critical so as to calculate the child's age and to trigger the leap year edit.

3. After converting the application we found that only one layout set was created which was named 'Interviewing1'. It uses the Browser Resource Set. You will have to add other Layout Sets for your target platforms which include the assignment of a Resource Set from the Resource Database. This is in contrast to when you code a new datamodel in Blaise 5 where a Layout Set is provided for each of the seven resources sets currently supported by Blaise 5.

V. DISTRIBUTING BLAISE 5 TO MOBILE PLATFORMS

There are two approaches within Blaise 5 that can be adopted to develop a survey for us on mobile devices. The first is to make the survey web interface mobile aware by using Adaptive or Responsive design techniques. The second is to build native mobile applications that are deployed to the device through the respective App store. The native applications are developed using tools and SDK's of the particular mobile platforms – iOS, Android or Windows 8. There are also mobile application development frameworks which allow for the application to be developed using one framework but be deployed to all the platforms (with some modifications). Statistics Netherlands is developing iOS and Android apps that are part of the product offering and will be supported and further developed as fully functioning interfaces.

Mobile websites are universally accessible, less expensive to develop and maintain and can be accessed by all mobile devices that have a browser. The issue of dealing with deployment to app store and certifications is avoided with a mobile web application.

There are lots of articles and books on designing web sites for mobile devices but our discussion will focus on taking our Blaise 5 instrument designed for a regular PC browser or a Windows desktop and with the addition of a few layout sets, make it compatible for mobile devices like an iPhone, iPad and Android Tablets.

While this paper focused on the conversion of a Blaise 4 application to Blaise 5 we took the added steps of installing it on a Blaise 5 web server along with Statistics Netherlands' Trade survey. We used an updated version of the Blaise app written by Statistics Netherlands for demonstration at FedCASIC.

As mentioned earlier in the Background section Blaise 5 uses Layout Sets (a combination of application code and Resource Set templates) to define the look and feel of a web application on a particular type of device. Blaise 5 provides some standard platform Resource Sets for:

- Windows Desktop
- Browser
- iPhone
- iPad Portrait
- iPad Landscape
- AndroidTabletPortrait
- AndroidTabletLandscape

The approach taken to make the Blaise 5 survey mobile aware is by using an Adaptive design principle – Which involves using different Resource Sets or UI layouts to suit different types of mobile device browsers. The application detects the type of device from which the request is coming from and uses the appropriate resource set. These designs fall into three basic styles, one for the desktop, one for mobile and one for tablets.

These styles are easily done in Blaise 5 and offer benefits such as:

- The user experience is the same across all devices
- Using the same application code across all platforms.
- Same links work across all devices
- Flexibility for future devices

We took our converted application Blaise 5 application and added layouts for an iPad, iPhone and a 7” Android tablet. When the Blaise 5 survey is accessed from any one of these devices, the application renders the correct type of layout to fit the screen size of the device. Similarly layouts for the Portrait and Landscape modes of a particular device like an iPad can also be added.

VI. CONCLUSIONS

Westat's process to convert a working Blaise 4 IS instrument to Blaise 5 spanned a number of Blaise 5 versions. As each version was released and bugs were fixed and templates were added or updated we would rerun the Source Converter. This conversion worked smoothly. We did not have to modify any Blaise 4 code prior to running through the Source Converter. However in post conversion, there was some coding to group fields to allow for custom displays. The standard templates provided in Blaise 5.0.1.553 covered the majority of layouts we required in our application.

As we moved past the actual conversion process we discovered a few not so obvious benefits of Blaise 5. We found that Blaise 5 provides greater flexibility to customize page layouts for our application. As we make changes to the layout, they are instantly applied in the Layout Designer which saves time and effort since we didn't have to run the application to verify the changes. Based on our experience we anticipate staffing requirements for converting Blaise 4 applications to Blaise 5 would be less than staffing needed to write new Blaise 5 code.

Though our target platform was the browser, we tested our application on other platforms including the iPad and iPhone. We were able to run our application on both the iPhone and iPad using the Statistics Netherlands Blaise app.

It is anticipated that future releases of Blaise 5 would include features such as the use of Alien Procedures, incorporating hyperlinks in question text and the ability to duplicate Layout Sets to customize for specific platforms.

Implementing Blaise 5 in a production environment

*Paul Segel, Mangal Subramanian, Ray Snowden, Richard Frey, Mike Florczyk
Westat, Inc.*

Blaise 5 is a fully re-engineered version of the Blaise product with new features and capabilities, a completely rewritten code base in Microsoft .Net, and a new deployment model and management tools. As such, one of the necessary steps for organizations preparing to adopt Blaise 5 is to begin to understand how to deploy, manage, and use Blaise 5 in a production environment particularly where the mechanisms and options in Blaise 5 are different from earlier versions of Blaise.

This paper discusses three aspects to the use of Blaise 5 in a production environment: 1) Deploying Blaise 5 in a server environment; 2) demonstrating how the Blaise 5 API can be used to support selected case management functions; and 3) considerations for conducting stress testing with Blaise 5.

This paper references our experiences working with Blaise 5.0.1.553. Development of the Blaise 5 product is ongoing and there will likely be changes to the software over the next several months. Any bugs or other similar issues are not addressed here because it is assumed they would be fixed in future releases.

1. Deploying Blaise 5

Blaise is a powerful and flexible system used for computer-assisted survey processing. With the introduction of Blaise 5 surveys can be deployed across multiple platforms including web/application servers, mobile devices (smart phones and tablets), desktops, and laptops. The surveys can be deployed as browser-based web-applications, native mobile apps, and as standalone applications. This paper focuses on the deployment of Blaise 5 in a server environment to support access via web browsers or rendered with the Blaise native mobile app.

The primary administrative tool used to deploy and manage Blaise 5 surveys is the Blaise Survey Manager. The Survey Manager is a browser-based tool that supports the following administrative functions:

- Configure and manage server parks
- Deploy surveys to server parks
- Create and manage end user accounts and permissions

1.1. Server parks

Blaise 5 has re-defined the packaging and deployment of the Blaise run time components in order to provide greater scalability, improved system continuity, and enhanced security. Blaise 5 distinguishes several types of logical server roles in the execution of a Blaise instrument. The server roles include:

- Web server – the server with which the end-user directly connects; process requests and send responses.
- Resource server – applies layout and formats the Blaise survey for different devices/modes.
- Data Entry server – handles pages and interprets/applies the Blaise rules.

- Data server– reads and writes data to a Blaise data source.

The collection of servers on which these server roles are installed to support a single Blaise 5 installation is called a server park. One or several server roles may be installed on a single physical or virtual server. Alternatively, for high volume applications and/or to provide redundancy to protect against component failure, any of the server roles, with the exception of the data server role, can be installed as a cluster of 2 or more servers.

The Blaise software is installed on any physical/virtual server which will be used to support a Blaise server role in a Server Park. The Survey Manager provides functions to assign and configure server roles to physical/virtual servers and to manage the distribution at run-time of Blaise-related processing to the server roles across the Server Park.

Under Survey Manager, the ip address and port number over which Blaise 5 communicates is specified for each server role defined. When installing Blaise 5, it is important to work with the network administrators and systems security staff within your organization to make sure that the ports selected for use are consistent with your organization's policies and that the necessary network access has been provided, e.g., through firewall settings.

Blaise 5 server roles can be installed either on physical or virtual servers. In our test installation, we used a Server Park with one virtual server hosted on VMWare running Windows 2008 R2, service pack 1 and used SQL Server 2008 on a separate virtual server for database support. The use of virtualization to support Blaise 5 provides additional flexibility and efficiencies over a physical server installation including:

- New servers can be quickly deployed on existing hardware.
- Additional resources (CPU, memory, disk space) can quickly be added to an existing server with little disruption.
- Virtual servers can be moved from one virtual host (physical server) to another very quickly in the event of hardware problems, maintenance, etc.
- Copies/clones of virtual servers can be created and saved for system recovery.

Blaise 5 no longer supports the use of the proprietary BDB file to store data but rather uses a DBMS for database services, such as SQL Server, or the default SQLite which is installed with Blaise 5. In addition to the Blaise Server Park described above, a Blaise 5 installation must also include a server running a supported DBMS.

1.2. Deploying surveys

When a survey is ready for deployment a survey installation package is created. The package is created in the Blaise Control Centre where source code is converted into executable code and other survey files are packaged together. The package contains the questionnaire's binaries and supporting files, together with layout and runtime settings.

The Survey Manager is used to install and manage survey packages on Server Parks providing such functions as activation, deactivation, previewing and removing surveys. You can also view details about a survey such as the data source, the active status of the survey, and the installation date. All these functions are handled by an administrator.

In software development, it is common practice to deploy an application through a series of environments including development, testing, acceptance and production. In Blaise 4, moving a Blaise instrument to different environments required manual reconfiguration of the Blaise OLEDB Interface (boi) files to specify the appropriate database. In Blaise 5 the Control Centre allows you to create different Blaise Data Interface (bdix) files, and assign a role to each file representing development, testing, acceptance or production. When the corresponding role is set in the creation of a deployment package through Control Center, the bdix file with the matching role will be used.

The Blaise 5 Server Manager also supports the deployment of a survey package to different testing environments. Server Parks can be used to support the different environments and make the survey available to the appropriate testers whether they are testing on a Windows desktop, a browser, a mobile device or a tablet. The bdix files can be used as described above to help allow packages to be easily deployed to different environments.

1.3 User Management

The Server Manager provides functions to create user accounts that are used to authenticate access to Survey Manager and assign permissions to access various functions. A built-in account called Root is the default Blaise 5 administrator account and is the only account that can create and manage users.

With the User Management function you can control which server a user can access and what type of changes these users can make to the servers and the installed surveys. Each person can have a separate user account with unique settings and preferences. The settings can be as detailed as allowing a user access to just one survey or as broad as allowing a user control over all server parks.

2. Security

Blaise 5 production instruments will need to comply with an organization's security policies. Rhoads and Snowden in their 2010 IBUC paper quote the FISMA definition of information security as:

Protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide the following general safeguards:

- Confidentiality – restricting access to information to authorized users only. This is a central security concern in survey research projects where respondent data often includes highly confidential personal information and PII. Unauthorized disclosure of confidential data is everybody's top concern.
- Integrity - guarding against the unauthorized modification or destruction of information due to either accidental or malicious actions. This is another important aspect of security in survey research projects where survey data is generally time-consuming and expensive to collect and the credibility of analytic findings depends on a high degree of confidence in the quality of the underlying survey data.
- Availability - ensuring timely and reliable access to and use of applications and information. Consistent and reliable access to survey systems in CAI projects can be particularly critical since response rates are extremely important. Once a respondent is contacted and ready to provide information, the CAI systems must work.

We will consider various aspects of the Blaise 5 system with respect to those safeguards.

2.1. Servers

We installed Blaise 5 in our secure data center on virtualized Windows 2008 R2, SP1 servers created using a secure server configuration template, regularly scanned for security vulnerabilities, and maintained with current security patches. Virtual images, backed up regularly, are maintained on dual SAN storage to provide redundancy for disaster recovery. Database servers are maintained in a separate security zone with network isolation provided by a firewall. This isolates database servers from direct access by end-users over the Internet. Blaise 5 has been compatible with our standard security templates, which maintain secure policies. We will discuss sizing and performance measurement in a later section of this paper.

2.2. Server Parks and firewalls

Blaise 5 Server Parks provide a ready context for implementing network isolation, continuity, and recovery. To scale out capacity, multiple servers may be employed to prevent overloading the Web and Data Entry servers (running the DataEntry and Resource services), with load balancing provided by the Management server, and accessing data through a single Data Server. Actual production databases, accessed through the Data server may be isolated by a firewall with normal port access for the database.

The port to the Management server is configurable when the server park is deployed. The Management server supports both internal and external functions. Internally it supports the configuration of server roles, the deployment of instruments, and user management. Externally, it can be used to list available instruments to a user. At this time, the Blaise team is considering supporting those functions over different ports allow tighter security for external users.

2.3. Connected data collection

Blaise 5, especially with its support for mobile devices, offers new models for configuring the relationship between central servers and remote data collection devices. These devices include remote web browsers, laptops, tablets, and mobile phones. Many of the issues associated with remote data collection devices are familiar and have been part of previous Blaise versions. We will try to describe some issues related to Blaise 5 specifically and the new devices that it supports.

Blaise 5 can be installed and operated to collect data in two different configurations - connected data collection and disconnected data collection. The two configurations share a number of common security considerations. With its tighter connection to the server, we will start with connected data collection.

In a connected configuration a number of security issues are handled by the central servers including:

- Authentication and authorization – these gateway functions might be performed external to the Blaise 5 instrument as part of the mechanism associated with launching the instrument or accessing the remote device. Within a Blaise 5 instrument, text can be hidden to shield password entry, and the proposed alien procedure could be used to encrypt and compare passwords. A second authentication factor is still probably best handled outside the Blaise 5 instrument
- Security of the configuration information - Blaise 5 encrypts sensitive elements of the configuration such as any connection strings in the data interface file (.bdix). Further the Control Centre can enforce password protection on its decryption.
- Security of data during collection – in the connected collection mode, data responses are transmitted from the device to the central servers. For web surveys, SSL encryption can be enforced to Blaise 5 web sites.
- Security of data collection at rest – with the data residing on central servers, the security of the data becomes an issue of access control to the server park servers and any attached database servers and possibly data encryption.

Security considerations for the client device in a connected configuration include:

- Application and security configuration on the device – organizations need to configure the remote device to comply with any required security and operational standards. As Blaise 5 nears release we will be testing compatibility with security standards such as the NIST United States Government Configuration Baseline (USGCB) for applicable operating systems, and with accessibility standards, such as Section 508.
- Application, security, and operating system updates - although data are not stored on the remote device, it is a gateway to the central servers. It is important to maintain the security features of the device with current patches.

With Blaise 5, mobile devices, such as iOS and Android phones and tablets, are running native applications to interface with the central servers. Updates to these applications need to be evaluated for compatibility, risk, and benefit. Procedures for version upgrades and enforcement will be required.

- Similarly, an organization’s applications, possibly from a private application store, such as through the iOS Developer Enterprise Program, and instruments need to be maintained and configured in a secure manner.
- Loss or damage to the device – Blaise 5 introduces new device types, but otherwise an organization needs to maintain its current procedures for reporting and addressing the loss or damage to the remote device.

2.4. Disconnected data collection

Data collection on disconnected devices adds additional considerations and new complications to the issues associated with connected devices. These include:

- Protecting data at rest – strategies that have been developed for platforms used with earlier Blaise versions, will need to be re-evaluated with Blaise 5 and extended to new devices and storage platforms. For example, methods for maintaining encrypted data may need to migrate to new storage and device platforms.
- Protecting data transmission – similar re-tooling may be required for maintaining secure transmission as data collected on the remote device is transmitted to central servers, as is normally required.
- Loss or damage to the device – with data stored on the remote device, even in a secure form, it may be more important to consider tracking and data wiping capabilities for disconnected remote devices.

3. Some case management functions with API

The Blaise 5 API can provide the tools to perform some case management functions. We will discuss an environment where the Blaise 5 API provides some functions in a component (it could be transformed into a Web Service) that can be used by a case management application (or web site). For purposes illustration we will assume that external data resides in SQL Server, and we will deal with database tables as the primary method of sending and receiving external data.

The sample component, BlaiseAgent, provides a wrapper around the Blaise 5 API, and creates functions for some common data management functions. These case management functions might include:

- Preloading data for a case
- Returning case-level status information
- Extracting case results

The Blaise 5 API is divided into four components, which current documentation describes as:

- Meta API for read-only access to metadata information
- DataRecord API for data validation, checking and routing
- DataLink API for reading/writing of stored data
- SessionData API for read-only access to session data

Because of the nature of our sample tasks, our sample component will most heavily focus on the Meta, DataRecord, and DataLink APIs . In discussing the code, we will frequently qualify an object with the name of the API to which it belongs.

The BlaiseAgent component may return Blaise 5 objects ,so a calling application may require references to Blaise 5 components including Statneth.Blaise.Data.SQLite and possibly any other Blaise 5 API that contains the returned object.In this discussion, the BlaiseAgent component contains a single class Agent. Agent is initialized with the file name of the data model (bmix) and the database (bdix) and owns the corresponding objects:

- MetaAPI Datamodel object named: dm
- DataLinkAPI IDataLink object named: dl

We will look at several public methods the Agent exposes. For these examples we are ignoring block structures in the data model. (They would be handled by recursing through the fields in the block.)

3.1. Preload data

This example code writes records to a Blaise data base from a SQL Server data table.

```
1 public void Load(DataTable dt)
2 {
3     ICollection<DataRecordAPI.IDataRecord> cDR = new List<DataRecordAPI.IDataRecord>();
4     foreach (DataRow r in dt.Rows)
5     {
6         DataRecordAPI.IDataRecord dr_target = DataRecordAPI.DataRecordManager.GetDataRecord(dm);
7         foreach (DataColumn c in r.Table.Columns)
8         {
9             string fieldName = c.ColumnName;
10            string fieldValue = r[c.ColumnName.ToString()].ToString();
11            SetDataValue(dr_target, fieldName, fieldValue);
12        }
13        cDR.Add(dr_target);
14    }
15    IEnumerable<DataRecordAPI.IDataRecord> eDR = cDR;
16    DataLinkAPI.IDataSet ds = DataLinkAPI.DataLinkManager.GetDataSet(eDR);
17    dl.Write(ds);
18 }
```

The Load function builds a list of DataRecordAPI DataRecords and adds them to the DataLinkAPI dataset.

- Line 6 creates a DataRecordAPI DataRecord to be filled and added to the list.
- Line 11 uses an internal function SetDataValue to set the field values on the DataRecord. We will look at that function below.
- Line 16 builds a DataLinkAPI DataSet
- Line 17 uses the BlaiseAgent's DataLinkAPI DataLink and writes that DataLinkAPI DataSet to the Blaise 5 database

This example ignores issues like insuring primary key uniqueness and other errors. But, we will take a quick look at the SetDataValue function to discuss data types. The function receives a value as a string and converts it to the appropriate Blaise data type or non-response attribute for the field.

```

1 private void SetDataValue(DataRecordAPI.IDataRecord dr, string fieldName, string fieldValue)
2 {
3     if (!string.IsNullOrEmpty(fieldName) && !string.IsNullOrEmpty(fieldValue))
4     {
5         DataRecordAPI.IField fld = dr.GetField(fieldName);
6         if (fieldValue == "DK")
7         {
8             fld.DataValue.SpecialAnswer = MetaAPI.Constants.SpecialAnswerNames.DontKnow;
9         }
10        else
11        {
12            DataRecordAPI.IDataValue dv;
13            if (fld.Structure == DataRecordAPI.FieldStructure.Data)
14            {
15                switch (fld.DataValue.DataType)
16                {
17                    case StatNeth.Blaise.API.DataRecord.DataType.Classification:
18                        if (fld.Definition.Type.IsValid(fieldValue))
19                        {
20                            fld.DataValue.Assign(fieldValue);
21                        }
22                        break;
23                    case StatNeth.Blaise.API.DataRecord.DataType.Date:
24                        dv = fld.DataValue;
25                        dv.DateValue = DateTime.Parse(fieldValue);
26                        fld.DataValue.Assign(dv);
27                        break;
28                    case StatNeth.Blaise.API.DataRecord.DataType.Enumeration:
29                        fld.DataValue.EnumerationValue = int.Parse(fieldValue);
30                        break;
31                    case StatNeth.Blaise.API.DataRecord.DataType.Integer:
32                        fld.DataValue.Assign(fieldValue);
33                        break;
34                }
35            }
36            ...
37        }
38    }
39 }

```

We can look at a few special cases. (We deleted from the listing more examples of converting some of the Blaise data types that were further examples of setting the field value using the Parse method to convert the string to a corresponding system type.)

- Line 5 creates the DataRecordAPI Field object that will receive the appropriate data type and value
- Lines 6 and 8 test for a received non-response code, e.g. "DK", and set the appropriate Blaise 5 DataRecordAPI Field DataValue.SpecialAnswer.
- Line 12 create a local DataRecordAPI DataValue to hold the type and value
- Line 13 checks that the DataRecordAPI field is a simple field and not block or array, by verifying that the DataRecordAPI Field.Structure is DataRecordAPI.FieldStructure.Data

The following lines perform type conversions by inspecting the DataRecordAPI Field.DataValue.DataType, performing any necessary conversions from the string value, and setting the value with a call to DataRecordAPI Field.DataValue.Assign().

A few might be worth mentioning:

- Line 18 illustrates one of the overloads of the `DataRecordAPI Field's Definition.Type.IsValid()` method. This example tests a string as a valid classification value. Other overloads of the method validate integers, real, datetime, and arrays of integers for a set type item.
- Line 28 sets the field's `DataValue.EnumerationValue` to the integer corresponding to that enumerated value

Some things we learned:

- You may need to match the CPU(X86 or x64) to match the version of `StatNeth.Blaise.Data.SQLite` that Blaise 5 installs

3.2. Display status information – selected cases

The example code populates a SQL data table from the Blaise dataset. Again, for simplicity, it assumes that the table's column names are the same as the Blaise dataset's field names. And, in this case, it displays the status information for a set of cases specified by an incoming SQL data table and the Blaise database's primary key name. (For simplicity, we assume a single key field.)

```
1 public System.Data.DataTable showAllStatus(DataTable dtable, string BlaiseKeyName)
2 {
3     foreach (DataRow dRow in dtable.Rows){
4         string BlaiseKeyValue = dRow.Field<string>(BlaiseKeyName);
5         // Retrieve record based on Primary key
6         DataRecordAPI.IDataRecord dr = GetDataRecord(dm, dl, BlaiseKeyValue);
7         if (dr != null)
8             {
9                 foreach (DataColumn cx in dRow.Table.Columns)
10                    {
11                        string fn = cx.ColumnName;
12                        dRow.SetField(fn, ReadDataValue(dr, fn));
13                    }
14            }
15     }
16     return dtable;
17 }
```

This example loops through the incoming SQL data table, retrieving the Blaise `DataRecordAPI DataRecord` by its key value, Line 6 - `GetDataRecord()`. It then populates a SQL data row with the Blaise field values, converted from the Blaise data type to string, Line 12 – `ReadDataValue()`. It omits error handling for simplicity.

Looking at GetDataRecord:

```
1 private DataRecordAPI.IDataRecord GetDataRecord(MetaAPI.IDatamodel dm, DataLinkAPI.IDataLink dl,  
2 string primaryKey)  
3 {  
4     DataRecordAPI.IDataRecord dr;  
5  
6     DataRecordAPI.IKey kv = DataRecordAPI.DataRecordManager.GetKey(dm, "PRIMARY");  
7     kv.Fields[0].DataValue.StringValue = primaryKey.ToString();  
8     dr = dl.ReadRecord(kv);  
9     return dr;  
10 }
```

- Line 6 uses the DataRecordAPI DataRecordManager.GetKey method to return Key object from the primary key.
- Line 7 populates the Key object with the requested key value
- Line 8 uses the DataLinkAPI DataLink.ReadRecord method to return the DataRecordAPI DataRecord object with the requested key value.

Once the code has the retrieved record, the BlaiseAgent's ReadDataValue method converts it to a string for the SQL DataTable being built as the return value.

```
1 private string ReadDataValue(DataRecordAPI.IDataRecord dr, string fieldName)  
2 {  
3     string retValue = "";  
4     if (!string.IsNullOrEmpty(fieldName))  
5     {  
6         DataRecordAPI.IField fld = dr.GetField(fieldName);  
7  
8         switch (fld.DataValue.AnswerStatus)  
9         {  
10            case DataRecordAPI.AnswerStatus.SpecialAnswer:  
11                retValue = fld.DataValue.SpecialAnswer;  
12                break;  
13            case DataRecordAPI.AnswerStatus.Response:  
14                if (fld.Structure == DataRecordAPI.FieldStructure.Data)  
15                {  
16                    switch (fld.DataValue.DataType)  
17                    {  
18                        case StatNeth.Blaise.API.DataRecord.DataType.Classification:  
19                            //  
20                            break;  
21                        case StatNeth.Blaise.API.DataRecord.DataType.Date:  
22                            retValue = fld.DataValue.DateValue.Value.ToString();  
23                            break;  
24                        case StatNeth.Blaise.API.DataRecord.DataType.Enumeration:  
25                            retValue = fld.DataValue.EnumerationValue > 0 ?  
26                                fld.Definition.Type.Categories.GetItem(fld.DataValue.EnumerationValue).Name : null;  
27                            break;  
28                        case StatNeth.Blaise.API.DataRecord.DataType.Real:  
29                            retValue = fld.DataValue.RealValue.ToString();  
30                            break;  
31                        case StatNeth.Blaise.API.DataRecord.DataType.Set:
```

```

31     string value = "";
32     int idx;
33     foreach (var r in fld.DataValue.DynamicValue)
34     {
35         idx = r;
36         value = fld.Definition.Type.MemberType.Categories.GetItem(idx).Name;
37         retVal = string.IsNullOrEmpty(retVal) ? value : retVal + "," + value;
38     }
39     break;
...

```

- Line 6 Like SetDataValue() above, ReadDataValue() operates on a DataRecordAPI Field object. It is retrieved from the incoming DataRecordAPI DataRecord by field name with the GetField method.
- Line 10 initiates a check on the DataRecordAPI Field. DataValue.AnswerStatus to distinguish non-response (DataRecordAPI.AnswerStatus.SpecialAnswer) from a response (DataRecordAPI.AnswerStatus.Response)
- Having determined that the field has its Structure property as DataRecordAPI.FieldStructure.Data, not a block or array (Line 14), the function converts the field depending on its DataValue.DataType.
- In Line 24, an enumerated field (DataType.Enumeration) is converted to its more descriptive Category name from its integer data value using Field.Definition.Type.Categories.GetItem(fld.DataValue.EnumerationValue).Name
- Starting in Line 30, a set field is converted to a comma-separated list of values. It loops through the field's DataValue.DynamicValue, retrieving the integer value, and converts it to its category name.

3.3. Display status information – all cases

To display status for all cases, we can read the entire data set and loop through its records.

To read the entire dataset, or a portion of a dataset with a primary key:

```

1 private DataLinkAPI.IDataSet getDataSet(string startKey, int recordLimit, bool includePrimaryKey)
2 {
3     DataLinkAPI.IDataSet ds;
4     string keyName = "";
5     if (dm.Keys.Count >= 1)
6     {
7         DataRecordAPI.IKey key = GetPrimaryKey(dm, startKey);
8         // Read a set of records:
9         ds = dl.Read(key, DataLinkAPI.ReadOrder.Ascending, recordLimit, includePrimaryKey);
10    }
11    else
12    {
13        // Read all records
14        ds = dl.Read("");
15    }
16    return ds;
17 }

```

- Lines 9 and 14 illustrate two overloads of the DataLinkAPI DataLink object's Read method.

To loop through the resulting DataLinkAPI Dataset, retrieving the DataRecordAPI DataRecord:

```
1 DataLinkAPI.IDataSet ds = getDataSet("", -1, true);
2   if (ds != null && ds.RecordCount > 0)
3     {
4       while (!ds.EndOfSet)
5         {
6           DataRecordAPI.IDataRecord dr = ds.ActiveRecord;
7
8           // ... process the DataRecord
9           ds.MoveNext();
10        }
11    }
```

4. Stress testing

As part of production deployment it is often useful to determine the capacity of the servers to handle a particular instrument and the anticipated user load. Factors affecting capacity include:

- Number of users and expected distribution of users over time
- Size of the data model
- Size and configuration of the server hardware

This section discusses an approach to testing the capacity of a web deployment of a Blaise 5 instrument, using a testing tool that simulates multiple users interacting with the instrument on the anticipated server configuration. Given the variation in instruments, user behavior, and server configuration, it is not a general prediction of how Blaise 5 instruments will perform, but describes some of the considerations and approaches for stress testing a Blaise 5 instrument in a server environment.

4.1. Stress Test Design Considerations

The purpose of stress testing is generally to verify that a given server park configuration will perform adequately at times when the maximum number of concurrent users are active. In a web-based survey, it is very difficult to predict what the maximum number of users will actually be as web surveys are normally self-administered and users take the survey at their convenience. There is also a distinction between the number of users that may be concurrently active in the survey and the processing load of this usage on the server park as a considerable amount of connected time for a user is spent reading questions and typing responses. Although some resources are used to support any connected user, there is a spike in resource utilization once a user submits a page to the server when their data is being processed. Because of this, and in order to not continually over deploy server resources, it is important to configure the test to simulate the expected maximum number of connected users and model the expected profile of their behavior, in terms of wait time between pages, etc.

The stress testing will then help determine what an adequate server park configuration must be to support this expected maximum load and also measure the rate of response degradation as usage is increased beyond the maximum. This information can be used to balance the costs of increased server capacity with the risks of unexpected spikes in concurrent usage.

We used Microsoft's Visual Studio Test Manager 2012, but other tools have similar consideration as those discussed here. The VS2012 Test manager provides the ability to set up a Web Test script by recording a user session against a Blaise 5 web survey. The Web Test script can then be associated with a Load Test, which allows one to specify various parameters like no. of users, test run time, think time, type of load (constant, stepped or simultaneous). Additional scenarios can be added to set different stress levels for the application. A test warm up time can be set to account for any application start up delays.

In our stress testing exercises, we held our server park configuration fixed at one virtual Blaise 5 server plus a SQL Server database server and increased the number of concurrent users from 50 to 200 in 50 user increments to observe the average response time of the survey as the load increased. Except in rare circumstances, perhaps a scripted scheduled training session, it would be unlikely that all 200 users would simultaneously submit a page for processing. And, we expect that users will spend varying amounts of time on each page. We configured the test tool to use a think time at each page randomly distributed around the time we spent on each screen when the script was recorded. To obtain a more realistic arrival time for the pages, and avoid an initial spike of unrealistic simultaneous access, we specified the rate at which users would start the test. Once the test was started and after a short delay we arrived at the specified maximum number of users and maintained that number throughout the test.

We selected a test duration that we thought would allow the users to be on reasonable distribution of pages and locations in the instrument. A longer test duration also allowed us to monitor server resources to see if there was increasing demand on memory (indicating possible leakage) or other resources such as the ASP.Net worker process and to monitor the frequency of application problems, such as page failure, database contention, web exceptions and instrument failures.

In working through various preliminary test results, we varied a number of configuration settings on our one server Server Park such as the number of processors and the amount of memory to see the effect on test results. These configuration changes were somewhat easier to make since the test machine was a virtual server and required no physical hardware change.

We also found that the configuration of the client machines executing the test scripts was an important component of the test design. As these client machines are simulating the execution of a browser-based application for multiple users, it is important that limitations in the configuration of the client test machines do not create artificial bottlenecks that may skew the test results. We found that it was important to add new client test machines for every 75 – 100 simulated users, e.g., simulating 200 users might best be done by using 3 test machines each simulating 70 users. These ratios will be different based on the testing tool and the configuration of the client machines but is an important point to consider.

4.2. Creation of the test script

The testing tool we used provides a feature that records all of the key strokes and timing of a user session through a Blaise 5 instrument. This recorded session can then be “played back” by the test tool to simulate multiple users completing the survey with the same responses. When you record a test script for an instrument, one approach is to provide responses that reflect an “average” path through the instrument, e.g., when it contains a roster that would require looping, select some middle-of-the-road roster sizes. Alternatively, the script can be recorded using responses that reflect the most complex and resource intensive path through the instrument to reflect higher usage profiles.

Typically, each invocation of the instrument requires a case id. Our tool allowed us to build a database of case ids and configure the script to use a unique id for each test cycle. We limited the number of case ids to test a mixture of new and previously opened cases.

4.3. Metrics

Stress testing is normally an iterative process where test scenarios are run and server configurations are changed until an adequate configuration is achieved. It is important when running stress tests to define and record various metrics to establish criteria for a successful test. Measures that we used to evaluate the success of the test included average response time (the time a user waited for a page to be ready for input) and the 95th percentile of response time on any page.

We also monitored several performance counters on the server, some associated with the server resources and some with the ASP.Net framework. These measures can be used to identify resource bottlenecks so that adjustments can be made to the configuration of the Server Park if necessary. Items we tracked included:

ASP.Net Counters;

- Application Restarts
- Requests Queued
- Worker Process Restarts

- Requests per second
- Errors Total

Web Server Processor;

- % CPU Utilization
- Memory Utilization

Database Server;

- % CPU Utilization
- No. of connections

In addition to these performance counters, following key metrics captured by the test tool were used to determine the health of the application under various loads.

- Tests/Sec
- Tests Failed
- Avg. Test Time (sec)
- Pages/Sec
- Avg. Page Time (sec)
- Requests/Sec
- Requests Failed
- Avg. Response Time (sec)

In addition to the automated metric collection, we also had a few live users also navigate the instrument while the load tests were running, noting any unusual delays or problems, and to compare the metrics with the actual user experience, e.g. the average page time recorded by the tool was compared with the average time it took for a page to load during a simultaneous manual test by a user.

4.4. Scaling Blaise 5 server capacity

Blaise 5 offers a number of strategies to scale a Blaise 5 server configuration if it is determined that additional resources are required. These include:

- The resources for any physical/virtual server in a Server Park can be increased to provide additional resources.
- The various server roles can be deployed to different physical/virtual servers.
- All of the server roles except the data server can be installed on 2 or more servers. Blaise 5 will manage the distribution of work across these servers.

4.5. Thoughts on Stress Testing

Stress testing is an important step in preparing for a production implementation of Blaise 5 particularly if a large number of respondents is expected or if a complex survey is used. In this section we discussed a number of considerations and options for conducting stress testing based on our preliminary work in this area.

We plan to continue our work in stress testing Blaise 5 in a server environment. It is our goal to develop some baseline stress testing results that can be used to help organizations begin to project the types of usage loads various server parks configurations can support.

Conclusion

Blaise 5, especially with its new modes of data collection, will require new considerations for configuration, deployment, security, and maintenance. With its unified design across modes of data collection many of these considerations are also unified and may be dealt with a common approach.

Rhoads, Mike and Ray Snowden. "Security Considerations in Blaise Environments: Options and Solutions". *Proceedings of the 12th International Blaise Users Conference*. September 2010. <<http://www.blaiseusers.org/2010/papers/7e.pdf>> (August 8, 2013)

Testing a Complex Blaise CAPI Instrument

John Fitzgerald, Central Statistics Office, Ireland

1) Introduction

The Central Statistics Office (CSO) currently uses CAPI interviewing for all of our Household Surveys. We have been developing Blaise Questionnaires and customising Blaise tools since 1997. Our Survey Instruments are generally used for longitudinal Surveys such as the Quarterly National Household Survey (QNHS) which has a 13 week development cycle each quarter and the Survey on Income and Living Conditions (SILC) which is developed for bi-annual release. We also provide Questionnaires for 'one off' Surveys such as the Programme for the International Assessment of Adult Competencies (PIAAC) in 2011 and the Household Finance and Consumption Survey (HFCS) in 2013.

This paper will demonstrate new testing practices and procedures we implemented during the development of the HFCS Questionnaire in 2012/2013. Our goals were to improve the efficiency and flexibility of our Questionnaire testing, while also improving the quality of the instrument and to ensure we were delivering the Questionnaire as specified.

I will present an overview of the testing throughout the development lifecycle, but the emphasis will be on Independent testing performed by the Blaise team on the completed versions of the Survey Instrument.

2) Planning & Implementation

The HFCS Instrument was one of the largest and most complex we have yet developed. A development period of 130 working days was originally estimated. The final Survey instrument that was developed contained over 850 variables. There was also in excess of 370 checks and signals. It contained a large number of text fills to cater for direct or proxy interviews and previous responses. The fills made the questions more relevant for the respondent but were time consuming to code and test.

Table 1 and Table 2 illustrates the size of the instrument and the challenges we faced in planning for the testing. It was important to achieve maximum test coverage even though we had limited resources and a limited amount of time.

Table 1 Technical description of the model HFCS2013 - Data Types

Data Fields types in DB	Total
Integer	1729
Real	106
Enumerated	3344
Set	114
Classification	0
Datatype	73
Timetype	48
String	922
Open	0

Table 2 Technical description of the model HFCS2013 - Fields

Field types	Total
Number of uniquely defined fields*1	918
Number of elementary fields*2	857
Number of defined data fields*3	6336
Number of defined block fields*4	61
Number of defined blocks	66
Number of embedded blocks	0
Number of block instances	487
Number of key fields	5
Number of defined answer categories	132
Total length of string fields	40010
Total length of open fields	0
Total length of field texts	99580
Total length of value texts	17539
Number of stored signals and checks	10127
Total number of signals and checks	10127
*1) All the fields defined in the FIELDS section *2) All the fields defined in the FIELDS section wich are not of type BLOCK *3) Number of fields in the data files (an array counts for more than one) *4) Number of fields of type block	

As this was a new project, it was decided to implement a more structured approach to software testing than we had used in the past. We devised metrics that would provide a quantitative measure of the testing. Realistic targets were set in order to achieve a high level of testing coverage within the resources and time constraints. Testing was scoped and all stakeholders were assigned responsibilities for each stage of the development life-cycle.

Table 3 illustrates how we customised Software engineering test levels into our own Bliase development life-cycle.

Table 3 - Test levels in the development lifecycle

Test level	Who	Test Type deployed	Techniques\methods
Requirements Testing	Development Manager in collaboration with programmer and author of requirements	Static testing (doesn't require code)	Reviews Walk-through of documentation flowcharts
Component Testing	Blaise Programmer tests their code.	Functional Testing (white box) Structural testing	Boundary Testing, Equivalence Partitioning, Statement Testing, Decision tables, Question text, variables, routing. We implemented peer-reviews and code reviews
Independent component Testing	Developer other than programmer	Functional Testing (black box)	Boundary Testing, Equivalence Partitioning, Decision tables, Question text, variables, routing, Computations. Informal Reviews
Integration testing	Developers - overseen and documented by team leader	Non-Functional testing (Black box)	Interaction between components of the system. Hardware v software tests Software design Use case testing
Acceptance testing	Business area and/or Interviewers	Non-Functional (Black box)	Verification of delivery on specifications

2.1 Test Levels

Throughout the development process, testing played a significant role. It was important to ensure that Questionnaire was being developed correctly and that it would meet the Business needs. The following is a description of the customised test levels that were used throughout the HFCS development lifecycle.

2.1.1 Requirements testing

The Questionnaire specifications and other work products around the requirements would become the baseline for the new formal testing process. Static testing techniques were employed to find errors or defects in the specifications. Static techniques are tests performed without executing any code. This approach finds errors and defects earlier in the life cycle of the project and this makes corrections easier and cheaper than finding those same defects later on in the project.

The development manager reviewed the specifications to find and remove errors and ambiguities in the document before they can become part of the executable code. This review process included informal and formal meetings with experienced programmers and the documents author(s) and other relevant stakeholders. The review objectives included: Finding defects, gaining understanding, generating discussion & forming consensus on the requirements.

2.1.2 Component testing

The Development Manager in collaboration with Senior programmers split the requirements document into logical blocks for coding [see figure 1]. This process was documented and the requirements were split and assigned to programmers.

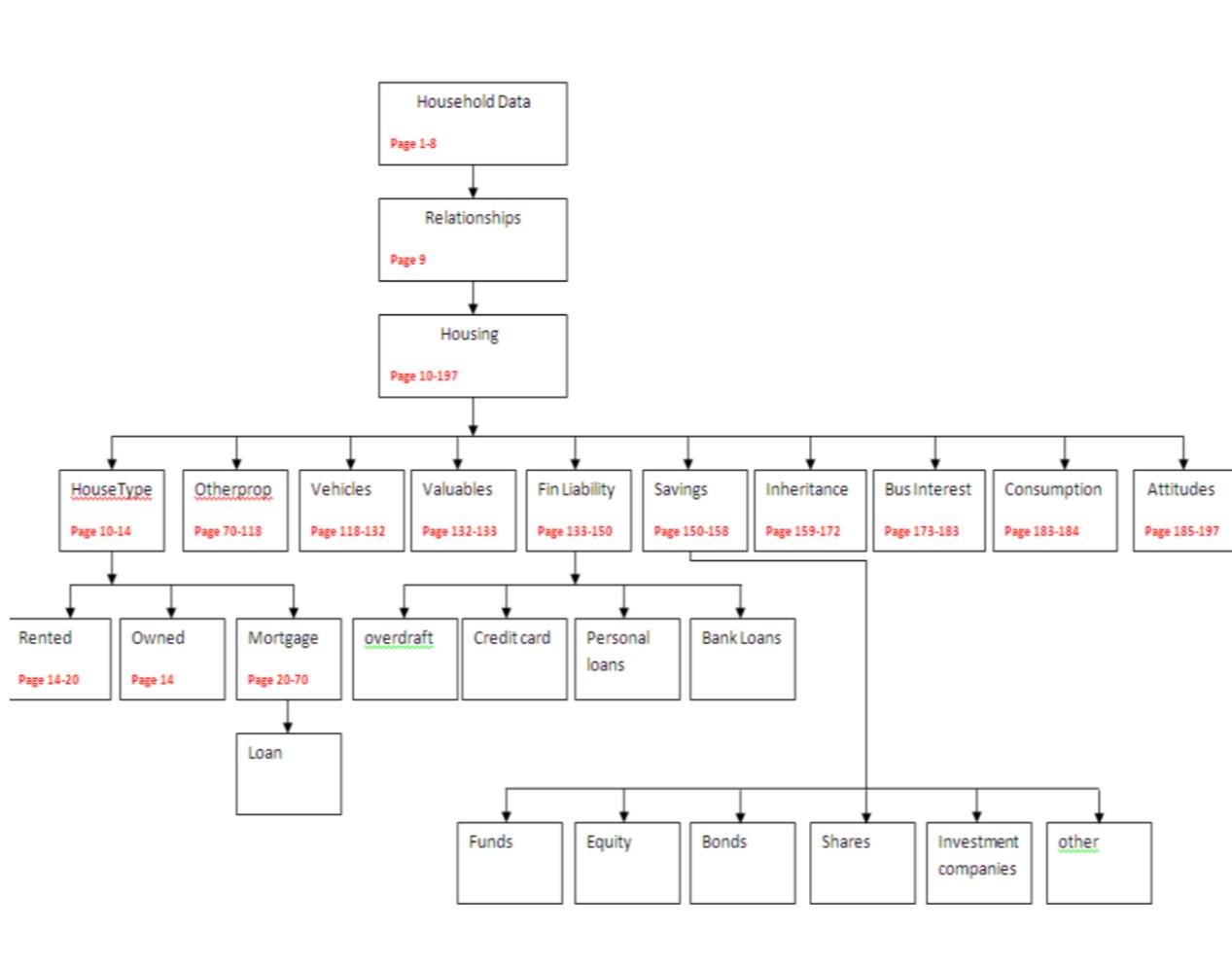
The programmers coded their assigned blocks from the specification document. Any issues or clarifications were logged and followed up by the Development manager at regular development meetings. The programmers performed component tests on their own code to ensure that the specifications were being fully met. Collaboration was encouraged and we also held informal code reviews attended by the code author, his/her peers and Manager.

Code reviews proved a very effective tool for the following reasons:

- Ensure Blaise standards have been implemented
- Best coding approach has been used
- Removal of bugs
- Forming consensus on requirements
- As a knowledge sharing exercise

It was important to establish with everyone involved in the review process that the emphasis should be on learning and improvement. Defects or deficiencies should be welcomed and suggestions or solutions should be expressed objectively.

Figure 1 Household level Requirements broken into manageable Blocks



2.1.3 Independent Component testing

Once a version of the Questionnaire was compiled and ready for Independent testing we could begin to complete the logs that were written from the specifications. The approach we adopted was 'Question-by-Question' testing where each variable in the specification document would be tested against the fully coded Questionnaire. While there was an awareness of the big commitment of time and resources to this approach, the trade-off of a high quality Questionnaire would be worth the effort.

Based on our own experience and also using risk analysis techniques we prioritised logs [or subjects] where potential bugs were more obvious. For example, the routing and computations on the 'Mortgage & Loans' block was very complex, therefore there was a high probability of bugs which could potentially have a high negative impact on the functioning of the Questionnaire.

To manage testing, the questionnaire was divided into Household and Personal sections. A test log was created for each block and for each of the following test approaches:

- Routing
- Variable Ranges
- Fills/Inserts & Question text
- Errors/Signals
- Overall Appearance
- Computations/Errors/Signals/Don't Know/Refusals

Every Block had potentially 6 test logs to be completed by an independent tester. We used a Lotus Notes repository to manage all testing documentation A table was devised for each test topic. A link was provided to each log template [Figure 2] which would be assigned or signed out by an Independent tester. Any bugs or issues raised by the tests were written into an error log [Figure 4] which was fed back to the developers.

Figure 2: Routing Tests of Household Blocks

Route Tests						
Household Level	Test Log Template	Completed Test Log	Error Log	Specification Document Page Number(s)	Tester	Changes Made?
Household Data				1 > 8	John	Denis
Relationships				9	John	Done
Housing Module			N/A	9 > 14	John	N/A
Tenure				14 > 20	John	Denis
Mortgages & Loans				20 > 70	John	Denis
Other Properties				70 > 118	John/Denis	N/A
Vehicles				118 > 132	Denis	Denis
Valuables				132 > 133	Denis	N/A
Financial Liabilities				133 > 150	Denis	Denis
Financial Investments				150 > 158	Denis	Denis
Inheritance				159 > 172	Caroline	Caroline
Business Interests			N/A	173 > 183	Jacqueline	N/A
Household Consumption			N/A	183 > 184	Jacqueline	N/A
Attitudes about Saving			N/A	185 > 197	Jacqueline	N/A
Interviewer Questionnaire (Paradata)				1 > 5 of separate spec	Denis	N/A
Self Completion Tests					John	To Do
Self Completion changes						Conor
HFCSSurvMgmt changes						Conor
BankAcc change						Conor

Figure 2 shows the Routing tests for the Household blocks. The table contained a link to the log template. The tester signed and took a copy of the log template. The test log was then completed (along with the error log) and then attached back to the table.

Figure 3: Completed test log for Routing of Vehicles Block

HFCS 2012: Vehicles - Routing test log template						
Higher Level Condition	Variable Name	Page Ref	Test Performed	Expected Result	Actual Result	Fail / Pass
	Cars	123	Cars = 1	Cars_No	Cars_No	P
			Cars = 2	Oth_Veh	Oth_Veh	P
	Cars_No	124	Cars_No = 0	Oth_Veh	Error message	Fail
			Cars_No = 1 to 4		Cars_Val	Fail
			Cars_No > 4	CarsVal	Signal, followed by Cars_Val	P
	Cars_Val	124	Cars_Val <> EMPTY	Oth_Veh	Oth_Veh	P
	Oth_Veh	124	Oth_Veh = 1	Oth_VehT	Oth_VehT	P
			Oth_Veh = 2 AND Cars = 1	Veh_Buy	Veh_Buy	P
			Oth_Veh = 2 AND Cars = 2	Valuables > 	Valuables	P
			Oth_VehT = 1	HowMany (MotBk)	HowMany(MotBk)	P

Figure 3 shows a completed test log for Routing tests that were performed on the Vehicles block. If a test was marked 'fail' the variable details along with the test performed were written to the error log [figure 4

Figure 4: Error Log for Routing of Vehicles Block

HFCS 2012: Vehicles - Routing error log					
Variable Name	Spec Page Ref	Scenario	Possible Error / Problem	Comment	To be resolved by
Cars_No	124	Cars_No = 0	Test plan says it should go to Oth_Veh but error message is generated saying range 1-99	Limits not stated in spec. Gerry said he intended to allow 0. Now says leave as is.	Denis
		Cars_No 1-4	Test plan says it should go to Oth_Veh but routes to Cars_Val.	Check Expected Result. O.K. as per spec 6/11	Denis
Business	125 - 132	Business = 1 AND HowMany <> 1	Test plan says HowMany <> 1.	0 not allowed in coding but Gerry may have wanted it. Check with Gerry. Gerry said leave as is.	Denis
Other_V	133	Other_V = 0 AND Cars = 1	Test plan says route to Veh_Spnd but form routes to Veh_Buy.	Check Expected Result as Veh_Buy should be asked before Veh_Spnd. Error in Test Plan.	Denis
OthVehVal	133 - 136	Other_V = 2 AND Cars = 1	Test plan says route to Veh_Spnd but form routes to Veh_Buy.	Check Expected Result as Veh_Buy should be asked before Veh_Spnd. Error in Test Plan.	Denis
OthVehVal	133 - 136	Other_V = 1 etc. AND Cars = 2	Test plan says route to Jewels or Valuables but form routing to Veh_Buy	Recheck spec. and check Expected Result. O.K. as per spec 6/11	Denis
Filter		Cars = 1 AND Oth_Veh = 1	Test plan says ask Veh_Buy if Cars = 1 AND Oth_Veh = 1. Spec says as Veh_Buy if Cars = 1 OR Oth_Veh = 1.	Check Expected Result. Spec 6/11 states: Ask if Cars=1 or Oth_veh=1	Denis

Figure 4]. The error log was assigned back to a developer who resolved the problem or addressed any issues raised in the document.

2.1.3.1 Test Design Techniques used to create Test Logs for Independent Component testing

For our Independent Component Testing we used Specification-Based (Black box) techniques to derive our test cases. The test logs were designed from the specifications - independently of the developers to ensure that the specifications have been implemented fully and both processes have arrived at the same results.

We employed a number of Software testing techniques to derive and prioritise Test cases for the Logs. The techniques were to be employed as aids to derive test cases for each of the approaches to testing.

Routing - Decision Tables, Flow Charts/State Transition tables

Variable Ranges - Equivalence Partitioning, Boundary Value analysis

Fills/Inserts - Use Case testing

Question text/Appearance - Use Case testing

2.1.3.1.1 Equivalence Partitioning

Equivalence partitioning is based on a very simple idea: Inputs into a program can be classified into groups of similar inputs. For example a variable defined as an Integer will accept as valid any input that is numerical and will reject anything else [characters, symbols]. The range of numbers is infinite (though computers will limit the Integer to a finite definition). Equivalence partitioning requires us to test once an

2.1.3.1.3 Decision Tables

Specifications define the conditions under which a function operates. The conditions can get quite complex so it's important to try to ensure that every combination of the conditions has been tested.

A decision table lists all the input conditions and all the actions that arise from them. The conditions are structured into tables as rows and below the conditions are the resulting actions that arise from the combination of true/false conditions in the top part of the table.

Example:

The Credit Card details part of the Financial Liabilities module is only to be asked of People aged 18 (Condition 1) or over, who are Students (Condition 2) or working (Condition 3) and possess a Credit Card (Condition 4)

It might be coded as follows:

IF (Age >= 18) and (Credit Card = Yes) and ((Job = Student) or (Job = Working)) Then

Ask Module

Else

Goto End

Endif

In order to achieve the greatest coverage of all possible conditions, the tester should draft and refine a table something like the following:

HFCS2013: Test cases for Financial Liabilities module										
	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10
Conditions										
Aged >= 18	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes
Students	No	Yes	Yes	Yes	Yes	No	No	No	No	Yes
Working	Yes	No	Yes	Yes	Yes	Yes	No	No	No	No
Credit Card	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No
Actions										
Ask Module	Yes	Yes	-	Yes	-	-	-	-	-	-
End Module	-	-	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes

Second reduction of table will remove duplicate tests (cases where respondent is not 18 or over)

HFCS2013: Test cases for Financial Liabilities module							
	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7
Conditions							
Aged >= 18	Yes	Yes	Yes	Yes	No	Yes	Yes
Students	No	Yes	Yes	Yes	Yes	No	Yes
Working	Yes	No	Yes	Yes	Yes	No	No
Credit Card	Yes	Yes	No	Yes	Yes	No	No
Actions							
Ask Module	Yes	Yes	-	Yes	-	-	-
End Module	-	-	Yes	-	Yes	Yes	Yes

Third reduction removes all duplicate tests for Credit Card=No

HFCS2013: Test cases for Financial Liabilities module					
	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Conditions					
Aged >= 18	Yes	Yes	Yes	Yes	No
Students	No	Yes	Yes	Yes	Yes
Working	Yes	No	Yes	Yes	Yes
Credit Card	Yes	Yes	No	Yes	Yes
Actions					
Ask Module	Yes	Yes	-	Yes	-
End Module	-	-	Yes	-	Yes

Rules 1 - 5 in the last decision table are the test cases which should be performed by the tester to provide maximum coverage for all possible combinations of conditions. The Expected results of each test are the actions of each rule. This technique is particularly useful in systems where combinations of input conditions produce various actions.

2.1.3.1.4 Flow Charts \ State transition Diagrams

Although generally only used to represent code, we encouraged our test log authors and programmers to draw the specifications using flow charts. This helped to get a greater understanding of the routing and validations in the specification and also proved to be a useful aid in generating test cases.

We adapted and used state transition diagrams to represent the routing through blocks also. This was a useful technique when we were developing our Use cases for scenario testing.

2.1.3.1.5 Use Case [or Scenario] Testing

Based on results from previous household Surveys we were able to design household scenarios to ensure correct routing through the household profile. While we were aware that a comprehensive list of all possible scenarios was not achievable, we could direct our tests based on households that took part in previous household surveys and weight our tests according to those profiles. This method of testing in conjunction with Question-by-Question testing helped us to ensure as near as possible to full testing coverage of the Questionnaire.

2.1.4 Integration testing

The purpose of Integration testing was to expose defects between the Survey Instrument and all of the other system components and interfaces. Workflows and Use case scenarios were the test bases and these formed the strategy for ensuring that the system was functioning as it should. The focus of the integration test was to ensure that all components and interfaces were interacting correctly.

Non-functional requirements were also tested at this level:

- Installability – installation procedures
- Maintainability – ability to introduce changes
- Performance – expected behaviour
- Load & Stress handling – System behaviour at upper limits of usage and data load
- Recovery – Procedures in the event of failure
- Usability – ease with which users can engage with the system

2.1.5 Acceptance testing

These tests were performed by the business users. The purpose was for verification that the requirements had been fully met and that the system provided for the Business needs. This testing was performed independently of IT. The Business area performed their own scenario testing to ensure compliance with the specifications.

3) Results

Independent testing of the Questionnaire was also performed by the Blaise team throughout the latter stages of the development lifecycle. During Integration testing we were simultaneously testing and re-testing the questionnaire to ensure all amendments and fixes had been implemented correctly.

There were over 80 test logs produced from the specifications. It took 2 people approximately 3 weeks to document the specifications and to devise all tables, logs and other documents required for testing. The logs were prioritised in terms of complexity and risk. It took 3.5 Independent testers 15-20 days to complete the logs. Testing and re-testing continued until Questionnaire sign-off which was 1 week before the Questionnaire was released for pilot testing.

Testing documentation was reviewed and updated throughout the lifetime of the testing process. Any changes resulting from the pilot were incorporated into the test logs. The pre-live Questionnaire was released to the testers again for another iteration of testing.

At the outset of test planning we established exit criteria to define when the testing was complete. Our Primary target was that every log should be completed (in priority order) within the time assigned. We assigned extra testers to achieve this. It was critical that all incidents raised were corrected, retested and signed off – or waived by the Development Manager.

Independent Questionnaire testing ensured no critical problems occurred in the field. The commitment of 20% of development solely to testing meant that we released a robust Questionnaire to the field on schedule. The only problems logged to the Helpdesk were generally of a 'User education' or training nature.

4) Conclusion

The biggest challenge of the testing process was documenting the Specifications into logs for all of the test approaches. The logs were designed so that they could be given to any member of staff and he/she would be able to launch the Questionnaire and complete the log easily.

The process proved very profitable in terms of Quality assurance. The approach developed for the HFCS has become a valuable template for any new development work performed by the CSO's Blaise team.

References & Further Reading

"Methods for Testing and Evaluating Survey Questionnaires" - Presser, Rothgeb, Couper, Lesser, Martin, Martin, Singer. ISBN 978-0-471-45841-8

“Software Testing: An ISTQB-ISEB Guide – Brian Hambling (Editor) ISBN 978-1-906124-76-2

“Software Testing in the Real World” – Edward Kit 1995 ISBN 978-0201877564

A Questionnaire Guide to Web Accessibility

Mark M Pierzchala, MMP Survey Services, LLC

1. An Approach to Achieving Web Instrument Accessibility

Statistical agencies around the world are required to field accessible web survey instruments. While there are general web accessibility standards (such as the US Section 508), there is little guidance on what accessibility means for a complex web questionnaire. Complicating the situation is that the people who produce these elaborate web survey instruments may not have adequate knowledge of accessibility issues.

While this paper uses the term 'accessibility', of equal importance is that the web survey instrument be understandable and useable for a blind user with a screen reader. 'Accessibility' encompasses many disabilities besides blindness such as other visual deficiencies, motor, hearing, and cognitive disabilities (Brenner 2013). It is possible that a web survey is technically accessible but not understandable and useable to people using screen readers and other assistive technologies. This paper concentrates on web-survey accessibility for the blind.

Most example screens used in this document are based on real-life examples from surveys worked on by the author. For many of the screens there was considerable discussion and experimentation with clients, study directors, specification writers, and programmers about the best way to **visually** display the survey concepts. This paper proposes ways in which these screens can also be portrayed, **aurally** and **linearly**, to the screen-reading user. This work is preliminary. The improvements to Blaise IS, while profound and well executed, have been made too recently to test these suggestions. This paper has a few goals:

- Describe some important Blaise IS accessibility features.
- Motivate organizations to take up and complete this work; that is, produce a standard set of guidelines for web survey accessibility which people can use to judge their instrument's web accessibility and usability for complex screens. This standard would be based on model questions and screens and would suggest ways that each kind of screen can achieve accessibility.
- Compare and contrast visual web design versus aural web design for complex screens.
- Find ways to make one instrument design work for both sighted and blind users.
- Give information for Blaise 5 screen accessibility requirements.

1.1 Acknowledgements

The Blaise Team underwrote the hours spent on this research and writing and improved Blaise IS accessibility. Tim Carati of the Blaise Team provided technical help and gave many key insights and explanations. MMP Survey Services, LLC paid for a JAWS license, wrote two example datamodels, tested, and suggested improvements.

Most examples are from the author's work at Mathematica Policy Research, Inc. (MPR) and include screens from the National Survey of Recent College Graduates (sponsored by the National Science Foundation) and the Kauffman Firm Survey (sponsored by the Kauffman Foundation). Additionally, most screens in this document use MPR web survey standards, sometimes with adaptations to illustrate a point. Two screens are from University of Michigan Survey Research Center Web Survey Guidelines. The author thanks all three organizations for their help and their permission to use their example screens. Larry Malakhoff of the US Census Bureau hosted me for several hours a few years ago and showed me many things regarding screen-reader technology, and I thank him for his time and sharing. I also thank Jim O'Reilly of Westat for pursuing accessibility issues with The Blaise Team over the years. Linda

Bandeh of MPR, Karen Brenner of Westat, Larry Malakhoff of the US Census Bureau, Tim Carati of Statistics Netherlands, and Esme Pierzchala (daughter) reviewed a late version of this paper and provided key comments. For two of them, I lifted text right out of their emails and plopped it in with attribution. I am grateful to all four reviewers.

1.2 Important Blaise Accessibility Features

☞ Important Blaise Accessibility features are marked with a pointed finger.

2. Blaise IS Model Instruments

Two Blaise IS model instruments were created to illustrate model screens. These are now part of the Blaise 4.8.4 distribution under the Samples\Internet\Interview\BISAccessibility folder. These two instruments are called Forms and Grids. Both datamodels illustrate issues with multiple items per page.

- The Forms screens either have single items or have two or more related items. For multi-item screens, the user must understand the whole screen and how its items express a survey concept when combined. Examples in sections 3.1 and 3.2 are from the Forms datamodel.
- The Grids screens are tabular screens. These screens have several or many items. For example, there is a household roster grid and business related grids. Examples given in Section 3.3 are from the Grids datamodel.

3. Visual versus Spoken Experience

Sighted web-survey users have several means to understand complex screens.

- They can assess the screen as a whole, and study its parts in any order.
- They can take the time to study a screen, and all items on the screen, to understand how the items relate to one another and how they together express a survey concept.
- They can see visual cues such as question formats, indentations, colors and shadings, text that can be bolded, underscored, or italicized, indentations, and grid lines to organize the page.

Blind users rely on screen readers that read the underlying HTML of a web screen.

- A screen reader proceeds linearly through a web screen from left to right and from top to bottom.
- Blind users have to build up a mental image of the screen in order to understand the screen.
- While users can dictate the rate of reading, back-up, and fast forward, this navigational ability does not approach that of sighted users.

Computer users who make use of screen readers have many options they can set in order to make the screen reader behave the way they desire. This includes how much is read from Windows, the rate of speech, the way the voice is synthesized, and so forth. Examine the following from Figure 18a below.

“How many children who live with you as part of your family are . . .”

On one of my computers the screen reader will read the ellipses at the end as **“dot dot dot”**. On the other computer these periods are ignored. This is due to different setting levels between the computers. On a practical level, the web-survey producers cannot assume anything about screen reader configuration. This is a personal choice by the blind user and this configuration choice must be honored.

Screen-reading technology makes websites accessible with varying degrees of success depending on the website. It is up to web survey producers to field web survey screens that can be understood well enough to be useful to the blind respondent through the use of a screen reader.

In the following discussion, the term 'item' refers to a place where a value must be entered. An item may be a question or it may be part of a question. This distinction is important because a screen reader operates on the level of an item. That is, it stops at an item and waits for an answer from the user.

3.1 Single-Item Examples

Figure 1 and Speech Listing 1 illustrate differences in question presentation, for a simple single-item, between sighted and blind users. In Figure 1, the visual presentation, bolded text is question text while underscore clarifies the intent of the question. Unbolded italicized text is an instruction and unbolded plain text is for answer choices. Speech Listing 1 gives verbatim text expressed by the screen reader.

Figure 1: Item showing the Uses of Visual Screen Standards

What is the highest level of education you have attained?

Choose one answer.

- Elementary school
- Some high school
- Graduated high school
- Some college
- Graduated college
- Some graduate school
- Masters or higher degree

Speech Listing 1: Text read by the Screen Reader

"What is the highest level of education you have attained? Choose one answer. Elementary school radio button not checked."

☞ The question text is read only at the first choice.

The screen reader says words one after another without stopping. It does not differentiate bolded, underscored or italicized text. It reads the text of the first radio button and indicates that the cursor has landed on a radio button question, and that this first option has not been chosen. Then it stops. By context, the user can guess that there are additional choices and would know to use arrow keys to navigate the list. On the other hand, the screen reader gives no idea of how many choices there are. The sighted user can immediately spot the appropriate choice and can use the mouse to check it. The blind user must listen to each choice in turn. (Note: If any of the choices has already been checked, Blaise IS focuses that choice.)

For this question, the instruction is superfluous for both kinds of users. Note that the instruction intrudes between the question and the answer choices. For some questions, for a blind user, this placement may get in the way of understanding the question. On other questions however, the instruction is useful or necessary.

Figure 2 shows a similar construction but with a much longer list of choices. Speech Listing 2 gives the text that the screen reader expresses.

Figure 2: A Radio-Button Item with many Choices

We need to assign a standardized educational code to the field of study you just listed.

From the list below, please select one category that best describes the field of study for this degree.

- Agricultural Sciences
- Biological or Life Sciences
- Business Management and Administrative Sciences
- Computer and Information Sciences
- Conservation and Natural Resources
- Education
- Engineering and Engineering-Related Technologies
- Health and Related Sciences
- Languages, Linguistics, Literature or Letters
- Liberal Arts or General Studies
- Library Science
- Mathematics and Statistics
- Physical Sciences
- Philosophy, Religion or Theology
- Psychology
- Social Sciences or History
- Social Work
- Visual or Performing Arts, or
- OTHER Field Not Listed

Speech Listing 2:

"We need to assign a standardized educational code to the field of study you just listed. From the list below, please select one category that best describes the field of study for this degree. There are 19 categories in alphabetical order. Agricultural Sciences radio button not checked."

The shaded text in Listing 2 is additional instruction text that is available only to the screen reader. This additional text is in the source code. The idea is to give the blind user an idea of the size and organization of the list. This speech-only text is given in the same spirit as the underscored text or instruction text is given for the sighted user.

☞ The hidden text is produced by using a text font color that matches the color of the screen. This is an implementation feature by the author. It may or may not be a good idea.

Figures 3 and 4 show some clarification text inserted for both sighted and blind users. The second *Name* field in Figure 3a indicates the field length. Figure 3b and Speech Listing 3b shows why this is necessary for blind users.

Figure 3a: Name Field with and without Instruction Text

What is your name?

Name

What is your name?

Please type up to 20 characters.

Name

In Figure 3b below, when trying to type the name **Mark Matthew Joseph Pierzchala**, the sighted user can see that no text past the third name can be typed. However, the speech reader does not (or may not) indicate that the text limit has been reached and it keeps reading the text the user is typing past the 20

characters. The blind user may think that the whole 4-part name has been typed (which it has been) and stored (which it has not been).

Figure 3b: Typing a Name with more than 20 Characters

What is your name?

Please type up to 20 characters.

Name

Speech Listing 3b: Speech Expressed while the User types a Long Name

"M a r k space M a t t h e w space J o s e p h space P i e r z c h a l a"

The top question of Figure 4 illustrates a traditional method of presenting an open question where the typed answer can be many words. To the sighted user, the size of the box indicates that a long answer is invited. The screen reader, on the other hand, does not relate this information to the blind user.

The second open question contains part of an instruction. Speech Listing 4 shows shaded text which is spoken but not seen. This is given since the mouse is not useful to the blind user and since the Enter key just starts a new paragraph in the text box.

Figure 4: Providing instructions to Sighted and Blind Users

Please state how the accident happened.

Please state how the accident happened.

You can type up to several paragraphs.

Speech Listing 4: Second Open Question

"Please state how the accident happened. You can type up to several paragraphs. **Press the Tab key to leave the question.**"

This instruction text could be provided to the sighted user too.

Figure 5a shows 2 examples from the University of Michigan Survey Research Center.

Figure 5a: Dollar and Percent Questions

What was your household income in 2010?

Enter to the nearest dollar.

\$.00

By this time next year, what is the percent chance that the value of your home will have GONE UP by more than 10 PERCENT compared to what it is worth today?

Enter a number between 0 and 100.

%

These examples contain symbols to the left and right of the data entry box that help clarify, to the sighted user, the kind of answer that is requested. Speech Listing 5a lists the first question, the answer to the first question, and the second question. Note that the speech reader stops after the \$ until the user enters a number. Then it continues, but with "dot 00" before it begins reading the next question (see shaded text).

Speech Listing 5a:

Screen reader from screen: "What was your household income in 2010? Enter to the nearest dollar."

Screen reader from user data entry: "45000"

Screen reader from screen: ".dot 00 By this time next year, what is the percent chance that the value of your house will have gone up by more than 10 percent compared to what it is worth today? Enter a number between 0 and 100 dot."

Both displays in Figure 5a are achieved through the use of multicolumn groups in Blaise IS. The .00 in Figure 5b and the % in Figure 5c are auxfields called *Cents* and *Percent* respectively. The \$ before the box in Figure 5b is description text associated with the field *HHIncome*. This is not read by the screen reader. Despite the fact that these symbols are not timely read, the instructions give the needed clarity.

Figure 5b: Dollar Question in a Multicolumn Group

Enter to the nearest dollar.

\$ ← Auxfield Cents

Figure 5c: Percent Question in a Multicolumn Group

Enter a number between 0 and 100.

← Auxfield Percent

(Note: There are ways to stop the screen reader from reading the .00; for example, this text could be implemented as a field description instead of field text. This example thus serves as a caution as to the kind of testing that is needed to correctly implement these constructions.)

3.2 Multiple Item Examples

The following examples show 2 or more items on a screen. These are further subdivided into (1) related items forming a question, (2) answer one item or another, (3) forms, and (4) juxtaposition of items.

3.2.1 Related Items Forming a Question

Some questions consist of two or more items. Examples include further-detail, other-specify, quantity-unit, phone number, date, and time questions. Figure 6a shows a further-detail question. The question first classifies an employer at a high level then refines the classification with the second question.

Figure 6a: A Further-Detail Question

B 11. Which of the following best describes your principal employer during the week of April 1, 2006?
Were you ...

- Self employed or a business owner
- A private-sector employee
- A government employee
- Another type of employee

← **Once this option is chosen,**

Were you ...

- In a local government (e.g., city, county, school district)
- In a state government (including state colleges/universities)
- In the U.S. military service, active duty or Commissioned Corps (e.g., USPHS, NOAA)
- In the U.S. government (e.g., civilian employee)

← **This further-detail item appears**

Contrast this web layout, optimized for a screen reader, with the original paper version of the question in Figure 6b. The high-level choices in Figure 6a are interspersed with the lower level choices on paper in Figure 6b. While a web survey could visually implement this question in the same format, it is hard to see how a screen reader could faithfully translate the question's intent. Interestingly, but not surprisingly, the question format of Figure 6a above also proved optimal for the telephone survey version of the questionnaire.

☞ The first question is marked as a critical question in Blaise IS so the next question appears quickly.

Figure 6b: The Original Paper Format of the Further-Detail Question

B11. Which one of the following best describes your principal employer during the week of April 1, 2006? Were you...

Mark one answer.

SELF-EMPLOYED or a BUSINESS OWNER

- 1 In a non-incorporated business, professional practice, or farm
- 2 In an incorporated business, professional practice, or farm

PRIVATE SECTOR employee

- 3 In a for-profit company or organization
- 4 In a non-profit organization (including tax-exempt and charitable organizations)

GOVERNMENT employee

- 5 In a local government (e.g., city, county, school district)
- 6 In a state government (including state colleges/universities)
- 7 In the U.S. military service, active duty or Commissioned Corps (e.g., USPHS, NOAA)
- 8 In the U.S. government (e.g., civilian employee)

OTHER type of employee

- 9 Other – *Specify type of employer*

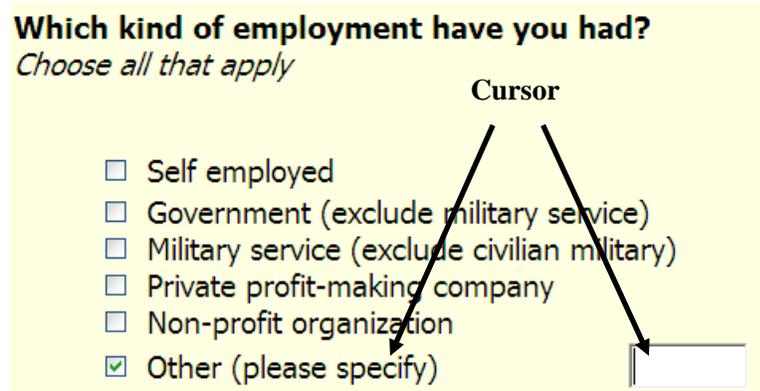
Figure 7 shows an 'other-specify' question consisting of 2 items. The second item, a place to type a text response, is only available if the *Other* choice is checked.

Figure 7: An Other-Specify Question

Which kind of employment have you had?
Choose all that apply

Cursor

- Self employed
- Government (exclude military service)
- Military service (exclude civilian military)
- Private profit-making company
- Non-profit organization
- Other (please specify)



For the sighted user, the appearance of the box informs that an additional response is needed. For the blind user, prompting text is helpful as shown in Speech Listing 7.

☞ This is an other-specify group in Blaise IS.

Speech Listing 7: Prompting text for the Specify Item

"Type other kind of employment in 20 characters or less."

Figure 8 shows two ways to implement a quantity-unit question for distance travelled. A helping text is inserted for the screen reader to inform the blind user of the question's two-part nature.

Figure 8: A Quantity-Unit Question

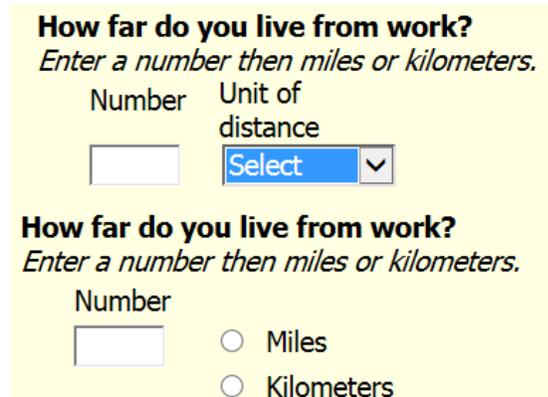
How far do you live from work?
Enter a number then miles or kilometers.

Number	Unit of distance
<input type="text"/>	<input type="button" value="Select"/> ▾

How far do you live from work?
Enter a number then miles or kilometers.

Number

<input type="text"/>	<input type="radio"/> Miles
	<input type="radio"/> Kilometers



☞ These are multi-column groups in Blaise IS. Special fieldpanes in the mode library help with these displays. These mode libraries are in the sample distribution.

Figure 9 shows a three-part date question. The instruction text is meant for the screen reader.

Figure 9: A Three-Part Date Question

What is the date of your birth?
Please enter month, day, and year starting with month.

Month	Day	Year
<input type="text" value="Select"/>	<input type="text"/>	<input type="text"/>

Figure 10 shows a three-part time question.

Figure 10: A Three-Part Time Question

What time did you eat this meal?
Please enter hour, minute, and AM or PM starting with the hour.

Hour	Minute	
<input type="text"/>	<input type="text"/>	<input type="radio"/> AM
		<input type="radio"/> PM

What time did you eat this meal?
Please enter hour, minute, and AM or PM starting with the hour.

Hour	Minute	AM or PM
<input type="text"/>	<input type="text"/>	<input type="text" value="Select"/>

☞ These are multi-column groups in Blaise IS.

3.2.2 Answer One Item or the Other

Figure 11 gives an example where you want the user to enter a phone number with a North American format or a phone with a different format, but not both. In this kind of screen, you can indicate to the blind user that (1) a choice of formats is available and (2) how to get to the second choice.

Figure 11: Enter a North American or International Phone Number

What is the phone number of this contact?
Enter a U S or Canadian phone number starting with the area code, or for an international phone number, tab 3 times.

Area code	Prefix	Number
<input type="text"/>	<input type="text"/>	<input type="text"/>

OR

International phone

Figures 12 and 13 show more examples of choices of questions to answer. In Figure 12, the clarifying text is visibly on the screen.

Figure 12: Enter a Graduation Date or confirm You did not graduate

A 1. In what year did you receive your high school diploma or equivalency certificate?
If you have neither, skip this question.

Year (yyyy)

Check here if you did not finish high school

In Figure 13, there is a clarifying screen reader instruction, but it is hidden from view.

Figure 13: Enter a U.S. State or a Canadian Province

A 2. In what U S state or Canadian province or territory did you attend high school?

U S state

OR

Canadian province or territory

Speech Listing 13: Enter a U.S. State or a Canadian Province

"A 2. In what U S state or Canadian province or territory did you attend high school? **Skip the state combo box to enter a Canadian province or territory.**"

3.2.3 Forms

Figure 14 gives an example of a data entry form for a name collection. These kinds of screens are probably common enough that it is not necessary to give any special screen-reader instructions.

Figure 14: A Form for a Name Collection

Please enter your name in the boxes below.

Title

First name *

Middle name

Last name

Suffix

☞ Both examples of forms in this datamodel are achieved with special groups in Blaise IS.

☞ The leading text on the page, "Please enter your name in the boxes below." is text associated with an auxfield. The screen reader will read this text first then proceed to the text for each box in turn.

3.2.4 Juxtaposition

Juxtaposition is the practice of putting two similar items on the same web screen as a way to help the sighted user distinguish between them. Figures 15 and 16 below use juxtaposition.

The items in Figure 15 were originally placed one web page after another, but the questions are so visually similar, users could think they were answering the same question twice. The same-page layout clarified that situation. Additional text for the screen reader attempts to communicate this juxtaposition to the blind user (Screen Listing 15).

Figure 15: Juxtaposition of Parents' Educational Achievement

What is the highest level of education your mother or female guardian achieved?

- Elementary school
- Some high school
- Graduated high school
- Some college
- Graduated college
- Some graduate school
- Masters or higher degree
- Not applicable

What is the highest level of education your father or male guardian achieved?

- Elementary school
- Some high school
- Graduated high school
- Some college
- Graduated college
- Some graduate school
- Masters or higher degree
- Not applicable

Screen Listing 15: Two Similar Items on the Same Page

"There are two questions on this screen. The first is about your mother or female guardian and the second is about your father or male guardian. What is the highest level of education your mother or female guardian achieved? Elementary school radio button not checked. To change the selection press up or down arrow."

The items in Figure 16 were placed side-by-side in order to help communicate to the user how the answers to the items should relate to one another. The arrangement helps to communicate a before-and-after relationship.

Figure 16: Juxtaposition of Money Borrowed and Money still Owed

The first question asks about the TOTAL amount you have borrowed to finance undergraduate degrees you completed before October 1, 2010, and the second asks how much you still owed as of October 1, 2010.

Total Amount Borrowed

- Did not earn a degree at this level
- None
- \$1 - \$5,000
- \$5,001 - \$10,000
- \$10,001 - \$15,000
- \$15,001 - \$20,000
- \$20,001 - \$25,000
- \$25,001 - \$30,000
- \$30,001 - \$35,000
- \$35,000 or more

Amount Still Owed as of October 1, 2010

- Did not earn a degree at this level
- None
- \$1 - \$5,000
- \$5,001 - \$10,000
- \$10,001 - \$15,000
- \$15,001 - \$20,000
- \$20,001 - \$25,000
- \$25,001 - \$30,000
- \$30,001 - \$35,000
- \$35,000 or more

☞ These side-by-side displays are achieved with multi-column groups in Blaise IS.

Screen Listing 16: Two Similar Items on the Same Page

"**There are two questions on this screen.** The first question asks about the total amount you have borrowed to finance undergraduate degrees you completed before October 1, 2010, and the second asks how much you still owed as of October 1 2010. **What is the total amount you borrowed?** Did not earn a degree at this level radio button not checked. Radio button not checked. To change the selection press up or down arrow."

In order to communicate the juxtaposition to the blind user, the first shaded text above was added. In order to clarify the question statement itself, the text "What is the total amount you borrowed?" replaces the column header.

3.3 Tabular Displays

Tabular screens, or grids, present additional challenges. They efficiently display and collect data. But the blind user might need more cues to navigate the screens, understand the questions, and provide correct answers. The strategies employed for tables extend those that are used for other multi-item displays.

3.3.1 Household Roster

A common table is the household roster. The one shown in Figure 17 has person rows and columns for Name, Gender, Age, and Relationship. A few strategies were used to present this information to the screen reader. First, information describing the entire page is given, (Speech Listing 17).

Speech Listing 17: Cursor in the First Row and First Column

"... the next screen collects a household roster. The household roster collects 3 or 4 items about each person in your household."

Second, text that is appropriate for each cell is given to the screen reader. Speech Listing 17a gives the screen-reader text for the first row. Speech Listing 17b shows the text for the second row.

Figure 17: Household Roster

What is your name?

	Name	Gender	Age	Relationship to you
Person 1	Harold	<input checked="" type="radio"/> Male <input type="radio"/> Female	61	
Person 2	Maude	<input type="radio"/> Male <input checked="" type="radio"/> Female	58	Spouse or partner
Person 3	Ellen	<input type="radio"/> Male <input checked="" type="radio"/> Female	23	Child
Person 4	David	<input checked="" type="radio"/> Male <input type="radio"/> Female	19	Child

☞ For each cell in the table, if there is question text present, Blaise IS will read it. If there is no question text present, it will read the column header.

☞ The row header is read in the first column only. This is because row headers can be very lengthy (e.g., see Figure 19 below). The user can cause the screen reader to repeat the row header.

Speech Listing 17a: Cursor in the First Row and Name, Gender and Age Columns

Name: "Person 1 What is your name?"

Gender: "What is your gender or sex?"

Age: "How old are you?"

Speech Listing 17b: Cursor in the Second Row for Name, Gender, Age, and Relationship

Name: “Person 2 What is the next person’s name?”

Gender: “What is the gender or sex of this person?”

Age: “How old is **Maude**?”

Relationship to you: “What is **Maude**’s relationship to you?”

Note that the screen reader operates the same way in the grid as it would if the questions were linearized, that is, as if they were taken out of the grid and placed one after another.

☞ The text fill is not used for Gender. This is because, upon experimentation, the screen reader would get ahead of the fill being in place (due to the time it takes for the server trip). For example, the text “**What is the gender or sex of [Name]?**” would be read as “**What is the gender or sex of __?**”.

☞ The Name fields are critical Blaise IS fields in order to execute the rules so that fills can be used.

3.3.2 Items placed vertically to give a Tabular Display

Figure 18a shows a table-like structure that collects numbers of children in each age category. A fairly lengthy description of the page precedes the question. Additionally, the text above each cell describes the cell. In this example, all screen reader text is displayed on the screen for both the sighted and blind user.

Figure 18a: A Table-Like Display collecting Age Information

This page asks you about your children's ages.

There are 5 age categories. Enter the number of children you have in each age category.

If there are no children in a category, please enter 0.

How many children who live with you as part of your family are . . .

Children under age 2

Children aged 2 to 5

Children aged 6 to 11

Children aged 12 to 18

Children aged 19 or older

Figure 18b shows the original web-survey page from the 2008 survey.

Figure 18b: The Original Display collecting Age Information

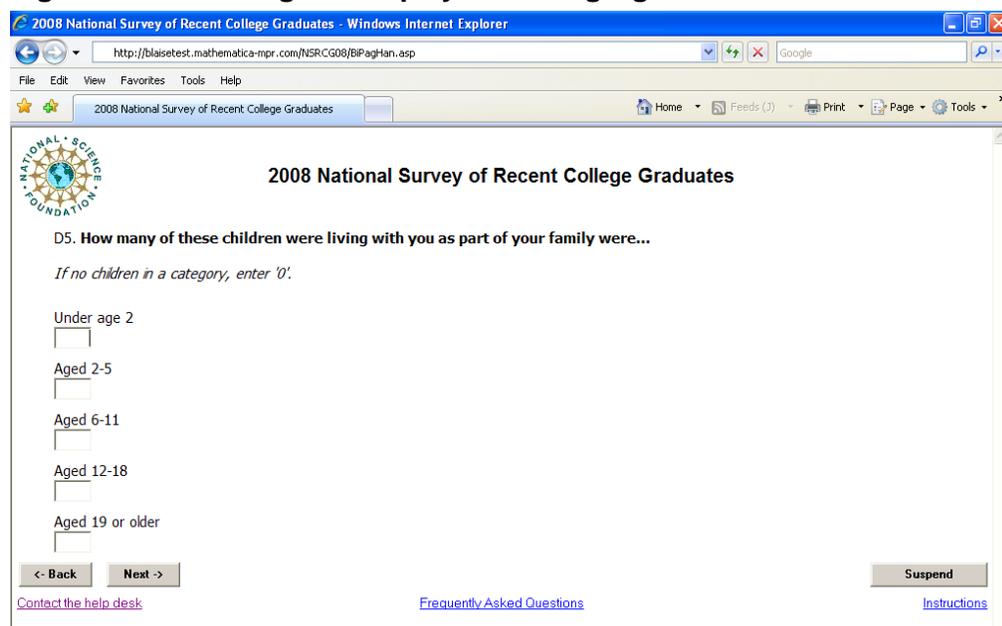


Figure 18a changes the original placement of the instruction text, placing it before the question stem, in order to avoid inserting lengthy instructions between the question and the answer boxes. It also gives clearer text for each category.

3.3.3 Table of Related Items

Figure 19 shows a question stem that relates to 10 yes/no items arranged in a tabular format. This is formed with a Group Table group in Blaise IS. For a sighted user, rows with alternate shading help separate one item from another. However, the screen reader will not notice the shading. On the other hand, it keeps things straight by landing on each item separately until it is answered or skipped.

Figure 19: Table of Related Items

Thinking back to the time you took community college courses, for which of the following reasons did you take those courses from a community college? Was it . . .

	Yes	No
To earn college credits while still attending high school?	<input checked="" type="radio"/>	<input type="radio"/>
To complete an associates degree?	<input type="radio"/>	<input type="radio"/>
To prepare for college / increase chance of acceptance to a 4-year college or university?	<input type="radio"/>	<input type="radio"/>
To earn credits for a bachelor's degree?	<input type="radio"/>	<input type="radio"/>
For financial reasons, for example because of the cost of a 4-year school?	<input type="radio"/>	<input type="radio"/>
To gain further skills or knowledge in your academic or occupational field?	<input type="radio"/>	<input type="radio"/>
To facilitate a change in your academic or occupational field?	<input type="radio"/>	<input type="radio"/>
To increase opportunities for promotion, advancement or higher salary?	<input type="radio"/>	<input type="radio"/>
For leisure or personal interest?	<input type="radio"/>	<input type="radio"/>
For some other reason?	<input type="radio"/>	<input type="radio"/>

☞ Blaise IS will read the question stem when it arrives at the first row, first choice. It will not read it again unless the user requests it from the screen reader.

☞ Blaise IS will read the row header once when it arrives at the row.

3.3.4 Name Collection Grid

Figure 20 shows a grid used to collect first and last names of business owners. The cell for the first name of the first owner is highlighted. The screen reader will read "Owner first name edit". This seems clear enough so no special features were added to this page for the screen reader.

Figure 20: Name Collection Grid

Please record the first and last names of these owners who joined **ABC Business** between December 31, 2011 and December 31, 2012.

	First name	Last name
Owner	<input type="text"/>	<input type="text"/>
Owner	<input type="text"/>	<input type="text"/>

3.3.5 Table collecting Sales Data

Figure 21 shows a matrix that collects percent of sales to three kinds of customer. It could be built like the age-category table in Figure 18a, but the tabular appearance reinforces that percents should add to 100%.

Figure 21: Table-Like Structure collecting Sales Data

D 7. Now we'd like to learn more about the type of customers that **ABC Business had during calendar year 2012. Please estimate the percent of the business' sales that were made to individuals, businesses, and government agencies.**

*Please indicate the percent for each type of customer.
The total of percents should equal 100%.*

	Percent of sales
Percent of sales to individuals? <i>Please enter a number from 0 to 100.</i>	<input type="text"/>
Percent of sales to other businesses? <i>Please enter a number from 0 to 100.</i>	<input type="text"/>
Percent of sales to government agencies? <i>Please enter a number from 0 to 100.</i>	<input type="text"/>

All text appears on the screen and is available to both the screen reader and the sighted user.

Note that in grids, form elements in all rows need to be labeled to be accessible (Malakhoff 2013).

☞ When the cursor is at the first cell, the screen reader says the question stem and the row header. For the second and third cells, the screen reader automatically reads only the row header text.

3.3.6 Two Column Table with Dependency

Figure 22 shows a grid with two columns. It is the most difficult table in this sample.

Figure 22: Table collecting Financing Data

F 7. Please indicate which of the following debt financing options you personally have used during calendar year 2012 on behalf of ABC Business.

If you used a debt financing option, choose yes in the combo box, and a number 1 or greater for the number of times you used it.

	Yes or No	Number used
Have you personally used credit cards for business-related purposes?	Select ▾	<input type="text"/>
Have you personally used personal loans from a bank or other financial institution, such as a mortgage or home equity loan used for business?	Select ▾	<input type="text"/>
Have you personally used business or corporate credit cards issued in your name?	Select ▾	<input type="text"/>
Have you personally used personal loans from any family or friends?	Select ▾	<input type="text"/>
Have you personally used personal loans from any other individuals not associated with the management of the business?	Select ▾	<input type="text"/>
Have you personally used any other sources (SPECIFY)?	Select ▾	<input type="text"/>

This table is the hardest to handle for the screen reader due to the way that the second column depends on the first. The Yes/No column uses a combo box rather than radio buttons because with the radio buttons the use of the Tab key seemed confusing in this table. (For the author, there is a temptation to use the Tab key to jump from the Yes choice to the No choice but it goes to the next column as it should.)

4. The Pieces of the Puzzle

In order to produce accessible web instruments you need:

- Web-survey software that has accessibility capabilities
- A specification process and web-survey screen standards that take into account the needs of both sighted and blind users
- Programming and testing processes and standards that implement accessibility features as part of fielding an instrument.

If the web-survey software does not have accessible features, the survey will not be accessible. On the other hand, it is possible to produce a survey that is not accessible even if the underlying web-software system is accessible.

The web-survey producers should be able to make some assumptions about screen-reading users.

- The user has a basic grasp of the use of screen-reading technology, including pause, back-up, replay, and fast forward.
- The user can change the rate of speed of the screen reader.
- The user has experience in understanding various web-screen controls such as data entry boxes, drop-down menus, radio buttons, and check boxes.

4.1 Versions of Web Browsers, Screen Reader, and Blaise

It is not likely that any web-survey software can handle all combinations of web browsers and screen reading software. There are just too many combinations to program and test, literally numbering in the hundreds. Even the screen-reading software itself may not be able to handle all browsers or browser versions. The research for this paper used JAWS version 13.0.1006 with Windows Explorer version 10 on

a Windows 7 computer. JAWS is produced by Freedom Scientific, Inc. Internet Explorer is produced by Microsoft Corporation. A particular web survey may run well on other combinations, but no other combination was tested. Instruments were rendered in Blaise version 4.8.4.1840 as updated with style sheets and java script files from the Blaise Team as issues were found. The Blaise Survey Processing System is produced by Statistics Netherlands. Westat is the Blaise distributor in North America.

5. Annotated References

References are given for accessible survey software, Section 508, and some publications.

5.1 Web-Survey Software Claiming Accessibility

A few-second internet search for accessible web-survey software yielded two results.

The snap Surveys statement on accessibility is linked below. The document is from 2005. It works down the list of Section 508 checkpoints and describes how the software is accessible for each guideline.

<http://www.snapsurveys.com/accessibility/w3c.shtml#ch1>

Survey Monkey claims Section 508 certification. Check out the following links from 2008.

http://help.surveymonkey.com/articles/en_US/kb/Are-your-surveys-508-compliant-and-accessible

http://s3.amazonaws.com/SurveyMonkeyFiles/508_Guide.pdf

5.2 Section 508 of the U.S. Federal Disabilities Act

Section 508 is regulatory in nature and applies to software procurement. It is part of a much broader law that applies to web accessibility. All US government agencies must follow these requirements.

<http://www.section508.gov/>

<http://www.section508.gov/index.cfm?fuseAction=stds>

5.3 Two Good Book References

This first book, by Mick Couper, is a very good guide to understandable visual web design.

Couper, M. P. (2008), *Designing Effective Web Surveys*, New York, NY: Cambridge University Press.

The second book, edited by Jim Thatcher, is a good reference on web accessibility.

Thatcher, Jim (editor), *Web Accessibility, Web Standards and Regulatory Compliance*, Berkeley, CA: Apress

5.4 Recent Fed CASIC Presentations

The Fed CASIC Conference, usually held in March each year at the US Bureau of Labor Statistics, often has sessions or presentations on accessibility and usability for web surveys. Following is a list of some PDF and Power Point presentations with links to each session. Some of these have good reference lists.

Fed CASIC 2013 https://fedcasic.dsd.census.gov/fc2013/index.php#5_1

Bikmal S., Mohammed A., Jayanthi B. and Berry A. (2013), "Challenges in Making Web Surveys 508 Compliant", RTI International.

https://fedcasic.dsd.census.gov/fc2013/ppt/2013%20FedCASIC_SBikmal_RTI.pdf

Fed CASIC 2012 https://fedcasic.dsd.census.gov/fc2012/index.php#4_3

Horan J (2012), "Section 508 Standards REFRESH", US Department of Labor, Power Point available through https://fedcasic.dsd.census.gov/fc2012/index.php#4_3

Brenner K. (2012) "Accessibility Testing: The Role of Tools, ATs, and Manual Methods" (Westat) https://fedcasic.dsd.census.gov/fc2012/ppt/07_brenner.pdf

Bikmal S. and Sattaluri S. (2012) "Usability vs.. Accessibility in Websites/Web Surveys" (RTI International) https://fedcasic.dsd.census.gov/fc2012/ppt/07_bikmal.pdf

Fed CASIC 2010 <https://fedcasic.dsd.census.gov/fc2010/index.php#11>

Brenner K. and Grant E. (2010) "Incorporating Accessibility in the Development Process" (Westat), Power Point available through <https://fedcasic.dsd.census.gov/fc2010/index.php#11>

Malakhoff L. (2010) "Accessible Web Survey Tools" (US Census Bureau), Power Point available through <https://fedcasic.dsd.census.gov/fc2010/index.php#11>

Matulewicz H. H. and Coburn J. (2010) "Universal Accessibility in Web Survey Design: Practical Guidelines for Implementation (Mathematica Policy Research, Inc.), Power Point available through <https://fedcasic.dsd.census.gov/fc2010/index.php#11>

Fed CASIC 2009 <https://fedcasic.dsd.census.gov/fc2009/index.php#1>

Lawler L. (2009) "Achieving Section 508 Compliance" US Census Bureau

5.5 International Blaise Users Group Presentations

This is a list of related previous IBUC presentations.

O'Reilly J. (2006) "Blaise IS and Accessibility" (Westat) <http://www.blaiseusers.org/2006/Papers/239.pdf>

Pierzchala M. (2006) "Disparate Modes and Their Effects on Instrument Design" (Mathematica Policy Research, Inc.) <http://www.blaiseusers.org/2006/Papers/207.pdf>

Pierzchala M., Wright D., Wilson C. and Guerino P. (2004) "Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey" (Mathematica Policy Research, Inc.) <http://www.blaiseusers.org/2004/papers/24.pdf>

5.6 Incredibly Useful Website

This is one of several useful websites. Webaim stands for Web Accessibility In Mind.

<http://www.webaim.org>

5.7 Personal Communication References

Brenner, K (2013). Personal communication, July 25, 2013.

Malakhoff, L (2013). Personal communication, July 24, 2013. Also, I visited Larry in December 2010 for an introduction to JAWS and the art of using a screen reader.

A New Tool for Visualizing Blaise Logic

Jason Ostergren, Rhonda Ash, The University of Michigan

Overview

The Health and Retirement Study (HRS) is a national longitudinal survey, administered biennially since 1992, on a variety of topics associated with aging and retirement. The HRS instrument is modified in the period between each wave both to improve existing sequences and to handle new circumstances, new research and new public policy, all while maintaining its longitudinal value. HRS has put considerable effort over the years into redesigning many sequences and sections scattered across the instrument. These redesigns require input from people at every level from the researchers to the programmers to the interviewers, with each group possessing different understandings of the workings of the instrument and what can or should change.

One section which has been the focus of multiple redesigns relates to employment and pensions. HRS produced the third major redesign of this sequence for its 2012 instrument. This occurred, somewhat unusually, in a series of face-to-face meetings between HRS programmers, testers, specification writers, and Co-Investigators. The meetings typically involved the use of a projector to demo relevant parts of the instrument and the review of printed specification documents. The redesign was more laborious than it needed to be in part because it was difficult for all parties to envision the effects of changes as they were being discussed.

Afterward, HRS began work on a tool to facilitate the redesign process in the future. The goal was to develop a visual representation of the instrument which could be useful in such a setting. The tool needed to allow easy-to-follow editing of the visualization which would help to preview the effect of a proposed change on flow and then could later be deciphered by programmers and turned into useable code. The result of this effort is a tool we refer to as “Visual Blaise,” which is the topic of this paper. Visual Blaise is a Windows desktop application that graphs the logic of a Blaise instrument one block at a time. It allows elements to be dragged and dropped, cut and pasted, and allows for the addition of new elements. Finally, it is capable of exporting a text file with the logic for each block written as a RULES section in correct Blaise syntax.

Problem Description

In the fall of 2011, HRS embarked on a months-long redesign of the portion of its instrument covering pensions. The pension section has a history of complex logic, spread out across multiple blocks and loops, interspersed with sections covering employment, job history, and other related topics. In fact, this section has seen rewrites and changes almost every wave since Blaise was adopted in 2002, but the changes for the 2012 wave were meant to be more fundamental. While the longitudinal value of the section obviously had to be preserved as much as possible, our Study Director wanted to completely change the structure of the data, consolidating all pension questions into one contiguous area of the instrument in a single array. In addition, he was willing to alter, drop or add content far more than would typically be allowed in a rewrite. Above all, the section needed to be made friendlier to respondents and interviewers, who had frequently gotten lost in prior designs, thus impeding the process of collecting high quality data.

The process of this major rewrite constitutes the backdrop for the development of the Visual Blaise tool discussed in this paper. One part of that process was simply deciding on and prototyping a Blaise program structure that would roughly match the known indispensable content and the desired data

structure outcome. Another part was analyzing and merging similar content from separate pension sections into one. Yet another was designing a flow which was flexible enough to handle the fact that respondents are often unsure about key information regarding their pensions. However, the development staff could not make the decisions about what to cut, what to add, and how to handle the new logic independently. The development of the rewrite became an iterative process (unusual for HRS) centered on weekly meetings in which programmers, spec writers and researchers were present together. Typically, the programmers would demo the latest version of the section, explaining limitations, and showing what would happen to particular types of respondents. The researchers would discover problems with the design and suggest solutions, tinker with question text, codeframes and skips, and would also propose new sequences to handle certain kinds of respondents or pensions more optimally. Spec writers would attempt to record the ideas and fixes that were suggested and turn them into actionable changes for the next round.

As an example of this, the thorniest issues tended to revolve around the need to guide respondents into the correct sequences and to exclude them from other ones, all while providing escape hatches for respondents who get stuck in an inappropriate area. This is a uniquely difficult portion of the HRS interview precisely because respondents' understanding of their own pensions is often sketchy. This means that when a major flow-control question comes up, the design has to take into account that the respondent may answer incorrectly and only later discover their mistake due to nonsensical follow-ups. Getting the design right was thus a dance among the competing needs for logical clarity from the programmers to write reliable code, the need to allow respondents multiple points at which to correct their paths, and the need to avoid asking too many clarifying questions, among other things.

This unusual design process also highlighted a gap in our documentation abilities. While we could demo using the DEP and a projector and we could pass around questionnaire documentation, neither of these could really convey to everyone present how the sequences were bound together by the logic and what it would mean to move a sequence, for example, or add a new escape hatch question in the middle of a sequence. Existing flowchart tools like Delta have not gained favor at HRS, perhaps due to the unsuitability of the format for finding and understanding long complex sequences. We found ourselves making flowcharts by hand, which was time-consuming, or occasionally looking directly at code, which was unsuitable for some people present, or, most often, going with quick sketches on the whiteboard or nothing at all to look at when making decisions. This inability to discuss changes from a common understanding of the flow led to a lot of backtracking when discussions or demos in subsequent meetings revealed logical failures in the design.

After the end of the development period for the HRS 2012 instrument, permission was obtained to work on a prototype for a tool that could answer the need for some way of visualizing survey logic which would be accessible to people with different backgrounds. Additionally, we wanted to be able to manipulate the visualization, so as to make it possible to observe the effects of changes immediately, particularly when there were several pieces to handle at the same time. That prototype, which we call "Visual Blaise" (a temporary name that may stick due to inertia), will be described in detail below.

Solution Concept

There are a few stylistic and organizational things about the way the flow chart in Visual Blaise operates which are intended to be different from typical flow charts and other Blaise tools such as Delta. We do not have a deep understanding of the history or classification of flow charts, so there is no grand theory behind the style of the diagrams used in this tool, only our observations about what ways of thinking about flow seemed to speak best to HRS staff.

Figure 1: a portion of a block, containing the HRS questions "P047_" and "P113_" as shown in the Delta tool (note for comparison: the "P149_" variable shown at the bottom is actually a keep statement).

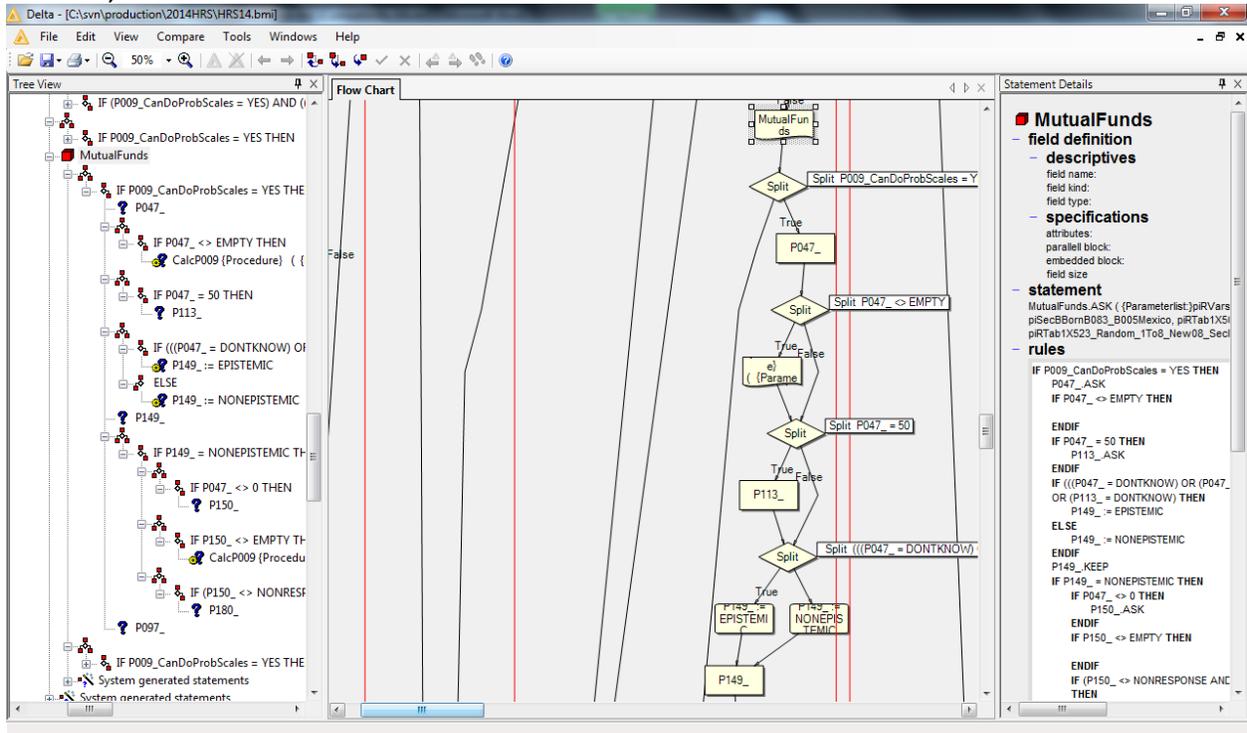
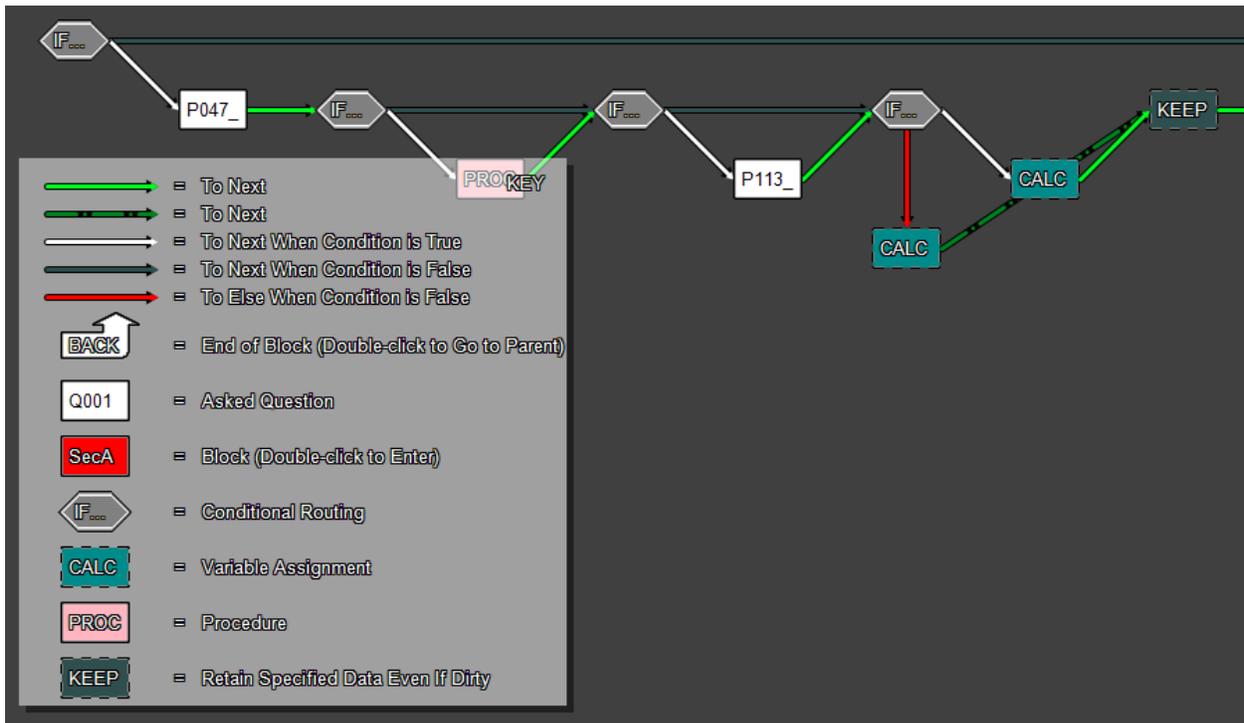


Figure 2: the same logic, as shown in the “Visual Blaise” tool.



The first significant stylistic difference was to make the chart horizontal with the shallowest layer of logic always at the top. Typical flow charts (and Delta) begin at the top and flow downward, expanding to the left and right, with “True” arrows sometimes angling rightward and other times angling left. The reason for making the chart horizontal is mainly about the screen real estate available on widescreen monitors and desks with multiple monitors (which are common at HRS). That is, a typical monitor setup can display much more logic horizontally than vertically. Perhaps the opposite is true if a paper printout is the goal, but we were more interested in a group looking at a screen in a meeting room (and, in any case, paper would not allow dynamic changes, so it couldn’t be the main view). The vertical ordering of the flow chart according to logical depth matters, in part, because one question that seems to float to the top of design discussions at HRS regularly is which questions are asked of (almost) everyone. When splits only angle in one direction and are consistent about which direction “True” and “False” arrows go, it is possible to glance across the top and instantly survey what are the common questions.

Another ambition driving this project was to use colors and motion to improve understanding of the diagram. The arrows are color coded and dashed, if necessary, to help identify their role (bright white coloring for “True” arrows, red for “Else” arrows and gray/green for “False” arrows). It is possible to click and drag a statement icon around (on release, it snaps back into place) for purposes of clarifying its connections in particularly busy segments. The way Visual Blaise handles “Else” is significant also. While “True” arrows always point diagonally downward and to the right and “False” arrows always point horizontal or upwards and to the right, “Else” arrows are distinguished by pointing directly downwards. This helps to keep the diagram compact, but it also helps in situations where there is a series of “ElseIfs” by visually setting that construction apart from the normal flow.

The last significant difference in this visualization was to break it up by blocks. As was suggested above, part of the reason that few at HRS have been interested in Delta may be that the navigation of the instrument (to find the sequences of interest) is done via the statement tree control. The statement tree control is very difficult to navigate for HRS, because there are many layers of difficult to follow

conditions above the main blocks, making it hard to unearth the major pieces of the block hierarchy. Of course there is a find feature, but it seems like the inability to just “see” the main blocks scares off many users.

What this means in Visual Blaise is that the visualization features the logic for only one block at a time, and a user must navigate to child blocks or parent blocks manually (via mouse, keys, or menu, as will be described later). At first this seems like a limitation, but since the logic of a block is largely independent of its parent or child blocks, it doesn’t hinder understanding too badly. More importantly, it limits the “canvas” a user is looking at to a manageable amount, free from artifacts of parent block logic which tend to be decreasingly relevant the farther removed they are (HRS has a proliferation of testing conditions and gate conditions at the top level which clutter our variable universes and contribute to the aforementioned difficulties “seeing” main blocks in the statement tree control).

Object Structure

This paper will not go into great detail about how Visual Blaise is programmed, except for two topics that relate directly to handling Blaise logic: importing the metadata and the internal object structure. This is helpful for understanding how editing works in this tool, which is described in the next section.

The first topic, importing, has to do with the fact that Visual Blaise does not use the Blaise API at all. Instead, it makes use of the xml file produced by HRS’ BlaiseRules tool described in the IBUC 2010 proceedings.⁷ That tool makes use of the RulesNavigator in the Blaise API to extract all the logic and metadata from a specified compiled Blaise file (.bmi) and outputs it in an xml format, organized hierarchically by logic. Visual Blaise reads this xml file and converts it into an object structure in memory which it can manipulate in order to generate flow charts. It is important to note that the BlaiseRules tool is now incorporated into MQDS (as of April 2012), and the xml output discussed here is actually produced and saved as a byproduct of running MQDS, so that anyone who has installed MQDS can make use of it.

One other point to make about importing is that Visual Blaise has an option to import an abridged version of the instrument, which ignores things like keep statements and assignments, in order to focus attention on asked questions. It retains any conditions necessary to reach the asked questions, but it ignores any conditions which only lead to discarded statements. Visual Blaise permanently flags this version as abridged, and it is not possible to switch back to the full version. However, it seems quite useful for some people, since the reduced amount of logic makes it easier to understand.

While building this software, the notion was to develop (though what it was called wasn’t investigated until it came time to write this paragraph) something like a “node graph architecture” for the core object structure and visualization. This would allow for combining the visual advantages of a flow chart with the drag and drop editing capabilities that might be found in a tree view. Additionally, it was imagined as being coded like a linked list, so as to allow easy insertion and removal of elements.

Under the hood, that meant that as the metadata was imported (as described above) it was converted into statement objects, which would become the nodes in the graph. Each Statement object contained a minimum of one and a maximum of two object references. Most statements have only a “next statement” reference, kind of like a linked list. Condition and loop (for..do) statements, however, have two. Typically this would be a “True” statement which represents where the program flows if the statement evaluates true and a “next statement” for when it evaluates false. However, where there is an else

⁷ IBUC 2010 Conference Proceedings, pp. 46-53.

(including elseif) involved, there is no real “next statement” (actually it is present, but for cosmetic purposes) and instead there is a reference to an “Else” statement. With this set of nodes and links, the logic of any block can be described. After that, it is not terribly complicated to display them graphically.

As for the block hierarchy, as the metadata is imported, Visual Blaise keeps a running collection of these linked statement collections for each block. This master collection is keyed so that each set of statements can be matched with their corresponding parent block statement, so as to link the entire datamodel together.

Editing Visually

A big part of the goal of this tool is to allow editing of the logic within the flow chart. This would make it possible to experiment with changes in a way that is more accessible to non-programmers, with the idea being that the results could be saved for later use by the programmer who would turn them into Blaise code. To that end, the ability to drag and drop or cut and paste statements in the flow chart was incorporated.

The drag and drop capability is limited to a single statement. If a selected statement is dragged over another and dropped, Visual Blaise will insert the dragged statement before the statement it is dropped on. The cut and paste capability operates similarly, via right-click pop-up menu when a statement is selected. The “cut” statement is inserted before the statement where “paste” is selected. Cut and paste functionality has the added advantage of being able to move the statement to another block.

The cut and paste feature can also operate on multiple statements, which is useful for moving whole sequences. Any sequence which can be selected can be cut and pasted in the same fashion as a single statement. Selection of multiple statements is possible by selecting a starting (leftmost) statement and then holding down the Shift key and pressing arrow keys until the desired sequence is highlighted, similar to highlighting text via keyboard in a word processor. However, the way selection occurs is limited to prevent nonsensical segments from being operated upon. What this means is that when a user proceeds beyond a split (condition or loop) with the selection, the entirety of the loop or condition, including everything within it, is brought into the selection as a whole. If that is more than the user intends to move, then the user would have to select the relevant subsidiary parts and move them one at a time. There were potentially other ways to handle this multi-select functionality, but we opted for the one which seemed most likely to prevent the user from unintentionally breaking the structure of nested conditions.

Finally, Visual Blaise allows other operations to be performed on selected or multi-selected statements such as deletions or inserts of new statements.

Output

Once changes to logic have been specified, they can be saved in Visual Blaise format and retrieved later. It is assumed that ultimately a programmer will look at the changes and implement them in the Blaise editor. There are a few ways that a programmer could approach this. One would be to work directly from the Visual Blaise flow chart on screen or by printing a copy of the relevant blocks, if appropriate. Another would be to work from a log of changes produced by Visual Blaise while edits are taking place. Finally, Visual Blaise allows any block (or all blocks) to be exported to a text file showing the rules section(s). The output is formatted so it can be pasted over the existing rules of a block and compiled in Blaise (assuming the Fields and other parts are already set), with end ifs and the like properly inserted.

However, comments and some other markup are not preserved, so a more likely use case is that a programmer would simply use this output as a specification, or else a compare tool to merge the changes.

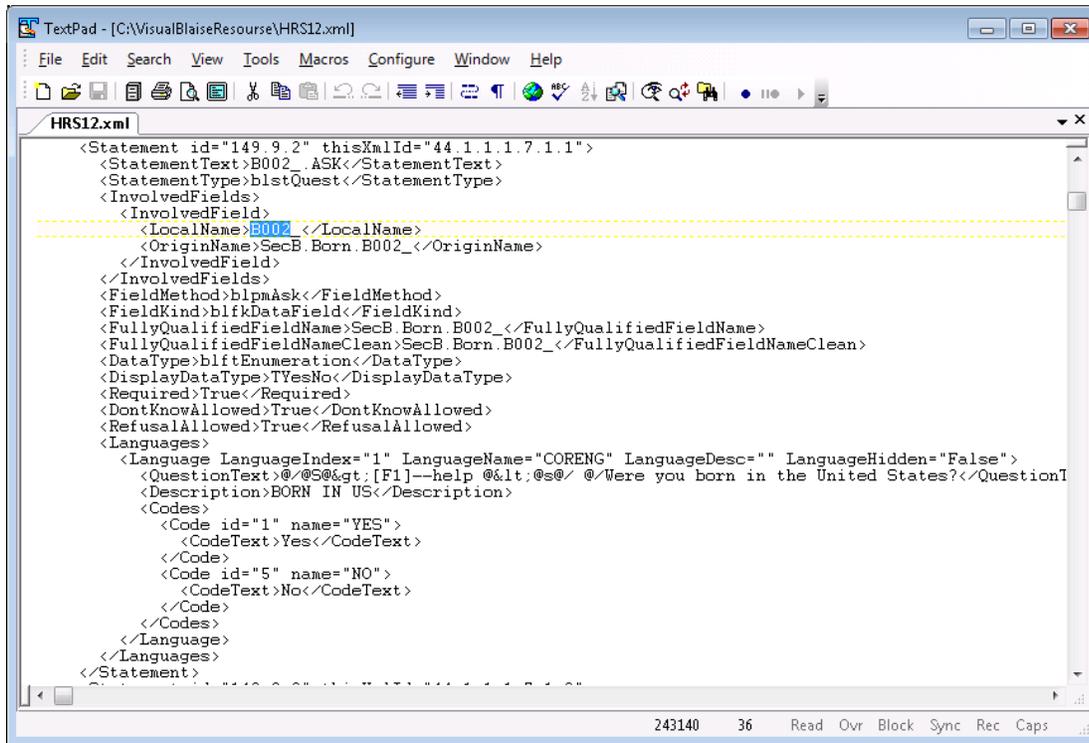
Operating Details of Visual Blaise

The remainder of this paper describes how Visual Blaise works from the user perspective, by illustrating some of the main functions.

Preparation Before Opening Visual Blaise

In order to run Visual Blaise, you will need the compiled files from your survey (.Bmi, .Bxi, etc). A separate program needs to be used to generate an XML file from those compiled files for use as input. As noted above, this can be done by the program called “Blaise Rules” included with MQDS, or by simply running MQDS itself. The XML that Blaise Rules generates is in the format that Visual Blaise uses to construct a flowchart.

Figure 3: an example of a question statement in XML output from Blaise Rules



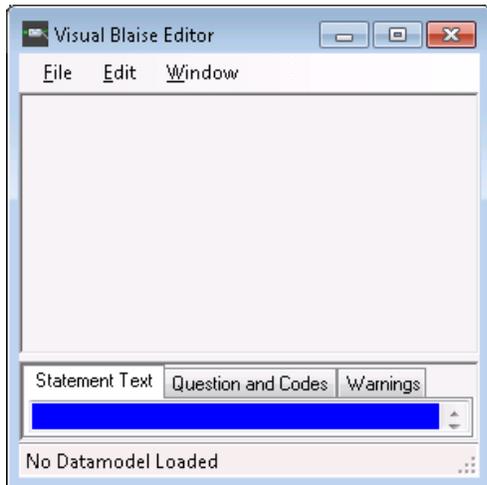
The screenshot shows a TextPad window titled "TextPad - [C:\VisualBlaiseResource\HRS12.xml]". The window displays XML code for a question statement. The XML structure is as follows:

```
<Statement id="149.9.2" thisXmlId="44.1.1.7.1.1">
  <StatementText>B002_ .ASK</StatementText>
  <StatementType>blstQuest</StatementType>
  <InvolvedFields>
    <InvolvedField>
      <LocalName>B002_</LocalName>
      <OriginName>SecB. Born. B002_</OriginName>
    </InvolvedField>
  </InvolvedFields>
  <FieldMethod>blpmAsk</FieldMethod>
  <FieldKind>blfkDataField</FieldKind>
  <FullyQualifiedFieldName>SecB. Born. B002_</FullyQualifiedFieldName>
  <FullyQualifiedFieldNameClean>SecB. Born. B002_</FullyQualifiedFieldNameClean>
  <DataType>blftEnumeration</DataType>
  <DisplayDataType>TYesNo</DisplayDataType>
  <Required>True</Required>
  <DontKnowAllowed>True</DontKnowAllowed>
  <RefusalAllowed>True</RefusalAllowed>
  <Languages>
    <Language LanguageIndex="1" LanguageName="CORENG" LanguageDesc="" LanguageHidden="False">
      <QuestionText>@/050&gt;[F1]--help @&lt;@s@/ @/Were you born in the United States?</QuestionText>
      <Description>BORN IN US</Description>
      <Codes>
        <Code id="1" name="YES">
          <CodeText>Yes</CodeText>
        </Code>
        <Code id="5" name="NO">
          <CodeText>No</CodeText>
        </Code>
      </Codes>
    </Language>
  </Languages>
</Statement>
```

Visual Blaise Functionality

Once you have obtained an appropriate XML from your survey, run the Visual Blaise executable. You will see a standard Windows menu and several tabs that will be described below

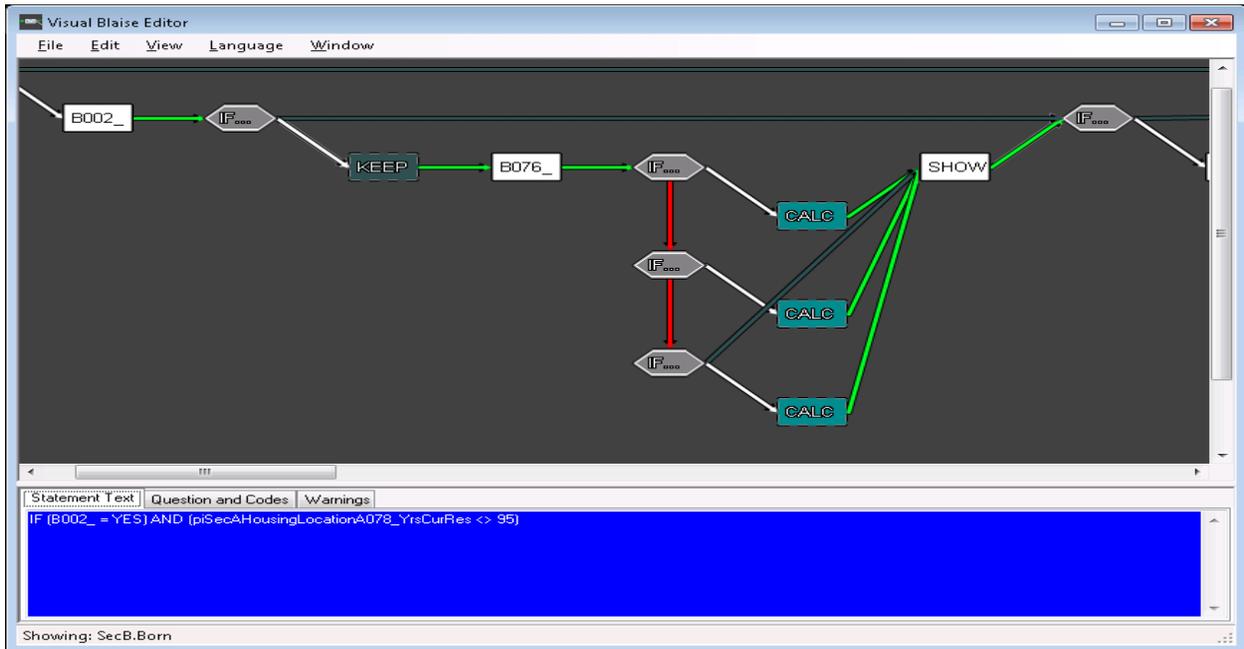
Figure 4: the starting screen of Visual Blaise



File Menu

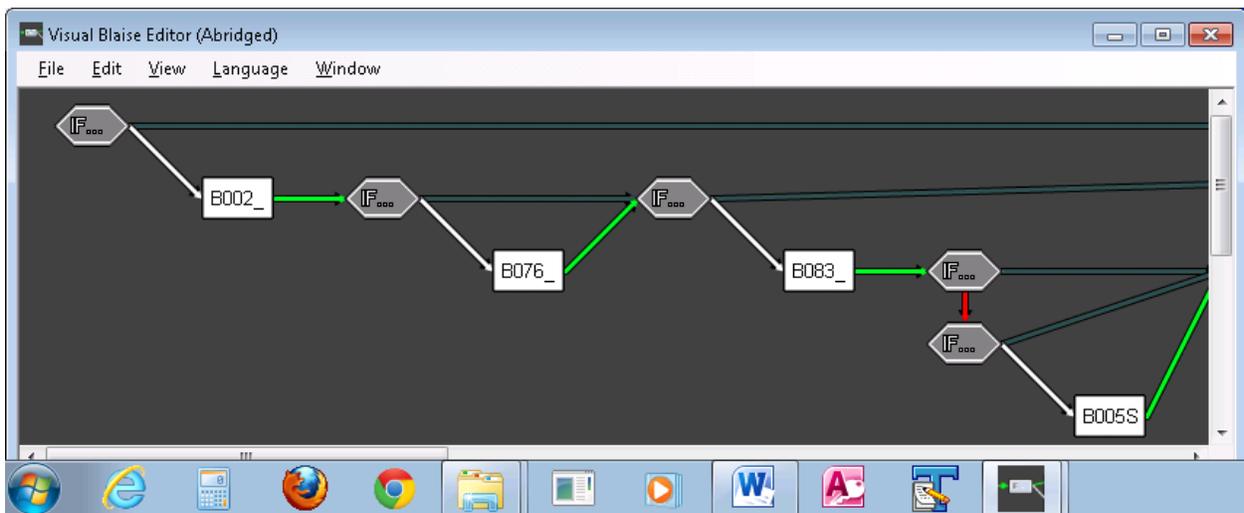
There are several File->Open menu options to choose from. The **From XML** option will provide a full view of the metadata with KEEPs, SHOWs, CHECKs, SIGNALS, and expanded logic. This view is great for a technical review, but may be too much information for a casual observer who does not need to know all the internal Blaise details.

Figure 5: Visual Blaise showing all statements, including keeps, assignments and conditions that do not lead to asked questions



The **From XML, Abridged** option will omit several background elements and show only the major flow information. This view may be easier to follow for the “not-so-techy” users.

Figure 6: Visual Blaise showing an abridged version of the same sequence



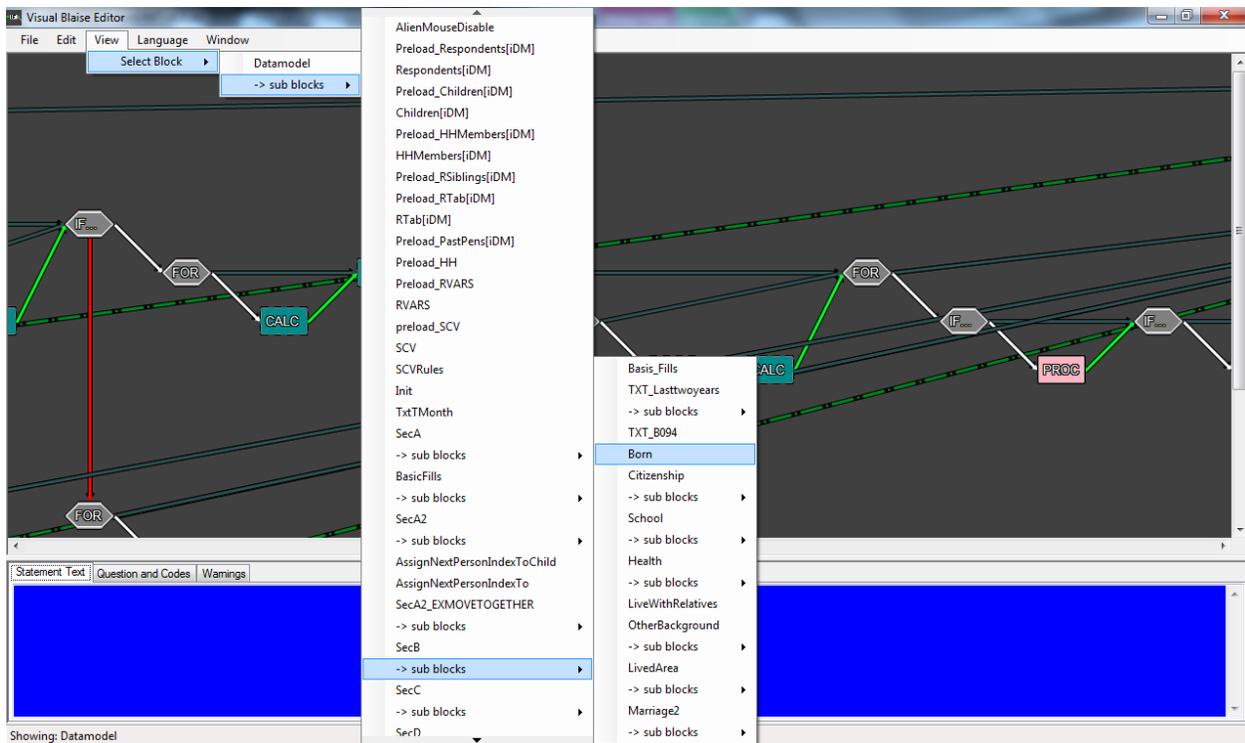
Edit Menu

If you are looking for a specific variable within the block in which you are currently located, you can use the menus Edit **F**ind field functions, or [CTRL]+ F. Then, [F3] will allow continued searching for the next instance of that variable. The Edit menu allows **U**ndo (up to 4 levels) or [CTRL]+ F.

View Menu – Select Block: Finding Where and Getting There

It can be frustrating to find a variable in a very large instrument. Visual Blaise provides the ability to move to a specific block without having to navigate through the editor, block by block. This feature is located under the View menu. A drop-down of blocks is shown. Sub-blocks are provided as off-shoot drop-downs. This makes it easy to see the structure of your survey and move to the block that you wish to review and alter.

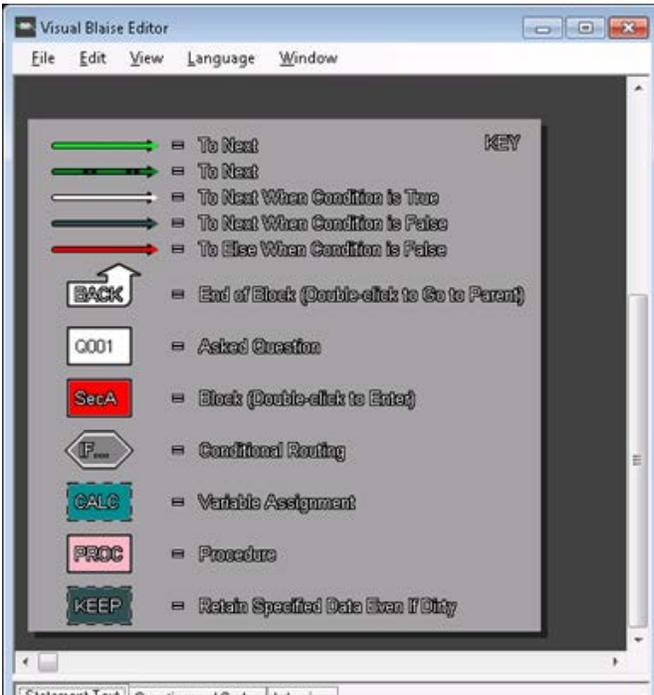
Figure 7: selecting a block via the “View” menu



Window Menu – Show Key

To help with the navigation and symbols there is a Key. This is toggled on and off in the Window menu option.

Figure 8: the symbol key



Attribute Tabs

At the bottom of the screen there are three tabs: “Statement Text”, “Question and Codes” and “Warnings”. As you move around in the flow diagram you will see attributes of each element presented. The information displayed will depend on the type of element selected.

Statement Text

Figure 9: with a variable selected

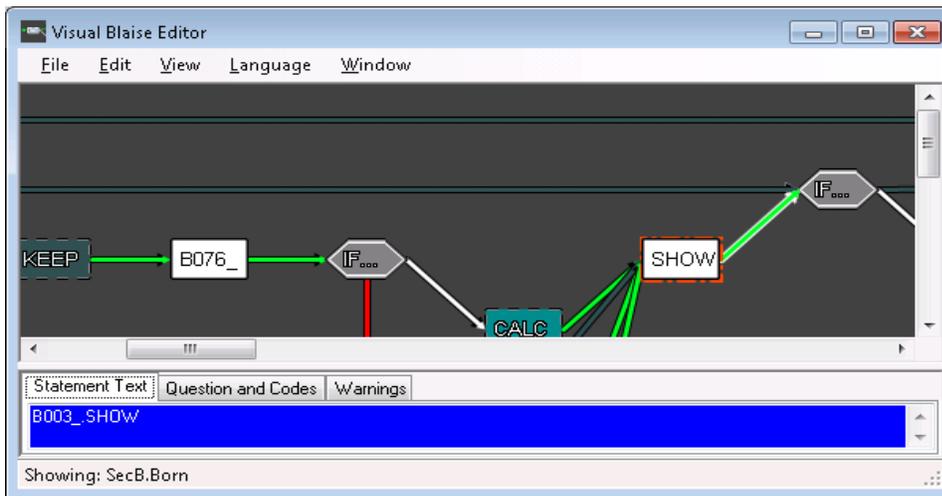
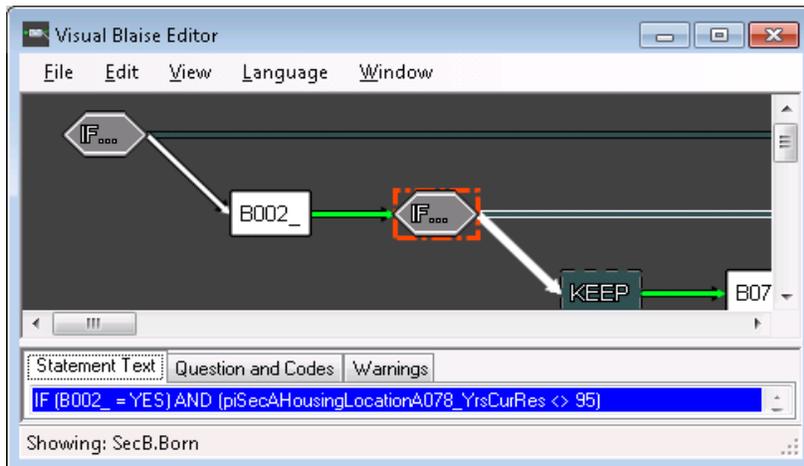


Figure 10: with conditional routing selected, the “Statement Text” tab shows the text of the condition

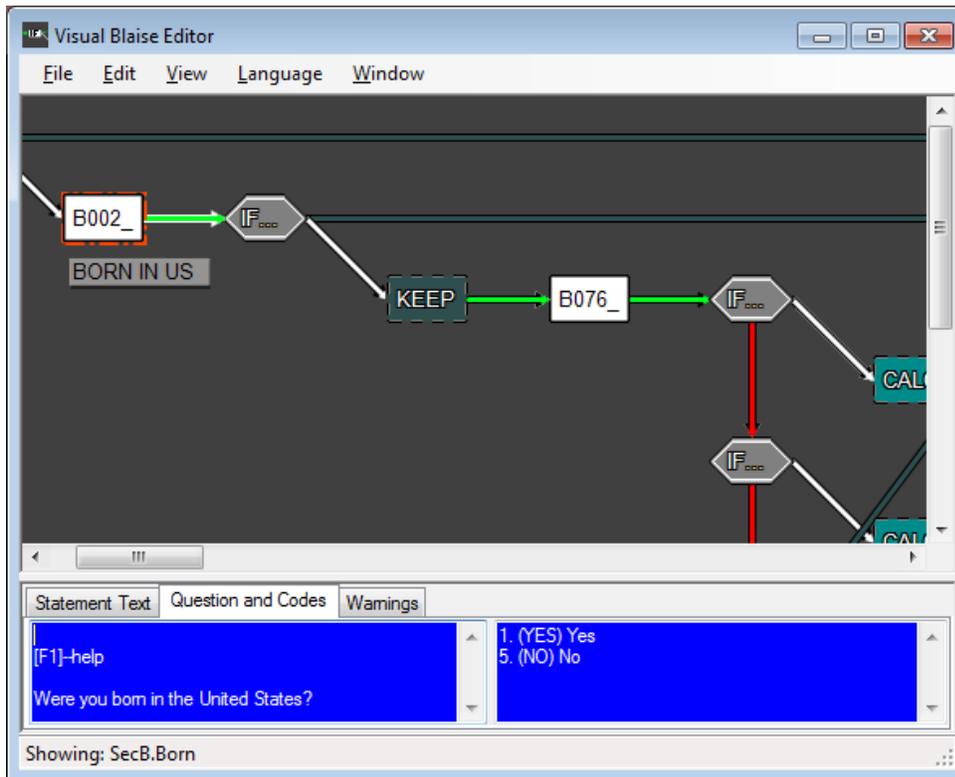


- **Checks or Signals** are represented by two elements, one indicating a hard or soft check and the other with the content of the check in the “Statement Text” tab:
(((B006_ >= 1880) AND (B006_ <= A501_CurDateYear)) OR B006_ = NONRESPONSE
- **Calculations** show the assignment in the “Statement Text” tab:
B005S := 'Mexico - assigned'
- **Procedures or Blocks** display the call and the parameter list.
TXT_B094 {Procedure} ({Parameter list:}piRespondents1X065ACouplenss, piInitA106_NumContactKids, FLB094)

Question and Codes Tab

The Question and Codes tab displays the variable text and code frame.

Figure 11: showing question text and codes of selected question (B002_)



Warnings Tab

The warning tab provides an alert to possible errors when changing the flow location of a variable. For example:

8:56 PM: Moved or cut B008_ used in (Check Statement) *NOT ((B006_ = EMPTY AND B007_ = EMPTY) AND B008_ = EMPTY) CORENG "You must answer one of these categories"*

Use Case

The following section will attempt to describe the design process and the uses of this tool in that process. We will walk through a few of the main commonly requested activities, including maneuvering through the flow chart, adding and deleting variables, manipulation of variables, and tracking connections.

Starting with the specifications from the last wave, the first step would be to review any reported problems during the data collection and determine the need to alter the specifications. The relevant Co-Investigators, by reviewing data and monitoring current affairs, will request changes to the specifications. Our Design Coordinators collect these requests and oversee the development process of the main survey instrument. This process starts with making decisions and developing specifications based on these change requests, on field feedback, on staff analysis of data, and on wait-list items that were not handled in the previous wave. Programmers will then add new content, cut unneeded fields, or change the sequence of the questions that are asked.

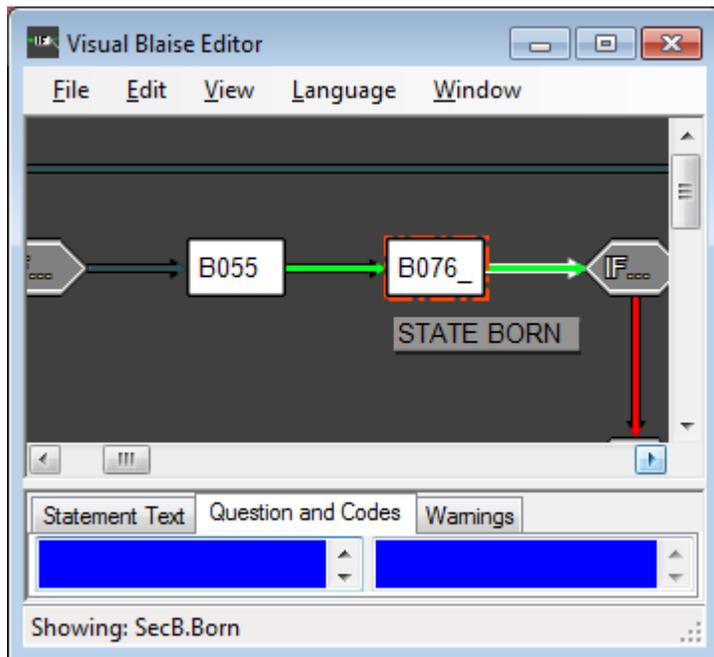
For more complicated sequences, a tool like Visual Blaise provides a visualization of the change and its impact on the block before actually making the change. Users are able to add, delete or change locations of a field and see what happens to the flow.

Getting Familiar with the Tool – Mouse and Keyboard Controls and Feedback

When the user clicks the mouse on a statement, it becomes selected and Visual Blaise offers some visual feedback:

- A flashing border is drawn around the selected statement.
- An asked question which is selected also displays its descriptor in a transparent gray box below the icon.
- A flashing highlight is drawn around the routing arrow extending from the statement, or from multiple arrows if the selected statement is a condition.

Figure 12: the highlighted border, routing arrow, and descriptor box of a selected question

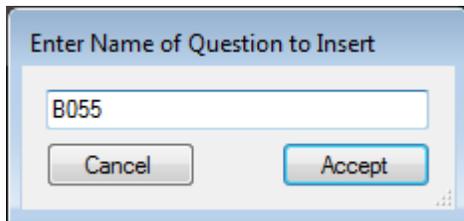


- Clicking and dragging a statement allows the user to see more clearly via the motion of the arrows where the connections lead when a segment is particularly busy.
- Pressing the right arrow key or the down arrow key moves the focus (and highlighting) from one selected statement to the next via the routing arrows. Pressing the left arrow key moves the focus to the previously selected statement (reversing the previous “history” of key presses, so it will not move further back than the selection starting point).
- Pressing the Enter key or double-clicking on a Block or Procedure statement will change the view to show the rules for that sub-Block or Procedure instead. Pressing the Backspace key or double-clicking on the “Back” return arrow icon at rightmost end of the rules will return the view to the parent Block or Procedure.
- Holding shift and pressing the right arrow key expands the selection to include multiple statements for the purpose of performing large moves or cuts.

Adding a Field

Via the View menu, Find function, or mouse and keyboard controls, the user can locate the part of an instrument where a change, such as adding a new question, needs to take place. Once at that location, right-clicking on a statement will cause a pop-up menu appear with options for inserting, deleting, or cutting statements. Insert will allow selection of what type of element you want add. Selecting a Question or Block type will prompt the user for the name of the new question or block.

Figure 13: pop-up prompt for field name



The new field will be inserted to the left of that element. (In the future you will be able to enter the question text and codes here, but for now only the Rules section is included in the output.)

Deleting a Field

Similarly, a selected question (or other statement) can be deleted via the same right-click menu. The warning tab will display references to the deleted field that need to be deleted or fixed (however, this is not comprehensive; warnings are still a work in progress at this stage).

Moving Elements Around

The following example illustrates a field move request that utilizes Visual Blaise's element moving capabilities. Suppose that new fields were inserted last wave and, after reviewing the data, the investigators would now like to move fields around for better flow and coverage. They decide to move B013_ to before B011_, which can be accomplished in Visual Blaise by clicking and dragging B013_ above B011_ and dropping it there. Since there is no logic dependent on B013_ , the dragged statement will be inserted to the left of B011_ and no warnings will be generated. Notice that all of the flow arrows automatically realign as appropriate for this change.

Figure 14: before moving question B013_ via drag and drop

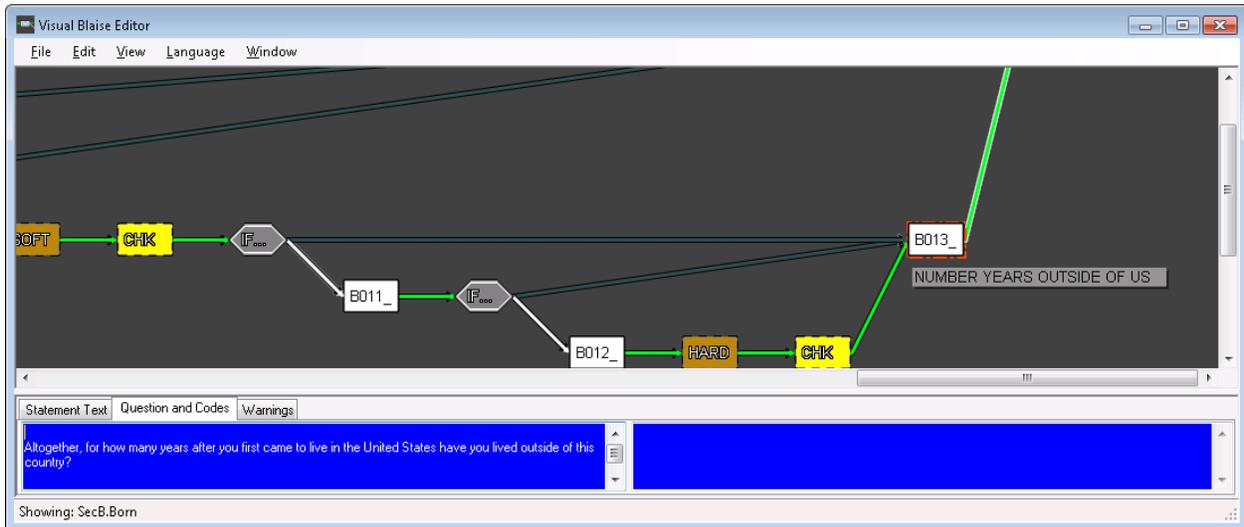
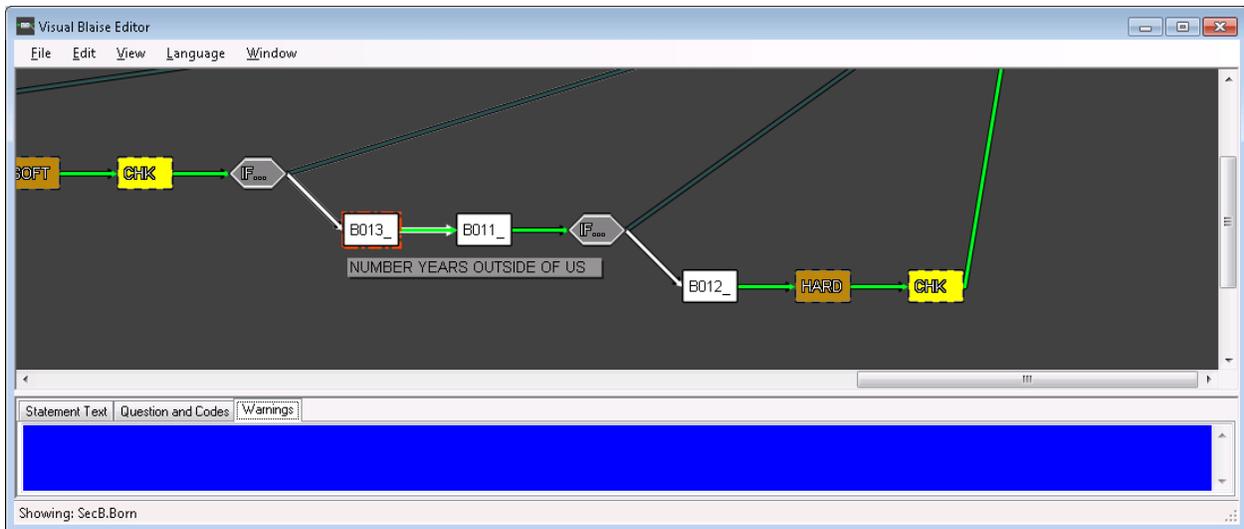


Figure 15: after moving question B013_ via drag and drop



More complicated moves might produce warning messages. If there are references to the element being moved in other statements passed over by the move, each reference should be reviewed and altered, if needed. For example, if a field is moved to a location after a conditional statement that references it, that reference most likely would need to be deleted. The warning messages are meant to alert the user to such contingencies.

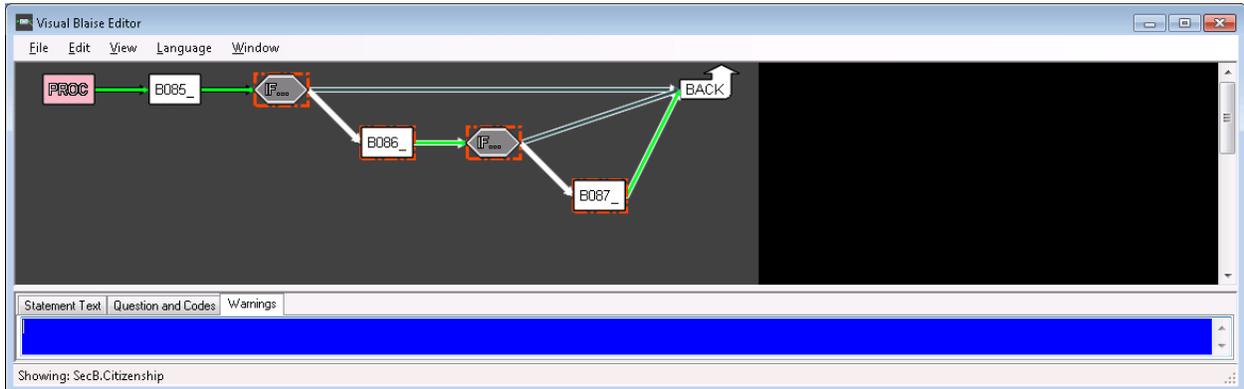
Example:

```
FieldA
if FieldA = yes then
  FieldB
endif
```

If Field A is moved to a location after Field B the conditional routing reference may need to be deleted or updated to something else.

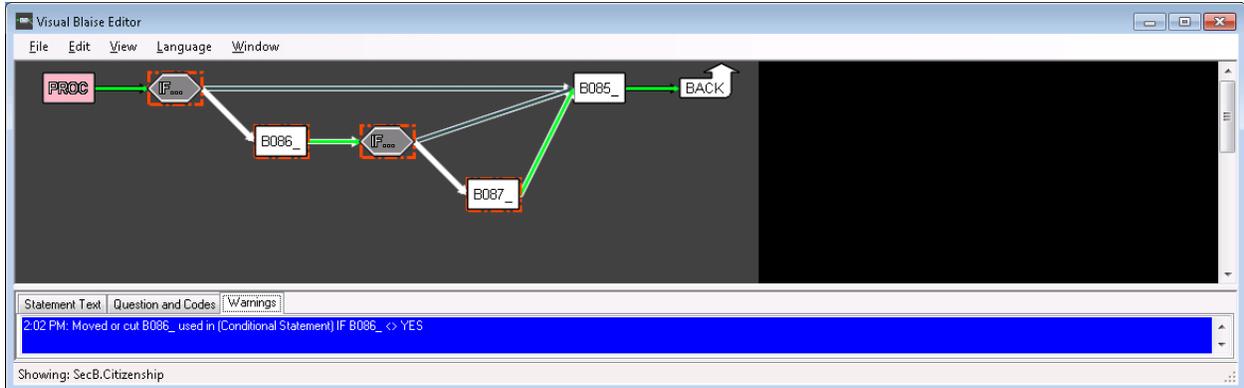
Another issue that arises during section redesign efforts is the need to move a large group of logic as a group. To select more than one element, first click on the statement you want to start with and then press shift-right arrow until all desired statements are selected.

Figure16: selecting multiple statements following the question B085_



Then, to move the selected group, right click on the group and select “cut.” Next, select the element before which the group should be inserted, right click and select paste.

Figure 17: multiple selected statements cut-and-pasted before B085_



In the above example, the Warnings tab reports that the far right IF condition in the moved group refers to the field that now comes after it. This flow will need to be fixed or deleted.

Producing and Reviewing Output

Now that the flow has been reviewed and updated, it is possible to output the changes into a Rules section readable by Blaise.

Selecting “Save” from the file menu offers two options: to save the flow to a Visual Blaise file or to save to a Blaise rules text file.

- Saving to a Visual Blaise file will save any changes to this point so you can retrieve and continue later.
- Saving to a Rules file will produce a text file with the logic that is represented in the Visual Blaise view. An option to save the Rules section of every block or only the currently viewed block is provided.

The output after the change from the above example would appear like this:

```
IF B085_ = YES THEN
  B086_.ASK
  IF B086_ <> YES THEN
    B087_.ASK
  ENDIF
ENDIF
B085_.ASK
```

Conclusion

We have only just finished this tool as a beta version and begun to use it in our design process. We are hopeful that this tool will allow investigators and coordinators to experiment with the design of a section and refine the flow to an optimal state more efficiently and with a better understanding of what they will be able to achieve by making each change. We believe this tool will also result in considerable time savings for the programmer, as the output can potentially be merged into the main instrument and, with a little “tweaking,” become the next wave’s rules.

Centralization and Regionalization at National Agricultural Statistics Service

Roger Schou, National Agricultural Statistics Service, USA

1. Introduction

The tides of change never seem to ebb in much of life. Changes at the National Agricultural Statistics Service (NASS) are no exception. NASS is the agency of the United States Department of Agriculture responsible for the nation's agriculture data. We use the typical modes of data collection including CATI, paper questionnaires for mail and field interviews, CAPI field interviews, and web self-administered instruments. Blaise is used at NASS for CATI data collection as well as one of two main editing systems for data collected in all modes. Our CASIC system utilizes Blaise, Manipula, ManiPlus, and Visual Basic .NET (VB.NET).

The office structure of NASS has undergone a very large change over the last year: regionalization. We are nearly complete in moving from forty-six field offices across the United States to twelve regional offices. The states that did not become a regional center will still have a presence person and a field interviewer coordinator, but the majority of the survey work will be completed in the regional field offices (RFOs). We still have our headquarters (HQ) in Washington, D.C., a research division in Virginia, and our National Operations Center (NOC) in St. Louis, MO. The NOC serves as a primary calling center for NASS with a capacity of up to 166 phone interviewers. As a result of the NOC becoming fully functional, we have closed two of our field office Data Collection Centers (DCCs). The majority of the CATI data collection is done at the NOC and our four remaining field office DCCs. The majority of the paper forms are processed at the NOC.

In 2010 at the International Blaise Users Conference (IBUC) XIII in Baltimore, MD, we reported on the successes and lessons learned from converting our first survey to a centralized environment. In 2012 at IBUC XIV in London, England, we had centralized another twenty-five surveys. We now have approximately fifty-five surveys in centralized Blaise.

2. Centralized Blaise Database and Tables at NASS

We continue to store our transactional data in a MySQL database. We are writing directly to the database from Blaise using BOI files. These files contain the connection information to the MySQL tables. We are using generic in-depth storage so all of our surveys are stored in the same format in the database. This allows us to have one Extract, Transfer, Load (ETL) program running to copy the necessary data from the transactional MySQL database to the analytical Redbrick database.

The generic in-depth storage in Blaise yields eight generic tables: BLAISE_DICTIONARY, BLAISE_ID, BLAISE_CASE, BLAISE_FORM, BLAISE_KEY, BLAISE_DATA, BLAISE_REMARK, and BLAISE_OPEN.

The BLAISE_DICTIONARY table keeps track of all of the surveys in the MySQL database. Anytime there is an instrument change (a change in the checksum), a new entry is created in this table. The change could be as minimal as data model name. This is how new instances are created of our surveys including our monthly and weekly surveys. We chose to use the survey folder name as the data model name, as it contains at a minimum a year, a month, if needed, and a day, if needed. This makes the data model names unique.

The paths to the externals also need to be update for a given survey. For a weekly or even monthly survey, manually changing the data model name and the paths of the externals for each survey period is too labor intensive.

Code to search and replace the name of the data model was put into place as the previous and current data model names are equivalent to the survey folder names and thus known.

The other issue with the weekly and monthly surveys is that the path to the externals changes with each survey period. To automate this, we put all of the external sections in their own INCLUDE files. The VB.NET code creates the INCLUDE files when a user clicks the button. These files are then included within the appropriate blocks of the instrument, so they are no longer hard-coded. Once the data model has been renamed and the external INCLUDE files have been created, the menu prepares the new instrument, and it is ready for the next survey period. Currently this is all in one button click of the menu. Once we are sure the process is bug-free, we are going to make it a CRON job that will run automatically.

Table 1. BLAISE_DICTIONARY

DMKEY: integer	Data model key (unique 1-up)
DATAMODELNAME: varchar(255)	Name of the data model
CHECKSUM: varchar(29)	Checksum of the data model
DPT: integer	Data partition type
BMI: varchar(255)	Path to the associated BMI file
BOI: varchar(255)	Path to the associated BOI file
SEARCHPATH: varchar(255)	Dictionary search path
ADDED: datetime	Date and time of adding
ADMINKEY: varchar(255)	Encrypted administrator password
COLLECTMODES: varchar(255)	<i>Not yet implemented</i>
CAB: blob(16777215)	Cabinet file

The BLAISE_ID table contains all of the block names and the field names within a given instrument. Blocks and fields are numbered independently: they each start at 1. We have replaced our Cameleon scripts with VB.NET code that makes use of the BLAISE_ID table. We create a mapping file for code data that is keyed from paper questionnaire linking the item code to a field in the Blaise dataset. For this, we use the field tag. We use the field description to store our variable names which are used by the ETL.

If it is simply a single field that needs the item code and variable name, assigning them is straightforward. However, if there is a block that is used more than once as a type and the fields within each instance of the block have different item codes and variable names, then the simplistic approach does not work. We have implemented what we call hash notation. If the field tag and field description are defined on the field using the block as its type, we use the hash mark (#) to separate the field tags and field descriptions. There is a one-to-one relationship between the field tags and field descriptions at this level and the fields within the block being used as the type. If the first field in the block does not have an item code or a variable name, we begin the field tag with a zero and the field description with a #. The VB.NET code then reads the BLAISE_ID table and for blocks that have field tags or field descriptions, the hash notation is deciphered and the appropriate field tag and field description are assigned to the corresponding fields within that block. The hash notation is then deleted from the block level.

In Example 1 the planted, harvested and production questions have item codes, and the amount and unit questions do not. The block is reused for corn and soybeans with the unique item codes and variable names defined outside of the actual block.

Example 1. Code Utilizing Hash Notation

```
BLOCK bCropBlk
  PARAMETERS
    piCropName : STRING
  FIELDS
    Planted "How many acres of ^piCrop were planted?" : INTEGER[9]
    Harvested "How many acres of ^piCrop were harvested?" : INTEGER[9]
    Amount "What was the total amount of ^piCrop produced?" : INTEGER[9]
    Unit "What was the unit produced?" : (bushels, tons, pounds, hundred "hundred weight")
    Produced : INTEGER[9]
  RULES
    Planted
    Harvested
    Amount
    Unit
    Produced.KEEP
    pCalcPrdctnProc(Amount, Unit, Produced)
ENDBLOCK

FIELDS
  Corn (530#531###370) "" / "CCRNXXPL#CCRNXXHV###CCRNXXPD" : bCropBlk
  Soybeans (600#763###227) "" / "CSOYXXPL#CSOYXXHV###CSOYXXPD" : bCropBlk
RULES
  ...
```

After running the VB.NET code to reassign the field tags and field descriptions, the Corn.Planted field would have 530 as the field tag and CCRNXXPL as the field description. Soybeans.Harvested would have 763 as the field tag and CSOYXXHV as the field description. The Corn.Unit would not have a field tag or a field description. The one thing that is forfeited by using this structure is the ability to jump to a field tag within the instrument.

For fields that are defined by arrayed blocks with repeated item codes and similar variable names, the field tags and field descriptions are actually attached to the fields within the block being used as a type. When an array is encountered with field tags or field descriptions, then the mapping file puts a keyword into the array element number and adds an extra parameter which is a string with the word "ARRAY." The Manipula that reads in the data detects the word and then substitutes the keyed table and row number on the raw data file into the array element keyword.

The variable names must also be unique for the ETL. So VB.NET code sees the array in the BLAISE_ID table and the field description within the block. It will then take the array element number from the Blaise field name using the first digits as a table number and the last two digits as the row number. The new resulting variable name becomes VarName_TableNum_RowNum. The arrays in the Blaise instrument must be assigned with this variable naming schema in mind.

Once the field tags and field descriptions have been updated in the BLAISE_ID table, our VB.NET code, that replaced our old Cameleon code, creates the files needed for reading code data into Blaise.

Table 2. BLAISE_ID

DMKEY: integer ID: integer TYPE: varchar(1)	Data model key (unique 1-up) Id of B/F (unique 1-up for each) Indicates type of ID (B=block, F=field)
NAME: varchar(255) TABLENAME: varchar(255) DATATYPE: varchar(255) DECIMALS: integer FIELDSIZE: integer EMPTY: varchar(3) DONTKNOW: varchar(3) REFUSAL: varchar(3) ARRAYINDEX: integer MININDEX: integer MAXINDEX: integer MINVALUE: integer MAXVALUE: integer ISSET: varchar(3) ISARRAY: varchar(3) ISTABLE: varchar(3) ISEMBEDDEDBLOCK: varchar(3) FIELDTAG: varchar(255) QUESTIONTEXT: varchar(255) DESCRIPTIONTEXT: varchar(255)	Fully qualified name of the B/F MySQL table name containing B/F B/F type as defined in the instrument Number of decimals defined for the B/F Length of the B/F YES/NO: Is EMPTY allowed for B/F? YES/NO: Is DK allowed for B/F? YES/NO: Is RF allowed for B/F? Array index for B/F (or -1) Minimum value for array index Maximum value for array index Minimum value for B/F (0 for string) Maximum value for B/F (0 for string) YES/NO: Is B/F and enumerated set? YES/NO: Is B/F an array? YES/NO: Is B/F a table? YES/NO: Is B/F an embedded block? Field tag for B/F (null if not defined) Question text for B/F Description text for B/F

The BLAISE_CASE table contains the JOINKEY values for each record in the survey.

Table 3. BLAISE_CASE

JOINKEY: integer DMKEY: integer	Integer identifying a record (unique 1-up) Data model key (unique 1-up)
KEYVALUE: varchar(255)	Internal culture independent value of the primary key

The BLAISE_KEY table contains all of the primary and secondary keys as defined in an instrument for a survey. The BEGINSTAMP value has to be unique in combination with JOINKEY and DMKEY.

Table 4. BLAISE_KEY

JOINKEY: integer DMKEY: integer BEGINSTAMP: datetime	Integer identifying a record (unique 1-up) Data model key (unique 1-up) Begin time of the period of validity of a particular record. This column is used for versioning.
KEYNAME: varchar(255) KEYVALUE: varchar(255)	Name of the key as defined in instrument Internal culture independent value of the primary key

The BLAISE_FORM table contains the status information about each record. The table also contains information about collection mode and the data entry behavior being used when the record was stored.

Table 5. BLAISE_FORM

JOINKEY: integer DMKEY: integer BEGINSTAMP: datetime	Integer identifying a record (unique 1-up) Data model key (unique 1-up) Begin time of the period of validity of a particular record. This column is used for versioning.
ENDSTAMP: datetime	End time of the period of validity of a particular record. The ENDSTAMP of newly inserted record will have the predefined value of '99991231

STATUS: tinyint	00:00:00'. Form status of the record (1=Clean, 2=Suspect, 4=Dirty, 8=NotChecked)
COLLECTIONMODE: integer	<i>Not yet implemented</i>
DATAENTRYBEHAVIOUR: integer	Data entry behavior during the time the record was written (0=bldbUncheckedEditing, 1=bldbCheckedEditing, 2=bldbFreeInterviewing, 3=bldbStrictInterviewing).
ERRORCOUNT: integer	Number of errors in current record
REMARKCOUNT: integer	Number of remarks in current record
DONTKNOWCOUNT: integer	Number of DK answers in current record
REFUSALCOUNT: integer	Number of RF answers in current record
STREAMSTATUS: varchar(1)	Status of the stream that has been stored in STREAMDATA. (Uppercase O=out of sync).
STREAMDATA: blob(16777215)	Binary stream of current record

The **BLAISE_DATA** table contains the data for the records in the survey.

Table 6. BLAISE_DATA

JOINKEY: integer	Integer identifying a record (unique 1-up)
DMKEY: integer	Data model key (unique 1-up)
BEGINSTAMP: datetime	Begin time of the period of validity of a particular record. This column is used for versioning.
FIELDID: integer	ID of the field to which the data belongs
STATUS: tinyint	Status of the field (1=Unprocessed, 2=Response, 4=DK, 8=RF)
STRINGDATA: varchar(255)	Answers to fields with string data type
INTEGERDATA: integer	Answers to fields with integer or enumerated type
FLOATDATA: double	Answers to fields with real data type
DATETIMEDATA: datetime	Answers to fields with date and time data type

The **BLAISE_REMARK** table contains the remarks left on fields for a survey.

Table 7. BLAISE_REMARK

JOINKEY: integer	Integer identifying a record (unique 1-up)
DMKEY: integer	Data model key (unique 1-up)
BEGINSTAMP: datetime	Begin time of the period of validity of a particular record. This column is used for versioning.
FIELDID: integer	ID of the field to which the remark belongs
REMARKTEXT: text(16777215)	Text of the remark

The **BLAISE_OPEN** table contains the answers to any **OPEN** fields with a response.

Table 8. BLAISE_OPEN

JOINKEY: integer	Integer identifying a record (unique 1-up)
DMKEY: integer	Data model key (unique 1-up)
BEGINSTAMP: datetime	Begin time of the period of validity of a particular record. This column is used for versioning.
FIELDID: integer	ID of the field to which the remark belongs
STATUS: tinyint	Field status of the open field (1=Unprocessed, 2=Processed, 4=DK, 8=RF)
OPENTEXT: text(16777215)	Answer text of the open field

In addition to these standard eight generic tables that Blaise creates, NASS has added a few flat data tables to help manage the processes as well as increase performance.

The CASIC_SURVEYINFO table contains survey-level information such as the instrument name, folder name, BOI file name, a number of indicators, and some starting and ending dates. Our menu system makes extensive use of this table.

Table 9. CASIC_SURVEYINFO

SAMPLE_ID: integer	Integer identifying a survey
YEAR_: integer	Four-digit year of the survey
MONTH_: integer	Month of the survey
DAY_: integer	Day or Week of the survey
INST_NAME: varchar(15)	Name of the .bmi file
FOLDER_NAME: varchar(15)	Folder name for the survey
BOI_NAME: varchar(15)	Name of the .boi file
PRODUCTION_IND: integer	Indicates if a survey is in production (0=beta, 1=production, 9=in production but deactivated)
SHELL_IND: varchar(4)	Indicates which NASS shell code was used (L7=list, MF7=multiple frame, C7=Census)
NONOPDOM_IND: integer	Loosens some of the restrictions on records
CAMELEON_IND: varchar(1)	Indicates which Cameleon scripts are to be used (I=code data input, O=code data output, B=both code data input and output, N=neither)
REGIONAL_IND: integer	Indicates if survey is regionally processed (0=not regional, 1=regional survey, 2=regionally-processed survey)
IDAS_FOLDER: varchar(15)	Folder name for data fed to analysis
DM_KEY: integer	Corresponding DMKEY in the BLAISE tables. Used by ETL to know which surveys to transfer to analytical database
SMETA_KEY: varchar(8)	Number identifying a survey in Questionnaire Repository System
ALERT_IND: integer	Indicates that alerts will be used to check in forms keyed at the National Processing Center (NPC)
NPC_IND: integer	Indicates NPC will be involved in survey
SURVEY_TYPE: varchar(2)	Two-character survey type for Census surveys
SURVEY_ABBREV: varchar(3)	Three-character survey abbreviation for Census surveys
SURVEY_DESC: varchar(50)	Survey name from Metadata Repository System
BETA_START_DATE: datetime	Date survey becomes available on Beta
COL_START_DATE: datetime	Data collection start date on Production
COL_END_DATE: datetime	Data collection end date on Production
EDIT_END_DATE: datetime	Editing end date on Production
SDATE: date	Date associated with survey
SURVEY_ID: tinyint	ELMO survey id
PERIOD_ID: tinyint	ELMO classify period
NEWADDCATEGORY: tinyint	Newly added record category to indicate if survey allows newly added records
SAMPLE_TYPE: tinyint	Sample type (e.g. aggregated, independent, area, census, etc.)
FREQUENCY: tinyint	Survey frequency (e.g. annual, quarterly, monthly, weekly, etc.)
PARTNER_INCL: tinyint	Indicator to include or exclude partners

The CASIC_FAT table is used to control access to certain groups of records to an authorized state, data collection center, estimation center, or regional field office.

Table 10. CASIC_FAT

SAMPLE_ID: integer YEAR_: integer MONTH_: integer DAY_: integer FO_FIPS: integer	Integer identifying a survey Four-digit year of the survey Month of the survey Day or Week of the survey State in the survey
SURVEY_DESC: varchar(50) DCC_FIPS: integer EC_FIPS: integer ADD_FUNC: integer BETA_INIT_PREP: integer BETA_INITIALIZED: integer INIT_PREP: integer INITIALIZED: integer NAME_STAMP: varchar(10) DATE_STAMP: datetime RFO: integer DCC_RFO: integer EC_RFO: integer	Survey name from Metadata Repository System State responsible for collecting the FO_FIPS state data State responsible for editing the FO_FIPS state data Used for multi-state DCC or EC functionality Indicates the input sample files for this state have passed the preparation phase on Beta Indicates this state has been initialized on Beta Indicates the input sample files for this state have passed the preparation phase on Prod Indicates this state has been initialized on Prod Login name of the person who updated this line in the table Date of the update of this line in the table Region code for the FO_FIPS Region responsible for collecting the FO_FIPS state data Region responsible for editing the FO_FIPS state data

The CASIC_MANAGEMENT table has several key fields that also appear in all of our instruments. This table has been indexed on these fields, which increases the performance of the Blaise instruments. It allows the record filters in Manipula and the .BOI files to be much more efficient because the database being accessed is indexed.

Table 11. CASIC_MANAGEMENT

DMKEY: integer LFINFO_STATE: integer LFINFO_ID: integer LFINFO_TRACT: integer LFINFO_SUBTRACT: integer	Data model key (unique 1-up) State field (part of primary key) ID field (part of primary key) Tract field (part of primary key) Subtract field (part of primary key)
BEGINSTAMP: datetime MANAGEMENT_DCC_FIPS: integer MANAGEMENT_EC_FIPS: integer MANAGEMENT_FO_FIPS: integer MANAGEMENT_BATCH: integer MANAGEMENT_PROCESSSWITCH: integer LFINFO_PID: integer LFINFO_COUNTY: integer MANAGEMENT_RFO: integer MANAGEMENT_DCC_RFO: integer MANAGEMENT_EC_RFO: integer LFINFO_XSTATELINK: integer	Begin time of the period of validity of a particular record. This column is used for versioning. State responsible for data collection State responsible for editing State to whom the record belongs Batch number to which the record belongs NASS process indicator (1=CATI Complete, 2=Default, 3=Edited, 4=Sent) Another ID-type field County field Region to whom the record belongs Region responsible for data collection Region responsible for editing Field used to link records

The CASIC_EVENT_LOG table tracks the activity on the CASIC Menu. It has an entry consisting of the person's login name, location, the date and time a button was clicked, and the survey involved. There is another entry logging the end of the process. This was created as a debug tool to indicate what procedures were running when we saw any undesirable issues arise.

Table 12. CASIC_EVENT_LOG

DMKEY: integer	Data model key (unique 1-up)
SURVEY: varchar(50)	Survey name from Metadata Repository System
YEAR_: smallint	Four-digit year of the survey
MONTH_: tinyint	Month of the survey
DAY_: tinyint	Day of the survey
WEEK_: tinyint	Week number of the survey
STATE: tinyint	State where the button was clicked
LOGIN_NAME: varchar(10)	Login name of person who clicked the button
PROCESS_NAME: varchar(50)	Name of the process running
EVENT: varchar(10)	The event of the process: BEGIN/END
TIMESTAMP: timestamp	Date process began or ended

3. Centralized Blaise Concept

The concept of Centralized Blaise is a simple one. There is one dataset, centrally located, with controlled access for all who need it. Implementing this concept was a bit more involved. Blaise generic in-depth data storage takes care of storing the Blaise data. The actual controls had to be developed by NASS. We only wanted to provide access to those who needed access. Identifying these needs was fairly straightforward at first, but then the lines began to get fuzzy.

The two obvious functions are data collection and interactive editing. Based on the entry in the CASIC_FAT table, each record is assigned a DCC_FIPS to give the data collection responsibility to a state, and an EC_FIPS to give the data editing responsibility to a state. Using record filters, the data can be logically separated. Now that the data is centrally located, HQ can play a much more interactive role. Special reports were created specifically for people in HQ, so an up-to-date picture of how a particular survey is progressing is available. Another benefit is the ability to look at the exact record of a state with which they may be having issues. It also allows someone in HQ to edit records for states.

Typically, the EC_FIPS state would be responsible for reading out the data (until WIP2, our analytical database, is fully functional), but since the data is centrally located, it made more sense for one readout at a national level. So the menus were created to do just that. Every rule seems to have exceptions, and the readout process was no different. For some surveys, the states wanted the readout control back, so they didn't have to wait on the whole country to get the data in which they were interested. This was accommodated by the menu script. An indicator was introduced to identify the party responsible for the readout. This is just one example of a number of tweaks made along our way.

The infrastructure is not yet ideal at NASS. We are 95% virtual, using Citrix. The country is split into two halves, east and west. The Blaise data servers, the MySQL database server, and the eastern Citrix servers are currently in HQ. The western Citrix servers are in Kansas City. This makes the distance for eastern Citrix users quite short; whereas, the distance for the western Citrix users is quite long. There is a noticeable difference in performance between the eastern and western users. Plans are to move all the servers to Kansas City in the middle of the country. This will make everyone equidistant and we are anticipating a significant performance improvement especially for the western users.

Figure 1 illustrates the current scenario for an eastern Citrix user. Response times are good because all of the servers, including the Citrix server, are very close physically.

Figure 1. Current Eastern Citrix Scenario

Workstation ← Local Area Network (LAN) → Blaise Data Server ← Local Area Network (LAN) → MySQL
(1 ms latency) (1 ms latency)

Figure 2 illustrates the reason we have users on western Citrix that are suffering from the poor performance. The 30-millisecond latency is thirty times slower than the eastern Citrix performance when sending something from the workstation to the Blaise data server and thirty times slower when receiving something back from the Blaise data server.

Figure 2. Current Western Citrix Scenario

Workstation ← Wide Area Network (WAN) → Blaise Data Server ← Local Area Network (LAN) → MySQL
(30 ms latency) (1 ms latency)

Figure 3 illustrates our optimal server solution when they are all located in Kansas City. We don't anticipate much of an increase in performance, if any, for the eastern Citrix users, but the western Citrix users should see dramatic increases.

Figure 3. Future Eastern & Western Citrix Scenario

Workstation ← Local Area Network (LAN) → Blaise Data Server ← Local Area Network (LAN) → MySQL
(1 ms latency) (1 ms latency)

The computer facility in Kansas City will also provide us with a more stable environment for the hardware. There is also staff dedicated to maintaining that environment.

4. Hybrid Surveys

We have a small number of surveys that have not made it through the entire conversion from decentralized to centralized. These tend to be our high-profile surveys with a very short data collection window. One night lost for calling, could derail the estimate due date. The hybrid survey approach was developed to handle these few surveys until the comfort level of moving them to fully centralized is acceptable.

The hybrid survey approach consists of the CATI data collection portion of the survey cycle to be decentralized in a typical Blaise dataset. Once data are collected, the completed forms are copied to the centralized dataset in MySQL. When the data arrive into the central dataset, the remaining survey processes are completed in the centralized environment.

The management of the hybrid surveys is much more disjointed than the centralized surveys. As our confidence has risen, plans to fully centralized these few surveys is underway.

5. Regionalization at NASS

Recognizing that resources are fewer and fewer, and the need to do more with less, NASS began plans to reorganize our field office structure. We have moved from forty-six field offices to twelve regional field offices. This is yet another change that has had a large impact on our CASIC system. We had already made the leap from decentralized surveys with up to forty-six physically separated datasets to one logically separated centralized dataset. All of the filters that we had built were dealing with data collection centers, estimation centers, and individual field offices. Now we have introduced regional field

office filters. Three additional fields have been added to the shell code of our instruments to identify the region where the record belongs, the region responsible for data collection, and the region responsible for editing the record.

Since not all of our field office staff have relocated, the CASIC system has to detect where the user is sitting, and to which region they belong. We pick up the user's location from their Windows AD group membership. We then have a procedure to determine their region. The filters are built on the fly to filter the centralized dataset down to just their region. It is much more efficient to use the record filters rather than using an if-statement in the manipulate section. When a record filter is used, only the records that qualify for the filter are actually read by Manipula. If the control logic is done via an if-statement in the manipulate section, then all of the records will be read and evaluated. In some cases, this could mean a difference of reading 10,000 to 20,000 records and reading 200,000 records.

As stated earlier, there are always exceptions to the rule. For example, not all of our surveys have made the transition to being processed regionally. So our system has an indicator in the CASIC_SURVEYINFO table to make this determination. Since the record filters are built within the menu code, which is written in VB.NET, the menu can react to this indicator and build the appropriate record filter. The filters are read by the Manipula programs using an INCLUDE file. This allows us to have one Manipula program that is independent of how the survey is being processed.

6. Conclusion

NASS still has systems slated to move into a more centralized environment that will communicate with our CASIC system. As these systems are developed, we will need to continue to adapt and modify our system. Some of the bridges that we have built will be replaced with more efficient methods and direct links to other systems at NASS.

The waves of change will continue to reshape our current system. We can only hope that through open communication and collaboration with our fellow NASS developers, the emerging systems connect seamlessly with the new system designs developed up to this point. We also hope to be able to sunset some of our legacy systems, as our new systems mature and prove themselves. Blaise continues to play an important role at NASS. It is our only CATI software and will make up one of our two edit systems, as we are phasing out one of our old legacy edit systems.

Blaise 5 Paradata Requirements

*Rebecca Gatward, Lisa Wood, Patty Maher and Gina-Qian Cheung
Survey Research Center, University of Michigan*

1. Introduction

Paradata are captured during the survey process (Couper, 1998) and are a valuable source of information about the quality of the survey data and in helping us understand, monitor and inform decision making throughout the survey lifecycle. As has been widely documented, there are various types of paradata. Those which are linked directly to the administration of a survey instrument are usually collected automatically through CAI software, such as Blaise. The source of this type of paradata are primarily audit trails and recordings of part or whole interviews using Computer Assisted Recorded Interviews (CARI), this category of paradata can also include server and client side data for web collection. Another type of paradata are collected or stored in survey and sample management systems. This type of paradata describe the actions and processes involved in data collection, it often includes data about contact attempts, the interview, observations from the interviewer, the sampling frame and respondent related administrative procedures, such as compliance to complete consent forms and payment of incentives.

During the last 15 years, paradata have become an essential tool in survey management and as O'Reilly (2009) describes many survey organizations have developed or enhanced existing management systems with paradata as a central feature.

In 2011 a working group was formed with the purpose of specifying the Blaise community's paradata requirements for Blaise 5 (the new build of Blaise developed by Statistics Netherlands). The group comprised representatives from Blaise User Corporate license holders, including the Survey Research Center at the University of Michigan and Mathematica Policy Research. The requirements paper was presented and discussed at the Blaise Corporate License User Board meeting in January 2013.

The main focus of this paper is to detail the paradata requirements as specified by the user group which, given the composition of the group, are primarily for instrument paradata. However, the paper will also provide some practical examples to illustrate how paradata can be used to inform decision making throughout the survey process and assist in fieldwork monitoring.

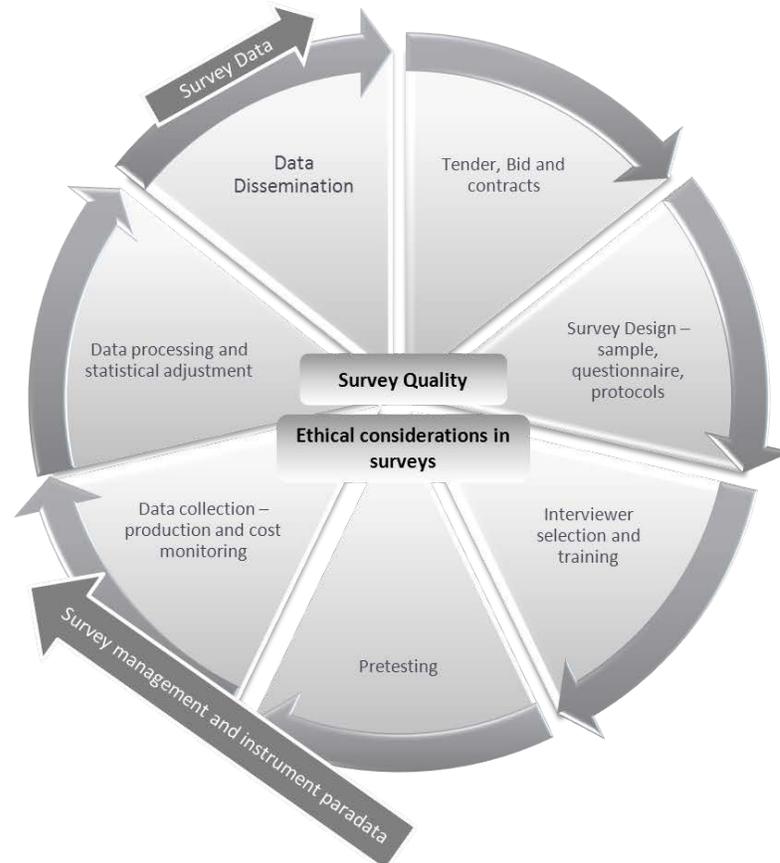
2. Use of paradata

Paradata are collected and used throughout the survey lifecycle – from tender or budgeting , through the design and piloting phase, during data collection and when writing final documentation. The primary uses of paradata are to:

- monitor data collection and other study processes,
- identify methodological error in design,
- reduce survey error,
- improve the quality of survey processes and products,
- aid analysis when reviewing survey measures,
- and improve organizational quality.

Figure 1 illustrates that although specific types of paradata are produced at certain stages of the survey lifecycle, they are used for multiple purposes throughout the survey process.

Figure 1 – Use of paradata throughout the survey process



The following sections provide some examples of the way paradata can be used as a survey and sample management tool, how they contribute to the quality assurance process and the type of analysis that can be undertaken using paradata.

- As a **Survey Management tool**, paradata can be used to:
 - Monitor interviewer’s skills, experience and appropriateness to be staffed on a project, for example, interviewer location, whether they are bilingual, the types of data collection they have undertaken and their success at refusal conversion on current or previous studies.
 - Determine the length of time required to administer a complete interview, an interview session (where the interview is not completed in one session), block of questions or individual questions. This analysis could also be carried out to provide comparisons by mode, by characteristics of the respondent or interviewer. A case study of this type of analysis is provided in a paper by Devonshire (2013).
 - Monitor the data collection progress for each sampling unit (within or across modes), some examples include,
 - contact attempts - time, day and date of contact, mode and outcome,
 - number of times the survey has been started and suspended,
 - point at which a mode switch occurred,
 - or the question at which an interview was suspended or abandoned.

- Identify areas in the questionnaire that are “trouble-spots” through the analysis of key stroke data, such as the frequency of the use of remarks by question or the use of short cut keys assigned to access any question specific help screens.
 - Monitor respondent payments, for example, amount paid to respondents, timing of the payment and total cost. This data could also identify the characteristics of respondents who do not cash their payment or methods that could reduce delays in paying respondents.
 - Track communication with respondents and the effect of any interventions.
 - Finally, paradata can be used to examine progression through the survey, for example, identify if certain questions are revisited multiple times.
- For **Sample Management**, paradata can help inform the following types of key decisions:
 - Planning the number of interviewers required to complete data collection within the desired field period and the assignment of cases to interviewers.
 - Identifying respondent characteristics and contact strategies, for example, the best time to make contact to respondents based on past waves or by characteristics of the household.
 - Determining the cost of the survey (cost per unit) and establishing best collection strategy to move forward.
 - Target collection effort based on stratum response rate.
 - Identifying follow-up strategies based on collection priority or weighting assigned to specific respondents.
 - To identify the need, type and success of a responsive design strategy.
 - Depending on the type of interviewer observations collected, observation data can be used to monitor if interviewers are implementing appropriate strategies for certain situations, identify better listing protocols, reduce coverage error caused by listing or reduce non-response rates by improving tracking methods.
 - Verify interviewer either listed or interviewed at the correct address using GPS coding.
- Paradata can be used as a **Quality Assurance** tool in the following ways:
 - To evaluate interviewer performance – for example, an analysis of the audit trails may detect or support suspicions of interview falsification. CARI files provide recordings which can be used to carry out detailed assessments of interviewers interviewing skills.
 - Where necessary, data can be recovered using audit trails or provides the ‘event history’ of an interview.
 - During testing and can facilitate bug identification, diagnosis and reporting - for example, analysis of paradata can help identify if the source of a reported issue is a specific system, survey instrument or the user. Paradata can also be used to monitor system performance and track the incidence of an issue.
 - To track interaction of the survey instrument with external systems and diagnosis system issues related to this interfacing.
 - Auditing access to survey data (security management).
 - **Adhoc analysis** – paradata can also be used to investigate specific research questions or to explore an issue in greater depth. One key factor to consider is that, although it is useful to collect as much paradata related to the administration of the Blaise instrument as we can, we should aim to do so without impacting the efficiency of the data collection process. Paradata have been analyzed to identify:

- Respondent response pattern and response order effects.
- CAI design improvements – this is particularly useful in the piloting stage of a questionnaire or whilst testing individual questions.
- Respondent preferred data collection method.
- The impact of processes surrounding data collection on the survey outcomes. For example, interviewer characteristics, mode and device used of data collection and the recording process.
- To examine the actions on each page of the instrument and the impact these actions have on survey outcomes. For example, on a web self-administered questionnaire; did the respondent have to scroll? Were questions answered in the order presented (if a grid is used)? Did the respondent use a mouse or keyboard? What was the total number of keystrokes?

3. Paradata requirements

The following is a summary of the requirements as specified in the Blaise 5 paradata white paper. These requirements are specific to paradata related to the administration of the CAPI and Blaise IS instruments, those for the Blaise CATI Survey Management System are not included.

3.1 General requirements

Users requested the following requirements which relate to the collection and storage of the paradata rather than specific elements of paradata.

- First, and most importantly, the level of detail included in the current Blaise ADT files should be retained.
- Security of the paradata is a concern.
 - Paradata should be stored securely (i.e., secure database or encrypted text file).
 - Users should be able to ‘turn-off’ collection of various paradata elements.
 - It should be possible to decouple or store the paradata separately but still be able to merge it with the questionnaire data for analysis.
- The data structure must be consistent across all modes of data collection although the data collected may vary by mode.
- Users should be able to define key project-specific variables to include in the paradata such as stratum, project name, or user-defined status or outcome variables (i.e., variables preloaded into Blaise or calculated within Blaise). These variables should also be read- and write-able in Rules, Manipula and API.
- It would be useful for Blaise 5 to include a ‘paradata viewer’ for all modes: The paradata viewer should provide a survey summary viewer for information at the survey level available at run-time for all modes. This can further be enhanced to provide customizable reports on the survey progress.
- Current and Historical Paradata Viewer. User should be able to view the current paradata in list view for all modes in one interface. Clicking on a selecting case should provide users with the historical paradata transactions associated to the case.

3.2 Specific requirements for Computer Assisted Personal Interviewing

Users determined that instrument paradata for Computer Assisted Personal Interviews (CAPI) should be collected for each respondent at the variable, page, block, session and interview level. The specific elements users requested are detailed in an appendix to this paper (Appendix A). The following are paradata which are required at an interview or specific part of the interview.

- Mode of data collection at a page or item level as well as at a session or form level.
- Geographic information about location of the respondent (e.g., GPS coordinates).
- Paradata should include CARI specific information (e.g., log of questions recorded, name of sound file(s) and/or screen shot(s)).
- Paradata should include information about the CARI set up or recordings (e.g., log of questions recorded, name of sound file(s) and/or screen shot(s)).

3.3 Specific requirements for web data collection

The following is a summary of the paradata required for surveys administered using Blaise IS. Again, the detailed elements required by users are listed in the appendix to this paper (Appendix A).

- When survey data is collected via the web, paradata about the environment in which the survey was administered should be captured - for example, the type of device, operating system, browser, connection speed and screen or browser size.
- Paradata for web surveys should capture data from both the server and client side:
 - Server side includes submissions to server, page-level times and break-offs.
 - Client side involves embedded java script code that captures user actions like changed responses, response latencies, keystrokes, and even mouse movement - for example, mouse coordinates every 'nth millisecond, details about whether and if so how the keyboard and scrolling keys were used and whether horizontal or vertical scrolling used.

4. Conclusion

As is evident from the user requirements and their extensive use throughout the survey lifecycle, paradata continue to be important, especially as mode of data collection impacts quality and cost. Consistency in format of paradata and the ability to turn on and off various elements will help tailor the analysis and usefulness of paradata.

Paradata will continue to be a central part of survey collection operations, the further development of its use and the developments or efficiencies gained through the analysis of paradata are something we can share amongst the Blaise community.

References

- Couper, M. (1998). Measuring survey quality in a CASIC environment. *In proceedings of the Section on Survey Research Methods of the American Statistical Association.*
- Devonshire, J. (2013). 'Adding Business Intelligence to Paradata: The Blaise Audit Trail'. *Presented at the 15th International Blaise Users Group Conference, Washington, DC, USA.*
- O'Reilly, J. (2009). 'Paradata and Blaise: A review of recent applications and research'. *Presented at the 12th International Blaise Users Group Conference, Riga, Latvia.*

Appendix A

Blaise Paradata – Specific Elements

Specific data elements currently used by Survey Research Center for Blaise and BlaiseIS surveys are listed in tables below. In addition to the current elements, the following additional elements should be included:

- Mode of Administration (page level)
- Location of Interview (session level) – IP Address and/or GPS coordinates
- Operating system and device used (session level)
- More detailed information for keystrokes (see number 28 listed below)

BlaiseIS Paradata (this is collected for each page visited)

#	Variable	Type	Size	Description
1	PostID	Num	8	Project-specific variable; preloaded
2	ServerName	Char	7	Name of web server
3	ID	Num	8	Primary Key
4	HashID	Char	8000	
5	PAGESTARTTIMESTAMP	Char	30	Date/Time the current page was started
6	PREVPAGETIMESTAMP	Char	30	Date/Time the previous page was started
7	PREVPAGELENGTH	Num	8	PREVPAGETIMESTAMP - PAGESTARTTIMESTAMP
8	SESSIONID	Char	20	Unique key for each web session
9	PRIMARYKEYVALUE	Char	200	Project SampleID; preloaded
10	VPROJECTID	Char	40	ProjectID; preloaded
11	MODE	Char	10	Mode Used (web SAQ or interviewer-administered)
12	SCREENSIZE	Char	50	Screen Size as "width, height"
13	BROWSERSIZE	Char	50	Browser size
14	JAVASCRIPTENABLE	Char	10	JavaScript enabled = true in the browser
15	CONNECTIONSPEED	Char	20	A value of bytes/second throughput, calculated
16	SURVEYNAME	Char	50	Name for the survey
17	BROWSER	Char	500	Browser name
18	CURRENTPAGENO	Num	8	Interview Page: Cstr(SessionState.StoredPageIndex)
19	PREVPAGENO	Num	8	Like CurrentPageNo, filled in by Blaise IS ASP
20	VERSIONDATE	Char	20	Version date of survey
21	VERSIONTIME	Char	20	Version time of survey
22	ACTION	Char	20	Values are: InterviewStarter, Next, Other, Previous, Start, Submit
23	PREVPAGEQUESTIONS	Char	32767	Like PageQuestions, filled in by Blaise IS ASP
24	PREVPAGEANSWERS	Char	32767	Like PageAnswers, filled in by Blaise IS ASP
25	CURRENTPAGEQUESTIONS	Char	32767	List of questions on current page, ^ delimited
26	CURRENTPAGEANSWERS	Char	32767	List of responses on current page, ^ delimited
27	SUBMITSTATUS	Char	20	Values are Completed or Missing (if "Completed" then Action="Submit")

28	FLASHENABLE	Char	10	Flash enabled
29	OTHERPLUGINSENABLE	Char	10	Other plugins enabled
30	KEYSTROKE	Char	32767	information added about client-side events listed in order that they occur on the page, (i.e., javascript that picks up keystrokes on the page); currently includes data indicating if scrolling occurred (H=Y/N or V=Y/N), but no scrolling details, includes information about use of keyboard vs. mouse, but does not include mouse-click x/y coordinates

ADT Database

The following tables and data elements are compiled from Blaise ADTs into a SQL database;

- tAdtField (this comes directly from the ADT files)

Field Name	Data Type	Size	Description
ProjectID	Text	30	Unique project identifier
CaseID	Text	30	Sample ID within a project
FldSeq	Long Integer	4	Sequential counter; increments by one for each ADTrow
MetaName	Memo	-	Name of Blaise data model
MetaTime	Text	200	Date/Time Blaise data model was created
UserID	Text	30	Interviewer ID for this particular instrument entry
FldName	Memo	-	Blaise long field name
BlockName	Text	100	Blaise block name
FormVstNum	Long Integer	4	Instrument visit number; increments by one for each entry
FieldVstNum	Long Integer	4	Field visit number; increments by one for each entry
PrevLeaveLineNo	Long Integer	4	Previous field Blaise line number
EnterLineNo	Long Integer	4	Current field entered Blaise line number
LeaveLineNo	Long Integer	4	Current field exited Blaise line number
ISLEAVEFORM	Long Integer	4	Interviewer exited Blaise at this field (Yes/No)
EnterDate	Text	30	Date interviewer entered the Blaise field
EnterTime	Text	30	Time interviewer entered the Blaise field
LeaveDate	Text	30	Date interviewer left the Blaise field
LeaveTime	Text	30	Time interviewer left the Blaise field

Field Name	Data Type	Size	Description
Fld_Time	Text	30	Time (sec) spent within the Blaise field
Fld_SS	Text	30	Time (sec) spent within the Blaise field (different format)
Fld_Time_Mss	Long Integer	4	Time (milisec) spent within the Blaise field
Btw_Time	Text	30	Time (sec) spent between last Blaise field and this one
Btw_SS	Text	30	Time (sec) spent between last Blaise field and this one
Btw_Time_Mss	Long Integer	4	Time (milisec) spent between last Blaise field and this one
Adj_Time	Text	30	Fld_Time + Btw_Time
Adj_SS	Text	30	Fld_Time + Btw_Time
Adj_Time_Mss	Long Integer	4	Fld_Time + Btw_Time
RspLat_Time	Text	30	Time (sec) between entering field and first keystroke
RspLat_SS	Text	30	Time (sec) between entering field and first keystroke
RspLat_Time_Mss	Long Integer	4	Time (milisec) between entering field and first keystroke
Key_Count	Long Integer	4	Number of keystrokes while in Blaise field
Enter_Value	Memo	-	The value of the Blaise field upon entry
Leave_Value	Memo	-	The value of the Blaise field upon exit
Leave_Cause	Text	50	Action that initiated interviewer to leave Blaise field
Leave_Status	Text	50	Field leave value is normal or DK/RF
Prev_Lang	Long Integer	4	Language was switched to previous while in field (Yes/No)
Next_Lang	Long Integer	4	Language was switched to next while in field (Yes/No)
Set_Lang	Long Integer	4	Language was set while in field (Yes/No)
CtrlL_SetLang	Long Integer	4	Language was changed with hot key while in field (Yes/No)
ALTXExit	Long Integer	4	Alt-X interview suspension was initiated at this field
RemClk	Long Integer	4	Interviewer remark was initiated at this field (Yes/No)
RemChng	Long Integer	4	Interviewer remark was changed at this field (Yes/No)
QHelp	Long Integer	4	QXQ Help was initiated at this field (Yes/No)
BlaiseHelp	Long Integer	4	Blaise Help was initiated at this field (Yes/No)
Error_Esc	Long Integer	4	Blaise check was closed at this field (Yes/No)
Error_Esc_Text	Memo	-	Text of the Blaise check encountered before closing
Error_Supp	Long Integer	4	Blaise check was suppressed (Escape) at this field (Yes/No)
Error_Supp_Text	Memo	-	Text of the Blaise check encountered before suppressing
Error_Jmp	Long Integer	4	Blaise field that interviewer jumps to after Blaise check
Error_Jmp_Text	Memo	-	Text of the Blaise check encountered before jumping
Media_Start	Long Integer	4	Blaise launched media file while in this field (Yes/No)
Mouse_Click	Long Integer	4	Any mouse click detected while in this field (Yes/No)

Field Name	Data Type	Size	Description
F1	Long Integer	4	F1 hot key was pressed while in this field (Yes/No)
F2	Long Integer	4	F2 hot key was pressed while in this field (Yes/No)
F3	Long Integer	4	F3 hot key was pressed while in this field (Yes/No)
F4	Long Integer	4	F4 hot key was pressed while in this field (Yes/No)
F5	Long Integer	4	F5 hot key was pressed while in this field (Yes/No)
F6	Long Integer	4	F6 hot key was pressed while in this field (Yes/No)
F7	Long Integer	4	F7 hot key was pressed while in this field (Yes/No)
F8	Long Integer	4	F8 hot key was pressed while in this field (Yes/No)
F9	Long Integer	4	F9 hot key was pressed while in this field (Yes/No)
F10	Long Integer	4	F10 hot key was pressed while in this field (Yes/No)
F11	Long Integer	4	F11 hot key was pressed while in this field (Yes/No)
F12	Long Integer	4	F12 hot key was pressed while in this field (Yes/No)
CtrlD	Long Integer	4	Ctrl-D hot key was pressed while in this field (Yes/No)
CtrlR	Long Integer	4	Ctrl-R hot key was pressed while in this field (Yes/No)

- tSuspendVariables (calculated using the adt files).

Name	Type	Size	Description
ProjectId	Text	30	Unique project identifier
CaseID	Text	30	Sample ID within a project
vTotalSuspend	Long	4	Total number of suspends for this sample line
vCaseComplete	Yes/No	1	Is the case completed?
vSuspendedVariable1	Memo	-	Blaise field at which first suspension occurred
vSuspendedVariable2	Memo	-	Blaise field at which second suspension occurred
vSuspendedVariable3	Memo	-	Blaise field at which third suspension occurred
vSuspendedVariable4	Memo	-	Blaise field at which fourth suspension occurred
vSuspendedVariable5	Memo	-	Blaise field at which fifth suspension occurred
vSuspendedVariable6	Memo	-	Blaise field at which sixth suspension occurred
vSuspendedVariable7	Memo	-	Blaise field at which seventh suspension occurred
vSuspendedVariable8	Memo	-	Blaise field at which eighth suspension occurred
vSuspendedVariable9	Memo	-	Blaise field at which ninth suspension occurred
vSuspendedVariable10	Memo	-	Blaise field at which tenth suspension occurred

Name	Type	Size	Description
vDate1	Text	30	Date on which first suspension occurred
vDate2	Text	30	Date on which second suspension occurred
vDate3	Text	30	Date on which third suspension occurred
vDate4	Text	30	Date on which fourth suspension occurred
vDate5	Text	30	Date on which fifth suspension occurred
vDate6	Text	30	Date on which sixth suspension occurred
vDate7	Text	30	Date on which seventh suspension occurred
vDate8	Text	30	Date on which eighth suspension occurred
vDate9	Text	30	Date on which ninth suspension occurred
vDate10	Text	30	Date on which tenth suspension occurred
vTime1	Text	30	Time at which first suspension occurred
vTime2	Text	30	Time at which second suspension occurred
vTime3	Text	30	Time at which third suspension occurred
vTime4	Text	30	Time at which fourth suspension occurred
vTime5	Text	30	Time at which fifth suspension occurred
vTime6	Text	30	Time at which sixth suspension occurred
vTime7	Text	30	Time at which seventh suspension occurred
vTime8	Text	30	Time at which eighth suspension occurred
vTime9	Text	30	Time at which ninth suspension occurred
vTime10	Text	30	Time at which tenth suspension occurred
vLastSuspendedVariable	Memo	-	Last Blaise variable at which a suspension
vLastSuspendedDate	Text	30	Date on which the last suspension occurred
vLastSuspendedTime	Text	30	Time at which the last suspension occurred

- **tCariField** (this from the DRI/CARI log)

Field	Type
ProjectID	varchar(30)
CaseID	varchar(50)
FormNum	int
FieldSeq	int
FileNum	smallint
FieldType	smallint
FieldName	varchar(400)
StartLineNum	int
EndLineNum	int
Mss	bigint
Ss	bigint
AccuMss	bigint
AccuHHMMSSMSS	varchar(50)
StartTimeStr	varchar(50)
EndTimeStr	varchar(50)

- **tCaseRecord** (this from the DRI/CARI log)

Field	Type
ProjectID	varchar(30)
CaseID	varchar(30)
RecordID	int
vLastChangeVarName	varchar(400)
vLastChangeDate	varchar(30)
vLastChangeTime	varchar(30)
vLastAccessVarname	varchar(400)
vUserID	varchar(30)
vDataModelName	varchar(30)
Adj_Time_Mss	int

Specific elements for BlaiseIS (organization two)

Interview Level Paradata: (via the Journal or journal like tool)

Record for each page presented to the respondent and also after the submission or movement from away from the current page thus recording a history of all respondent transactions.

At entry to page, record the following:

- Primary Key
- Referring Page (where they came from)
- Blaise IS page number
- Blaise .bmi version
- Blaise language #
- Time page was initially displayed (mm/dd/yy hh:mm:ss.00)
- Screen resolution (# x #)
- Browser resolution (# x #)
- Java enabled (Y/N)
- Question(s) displayed (Blaise Variable names)

At exit of page, or movement off the page (a hyperlink or button clicked, browser closed)

- Primary Key
- Blaise IS page number
- Blaise .bmi version
- Blaise language # (especially if they can toggle during interview)
- Time page was submitted (mm/dd/yy hh:mm:ss.00)*
- Screen resolution (# x #)
- Browser resolution (# x #)
- Java Enabled (Y/N)
- Question(s) displayed (Blaise Variable names) and answer(s) submitted
- Page action (page submitted, back button, closed browser, etc.)
- Note: if the session timed out/closed browser, record time as submitted (or at the least when the IIS session ended).

If scripting is enabled, researchers might be interested in movement throughout the page, the order items were selected (or unselected), but we haven't come across many requests for that level of detail. For the most part, they've been interested in how many times a page was viewed by the respondent and how the data changes per each submission. Since screen size (browser or machine) could change while the interview is active, it should be recorded on the interview level rather than the session level.

One note on collecting the browser resolution: while we can capture the size of the window, we can't capture if they are utilizing the zoom feature on their browser, or how they browser is configured (for example, which task bars are visible), therefore it is almost impossible to represent exactly what the respondent is seeing with 100% accuracy and this data may or may not be useful.

Session Level Paradata

While some of this data can be collected via IIS logs, it would be ideal to have it linked to the primary key in the Blaise instrument, especially for debugging purposes. This data would be collected from point survey session begins until the survey session ends either by the respondent properly closing the session

or by the respondent reaching a time out limit. Ideally it should be maintained separate from the interview data, as to maintain confidentiality.

- Primary Key
- Start time of session
- End time of session (recorded from session time out or via proper exiting of the questionnaire)
- IP Address (this should be optional to record and not mandatory to collect)
- Location of host (country or state, but this should be optional to record and not mandatory to collect)
- Browser & Version
- Operating System
- Status on exit (Submit, Abort, Timed out)

Specific Elements for Blaise 5 (organization three)

1. System Level: This records system information like the Diagnostic Tool.
 - Blaise Software version
 - Component installed
 - Services installed
 - OS of server
 - License
2. Survey Level. This records survey information like information in CATI Specification, *.log, and datamodel parameters.
 - Survey_ID
 - Survey Name
 - Version of data model
 - Version of Blaise software compiled
 - Modes (CATI, Web, Capture, etc) defined
 - Mode specific specifications
 - Language (English, French, etc) defined
 - Collection start date
 - Collection end date
 - User Define Management Variables:
 - Max 3 Stratum variables (used for filtering cases and to summarize performance indicators)
 - Max 10 Key variables (used for filtering cases in the dataviewer)
 - Key Performance Indicators (could also be by stratum):
 - Completion Rate (% of cases completed=finalized or extracted)
 - Response Rate (% of finalized or extracted “in-scope” “response cases)
3. Sampling Unit (a.k.a Case) Level (e.g. a household or an enterprise). This records general information and status about the Case.
 - Survey_ID
 - Case_ID
 - Link_ID to other cases
 - Collection mode assigned
 - Collection priority or weight
 - Time Zone
 - Geographic Identifier

- Respondents information (could be an array)
 - Name
 - Address and Geographic Identifier
 - Telephone
 - Email
 - Time Zone
 - If multiple Unit of Interests (UOI) (UOI can be members in a household or locations in an enterprise)
 - #Unit of Interests
 - Unit of Interests completion status
 - User Define Status or Outcome variables
 - Sampling Unit Resolution Status (No started, In progress, Pending, Disabled, Finalized, Extracted)
 - Sampling Unit Validity status (unknown, in-scope, out-of-scope)
 - Sampling Unit In Progress Sub-Status (No Contact, Validated Sampling Unit, Contacted with Sampling Unit, Screening Completed, UOI collection started)
 - Sampling Unit Finalized Sub-status (Response [full or partial case], non-response)
 - History of events associated to the case
4. Per Event (also per Interview or Intra-case) level . This records events that make changes to the cases data. This includes information in the BTH for CATI or the Journal for Web. There is a need to add event paradata via Manipula and API.
- Survey_ID
 - Case_ID
 - Mode
 - Language
 - Date/Day/Time of event
 - Duration of event
 - If Interviewer assisted
 - Interviewer information
 - Interviewer observation
 - Mode specific event outcome (complete, partial, refusal, appointment, change mode, etc)
 - Event Flags (see attached document)
 - Respondents Information changes log if applicable
 - Audit Trail for the event (same info as in ADT file)
5. Question Level. This is similar to a field in the ADT file or the CARI.
- Survey_ID
 - Case_ID
 - Question_ID
 - Mode
 - Language
 - Question text asked
 - Answer category asked
 - Answer Entered or key pressed to exit question
 - Time entered question
 - Time exit question
 - Recording (Sound and/or Screen.)

**SESSION 8 PAPER 2 – USING AUDIT TRAIL TO MONITOR INTERVIEWER BEHAVIOR UNDER
CAPI MODE IN CHINA FAMILY PANEL STUDIES**

Adding Business Intelligence to Paradata: The Blaise Audit Trail

Joel Devonshire and Gina-Qian Cheung - Survey Research Center, University of Michigan

1. Introduction

For the 2012 International Blaise Users Conference, we authored a paper that summarized an effort to make Blaise audit trail (ADT) data more readily available and easier to analyze for end users (Devonshire, Liu, & Cheung, 2012). This effort included parsing ADT files in an automated bulk process allowing the data to be loaded into relational database tables. This new centralized database represented a significant refinement in the University of Michigan's Survey Research Center, Survey Research Operations (SRO)'s approach to paradata processing, one that we hoped would facilitate responsive survey design and higher-quality data collection. While this effort was successful in meeting its initial goals, we also discovered a few shortcomings of this approach related to exploratory analysis capabilities for end users. This paper provides a summary of these shortcomings, and presents a strategy that was used to address them – an Online Analytical Processing cube that pre-aggregates raw ADT data and joins them to other sources of paradata.

1.1. Background

Ever since SRO began to use Blaise to support its operations, we have heavily utilized its paradata capabilities, and over the years have made improvements to how this paradata can be accessed and analyzed. For example, in Devonshire, Liu, & Cheung (2012), we describe the evolution of SRO's approach to ADT processing and analysis. Moving to a centralized SQL Server database to store "raw" ADT records – where every row in the table represents an entry into a Blaise field – was a significant step forward because it allowed the possibility of real-time data analysis on a secure and centralized server. That is, it removed the processing burden from the end user and helped to facilitate an on-demand analysis paradigm.⁸

The ultimate vision guiding that effort was one in which a survey director or analyst could log into a web site (or connect to the database directly with a tool such as SAS or Microsoft Access) and either manually write queries against the data or, ideally, point and click through an intuitive user interface to quickly answer specific questions. Such questions might include how long particular cases or groups of cases are taking, whether certain questions in the instrument might be problematic, how changes in the data model may be affecting data entry or interviewer behavior, or any number of issues. Thus, simply loading the ADT data into relational database tables was not sufficient in and of itself. We needed to consider what tools might be used to ultimately work with the data.

After some initial experimentation with a web-based user interface that read directly from the SQL Server database, we quickly began to realize that developing a tool that met all of our requirements would be challenging for several reasons. Among them were the following:

- As mentioned above, the main table in the ADT database stored one record per field entry in the Blaise instrument. This meant that very quickly, with only a few projects in the database, this table had millions of records. As of this writing, it contains roughly 37 million records over 15

⁸ Throughout this paper, the term "end user" is used to generically refer to any potential user of the ADT data. End users could include survey directors, project managers, data managers, programmers, principal investigators, or anyone else granted access to the data.

projects. This translates into slow query performance, even for relatively simple queries with optimized tables (e.g., indexed columns, etc.).

- The point above was further complicated by the fact that most of the interesting questions that one might want to answer with ADT data involve some type of aggregation of the raw data rather than a simple subset. For example, one might want to compare average interview length between two or more groups of cases. Aggregation queries are generally more processing-intensive, and contribute to slower query performance.
- To be especially useful, ADT data need to be joined to other sources of data. For example, common queries might involve pulling supplemental sample management system paradata (e.g., result codes, sample types, interviewer attributes, etc.) as a basis for grouping the aggregated ADT data. A front-end tool that simply read the ADT database could not allow for this ready-made joining of disparate data sources.
- Finally, as large scale ADT data analysis is still a somewhat unexplored area of investigation, it is not uncommon to have analysts or project managers unsure of the questions they want to ask, or are possible to answer, with this kind of data. Consequently, one important requirement for an analysis tool is that it be very flexible and customizable, allowing for open data exploration as well as targeted analysis. Static pre-defined reports, or dynamic query tools that only provide limited options, would be useful up to a point but would not facilitate the kind of exploratory data analysis that we needed to allow.

1.2. Online Analytical Processing

Given these limitations and requirements, we settled on one solution that allows fast and flexible queries against the ADT data with minimal programming knowledge needed by the end user. Additionally, it allows for pre-joining of ADT data to other data sources so that a wider range of data is available for analysts. This tool is known as an Online Analytical Processing (OLAP) database, or “cube.” OLAP cubes represent one aspect of what is commonly referred to as “Business Intelligence,” or “BI.” While there are various definitions of the term BI, the one attributed to Boris Evelson (2008) seems to capture BI in its broadest sense: A “set of theories, methodologies, processes, architectures, and technologies that transform raw data into meaningful and useful information for business purposes.” BI can include elements such as dashboards and other data visualization tools, data mining tools, predictive analytics, and a host of other things, including OLAP cubes.

Online Analytical Processing stands in contrast to Online *Transaction* Processing (OLTP), which is what is traditionally used in relational database environments. The terms themselves highlight the relative strengths of each type of data storage paradigm. While transactional databases are optimized for data transactions and storage, analytical databases are optimized for analysis, particularly the kind of analysis that involves the aggregation of long data sets. While it is beyond the scope of this paper to fully describe all elements of an OLAP cube, the major concepts are presented below.

The word “cube” is employed for OLAP databases in order to point to the multi-dimensional nature of pre-aggregated data. While a cube has three dimensions, an OLAP cube theoretically has unlimited dimensions, which are simply groupings of the data. The data that is being grouped is referred to as “facts” or “measures,” and what an OLAP cube typically boils down to is a series of dimension tables that are related to one or more fact tables. The dimension tables define how the fact data is to be grouped, and during cube processing, the fact data is aggregated in various ways according to the defined dimensions.

This is illustrated in Figure 1, an example cube image taken from the internet (Nicolas, G., n.d.). It represents a relatively common use of OLAP cubes: to summarize sales data. In this example, the fact/measure data are total sales (e.g., number of products or dollar amounts), and the groupings of that data, or dimensions, are Customer, Product, and Time. Each dimension has “members,” which are the individual units of that grouping. The primary strength of the OLAP cube is that all of the sales totals are pre-aggregated across all dimensions, so that it becomes relatively easy to query the data and quickly find a total for a specific product for a specific customer on a specific date.

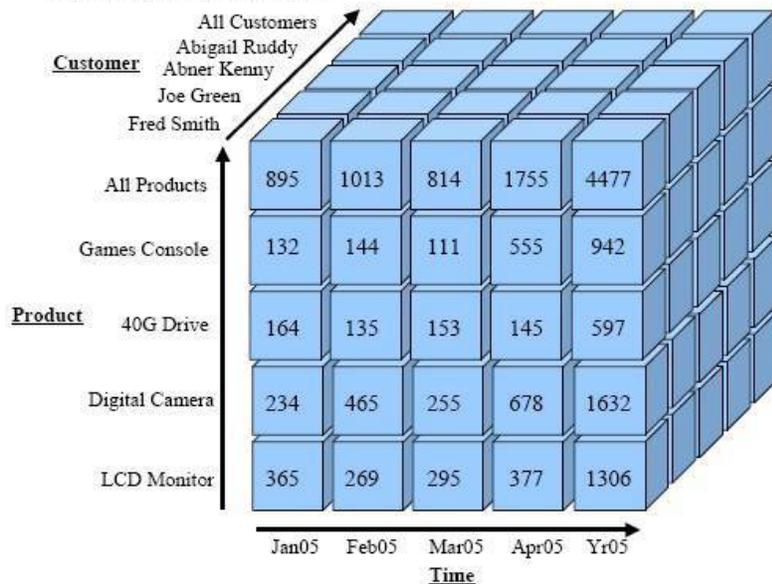


Figure 1. Example of an OLAP Cube (Nicolas, G., n.d.)

The same basic concept can be used to think about ADT data, where in this case the facts, or measures, are any quantifiable elements of the audit trail. Examples include things like elapsed time, keystroke counts, specific hot key counts, counts of “events” such as hard and soft errors, Help menu access, field backups, data-entry, and so forth. Each of these measures could be grouped by things such as Blaise data model fields and blocks, projects, data model versions, sample id, and any one of a number of sample characteristics gleaned from sample management system data that are joined to the ADT data (e.g., sample type, interviewer, result status, date of interview, etc.).

An additional strength of an OLAP cube is its ability to structure the data in such a way that hierarchical relationships are easy to define and navigate. For example, Blaise fields can be easily grouped into their respective blocks, interviewers can be grouped into teams defined by team leaders, days can be grouped into weeks, weeks into months, months into years, and so forth. This allows for a drill-down/roll-up capability when exploring the data.

Finally, OLAP cubes allow a choice between exploring the data through intuitive point-and-click user interfaces (of which more will be said later) or by manually writing queries in a language known as MultiDimensional eXpressions (MDX), which is somewhat similar to Structured Query Language (SQL) but allows one to refer to and query the multidimensional space that makes up a cube. The point to be made here is that OLAP databases can flexibly meet the knowledge base and skills of different user sets. While a set of common tools exist to navigate cube data, a more sophisticated data analyst could also write MDX code to generate a very precise data set to analyze.

2. Building a Data Warehouse

There are several software suites and basic approaches that one can use to create OLAP cubes. Again, it is beyond the scope of this paper to describe all of the considerations that went into deciding what specific approach we would take. It is worth mentioning, however, that we spent time considering two main options for software: SAS Enterprise Business Intelligence Server and Microsoft SQL Server Analysis Services. Both software suites offered very similar capabilities, but for a variety of reasons we decided to use MS SQL Server. This package actually involves three different, but related, services: SQL Server Integration Services (SSIS), SQL Server Analysis Services (SSAS), and SQL Server Reporting Services (SSRS). It is worth mentioning each one because they each serve a different function in the total BI implementation and will be described further in this paper.

One interesting aspect of the SQL Server package is its ability to allow different models to create an OLAP cube. While it is possible to create a cube directly on top of OLTP data sources – an option that carries the significant advantage of being much simpler to design and set up – we elected to go the more traditional route, which involves the design of an intermediate “staging ground” for the data. This staging ground is commonly referred to as a data warehouse, and it is designed in such a way so as to facilitate the cube creation by storing all of the needed data in one place, and having the data structure and relationships optimized for the cube aggregation processing. The process of preparing and loading the data into the data warehouse is known as the “Extract, Transform, and Load” (ETL) process, and will be described in more detail below.

2.1. *The Star Schema*

As mentioned above, an OLAP cube typically uses several dimension tables in the data warehouse that are related to one or more fact tables. Because all of the dimension tables need to relate back to the fact table(s), the resulting data warehouse database schema looks very much like a star when visualized. The “Star Schema” is one of the most common structures used to support OLAP cubes (though other schema designs are certainly possible). This is illustrated in Figure 2, which shows the structure of the ADT data warehouse as of this writing.

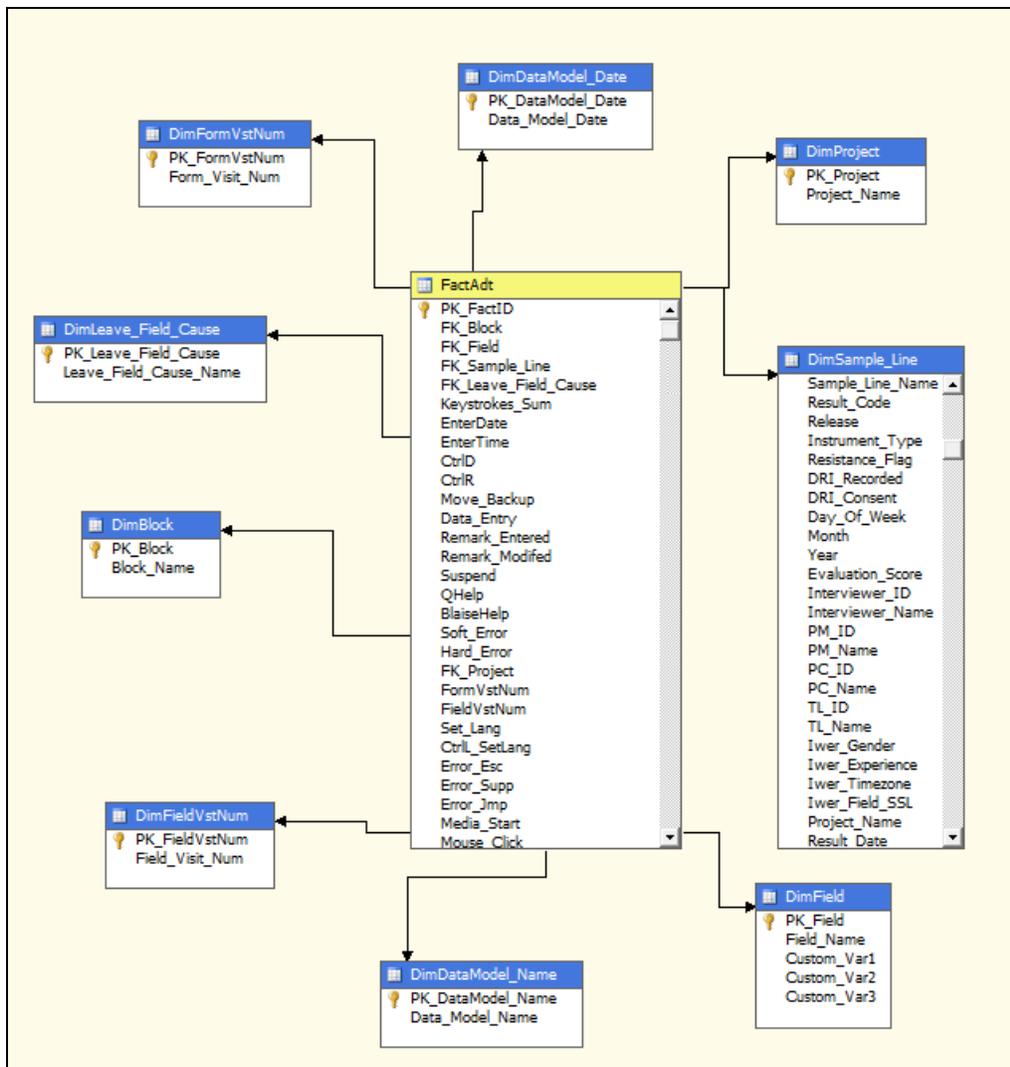


Figure 2. Star schema from ADT data warehouse

The ADT data warehouse currently contains nine dimension tables, and the primary keys in each one relate to foreign keys in the “FactAdt” table. A full list of all of the dimensions (including their attributes) and measures, as well as descriptions of each item, can be found in the Appendix.

2.2. Data Flow and Processing

In Devonshire, Liu, & Cheung (2012), we describe the process by which we parse individual ADT files and store them in an OLTP database. In the following discussion, this initial database will be referred to as the “raw ADT database.” As we briefly mentioned above, the process of creating an OLAP cube on top of that raw data involves a few distinct steps, the most complicated probably being the ETL process. Figure 3 illustrates the broader picture of steps involved. For example, note how the ETL process is where the raw ADT data are combined with any other data sources that need to be available in the cube (e.g., our sample management system known as SurveyTrak, etc.). The ETL stage is complex because not only do different data sources need to be combined, but this is the stage where new data columns need to be calculated, existing columns transformed, and relationships created so that the end result is a set of data that can be loaded into the star schema pictured above.

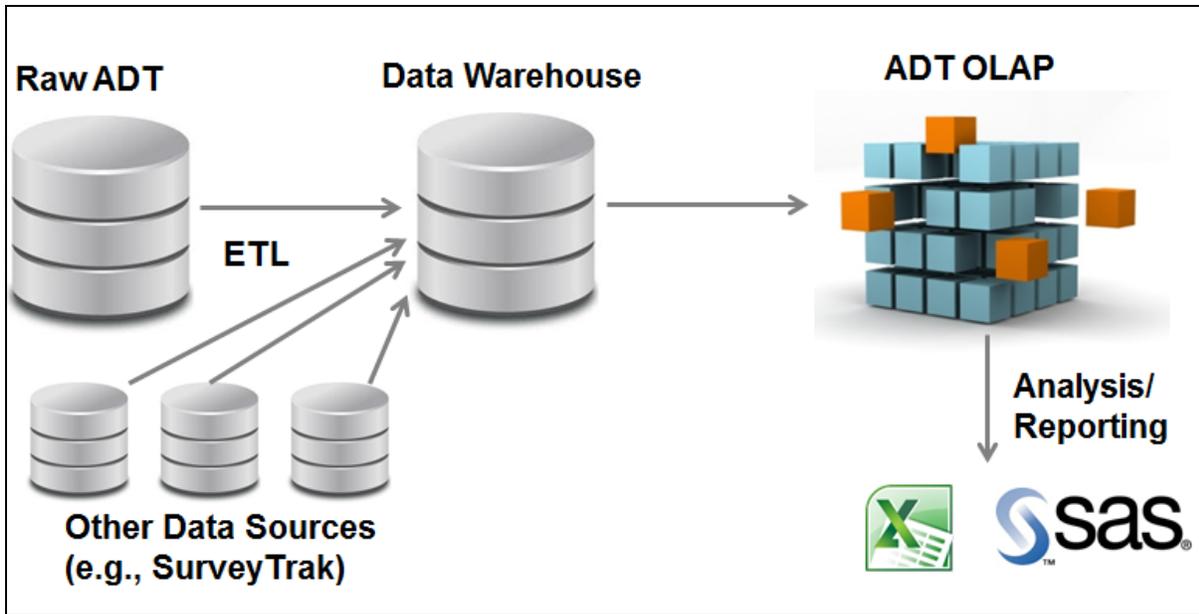


Figure 3. Steps to build an OLAP cube

The ETL stage is also where SQL Server Integration Services is used, which provides a user interface for developing a data flow procedure. An example of this is pictured in Figure 4, and shows each step of the process, including wiping out the data, loading each dimension table, and then loading the fact table. Figure 5 shows the steps involved in loading the fact table, and is only presented here to provide a sense of layered process behind preparing the raw data.

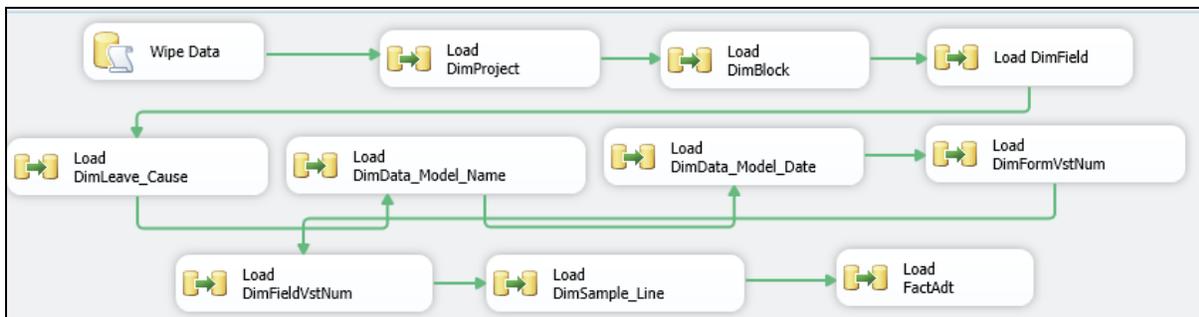


Figure 4. Data flow of ETL process

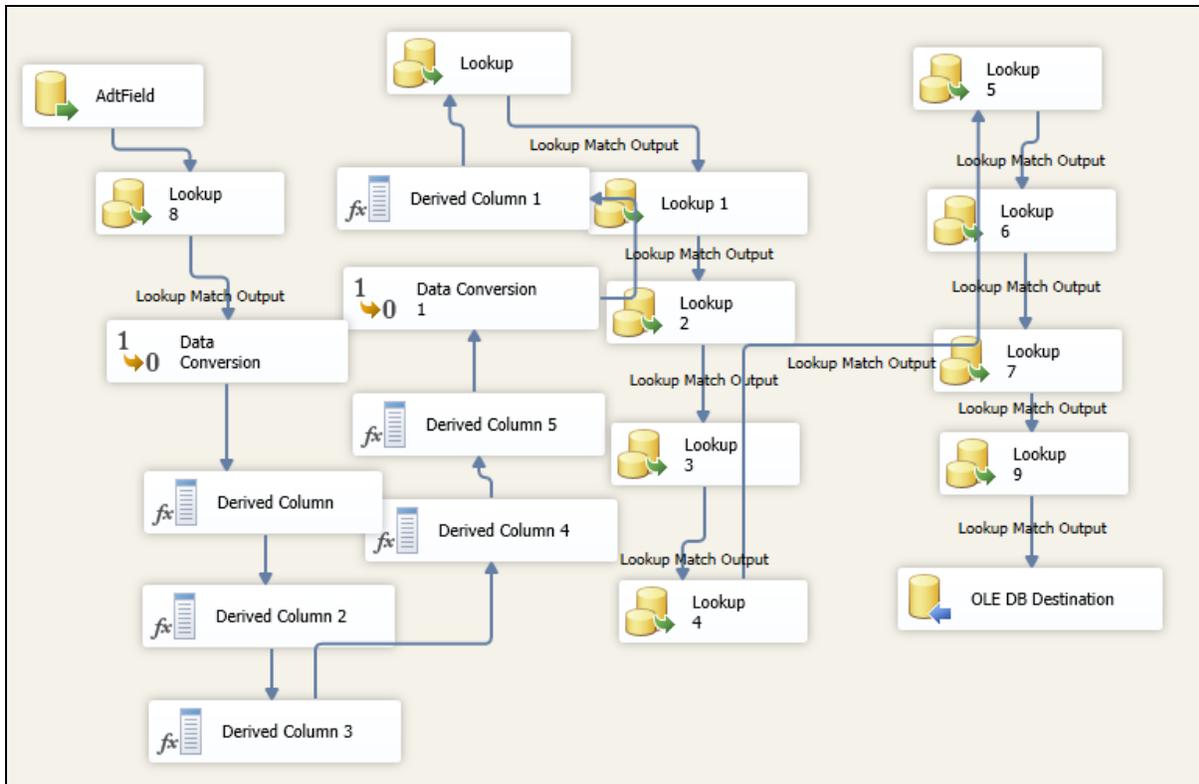


Figure 5. Breakdown of fact table loading process

3. Building the ADT OLAP Cube

Once the data warehouse is loaded with all the necessary data, SQL Server Analysis Services is used to construct the cube. This stage is also pictured in Figure 3. A different user interface is used to create a connection to the data warehouse, define all the dimensions and measures of the cube (which should correspond to the structure of the data warehouse), and define a variety of parameters of the cube, such as whether/how the data should be partitioned, what level of aggregation to define, and the various user roles that need to exist for end users. It is possible, for example, to define custom views of the cube, or allow access to only certain portions of the data to specific people.

3.1. Choosing Dimensions and Measures

Ideally, the dimensions and measures in the cube have been thought through and defined fairly early on in the process, long before the cube is created. This is necessary because, as is evident from Figure 3, the cube structure carries implications for how the ETL process and data warehouse are defined. In choosing the dimensions and measures for the ADT OLAP cube, we attempted to be broad but not overwhelming. For example, it is possible to include in the cube an average, minimum, maximum, standard deviation, and median values (among others) for any particular measure in the cube, such as elapsed time. Thus, a decision needs to be made up-front regarding what kinds of information might be most useful to see at the outset. Tools also exist for the end user to add their own custom calculated measures, which means that not everything needs to be included at the outset.

As can be seen in the appendix, we selected a wide range of measures and attempted to select enough dimensions to allow the same measures to be viewed at different levels of granularity (e.g., data model, block name, field name, field visit number, etc.). We also introduced an interviewer hierarchy to allow a

drill-down through various levels of field management structure. Some of the measures, such as “Data Entry” represent calculated measures defined in the ETL process (e.g., Data Entry = 1 when the value upon leaving the field does not equal the value upon entering the field). Another example of this is the measure “Move Backup,” which is calculated to equal 1 whenever the reason for leaving the field included pressing the back or up arrows, page up, etc. We also added fields to hold custom variables that can be defined for each project in the ETL process. These variables could contain different subsets of Blaise fields in order to create, for example, custom timing variables. Finally, we added a diverse set of variables from our sample management system, SurveyTrak. Some examples include indicators of whether the interview was digitally recorded, whether the case was associated with some respondent resistance, the experience level of the interviewer, the date of data collection, and so forth.

3.2. Automated Processing and Implementing Changes

Once the ETL process and OLAP cube have been created, they can both be deployed to a central server and put on a scheduler to run at an ideal time, which for us is overnight. This means that the data is refreshed and all cube processing happens in the background, before the end user connects and runs queries. The more complex the cube is, the longer processing will take, but the ADT OLAP cube typically takes roughly one and ½ hours, including the ETL process.

Once the cube is deployed in production, usage can be tracked in order to collect data about common queries run against the data. Requested changes to the cube structure can also be made fairly easily, and take effect the next time the cube is processed. Consequently, it is relatively easy to add in additional variables or calculations that analysts find useful.

4. Real-World Usage Examples

In many ways, the power and flexibility of an OLAP cube is difficult to grasp until one actually connects and starts to use it to explore the data. The structure of the data can be best likened to a large pivot table, in which different measures can be selected or combined, and the grouping changed at will by selecting various combinations of dimensions. In fact, the pivot table is exactly the mechanism that many applications use to display and work with cube data. In this section, we will discuss the ways to connect to and use cube data, and some potential real-world applications of the ADT data in particular.

4.1. Tools To Connect To the Cube

Just as there are many tools one could use to build an OLAP cube, there are many ways to view the resulting data. One of these tools is familiar among many people at SRC/SRO, Microsoft Excel, which can natively support connecting to a SSAS OLAP cube once it is deployed to a central server. When using Excel, navigation of cubes takes place with a pivot table that reads all of the dimensions and measures directly from the cube. This is illustrated in Figures 6 and 7.

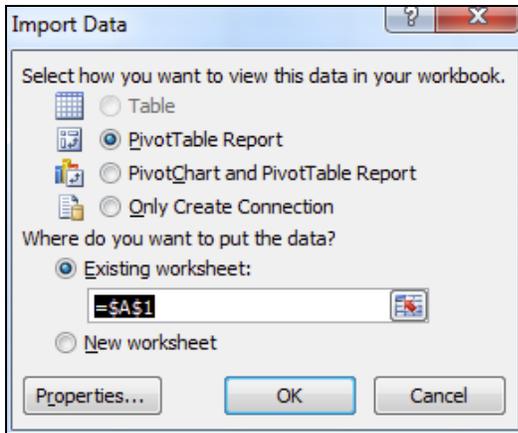


Figure 6. Option to create pivot table or pivot chart in Microsoft Excel

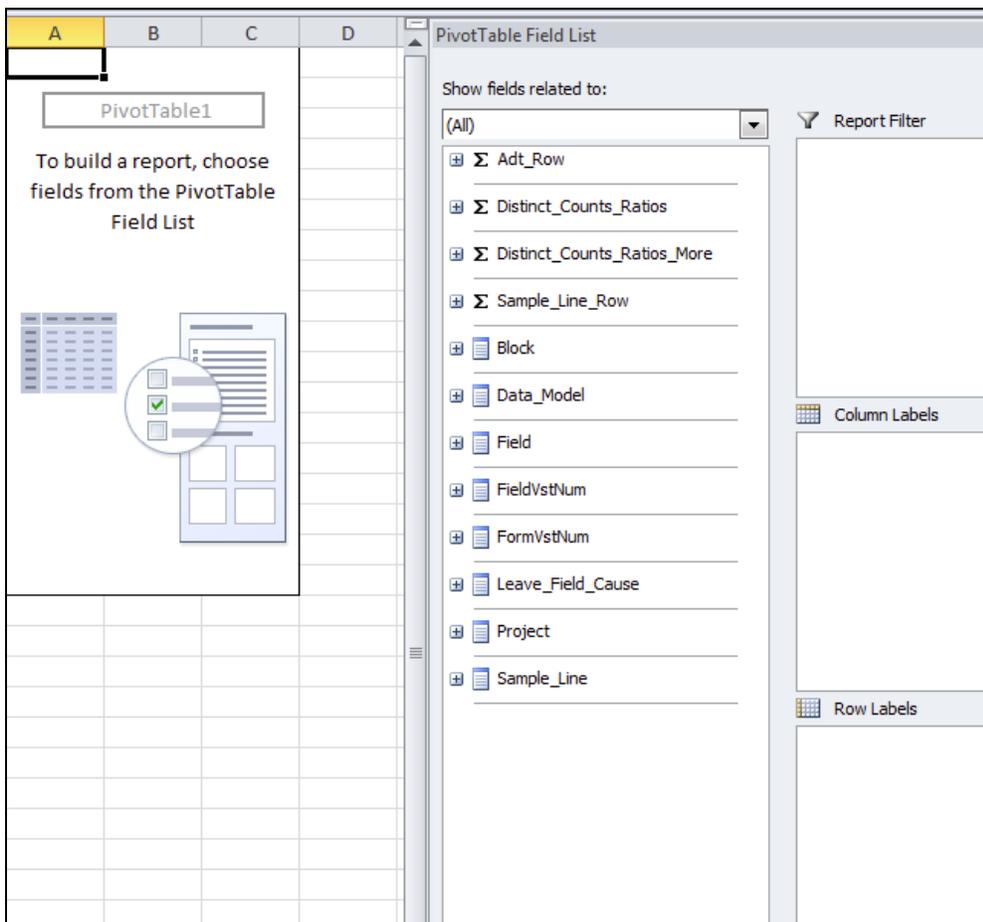


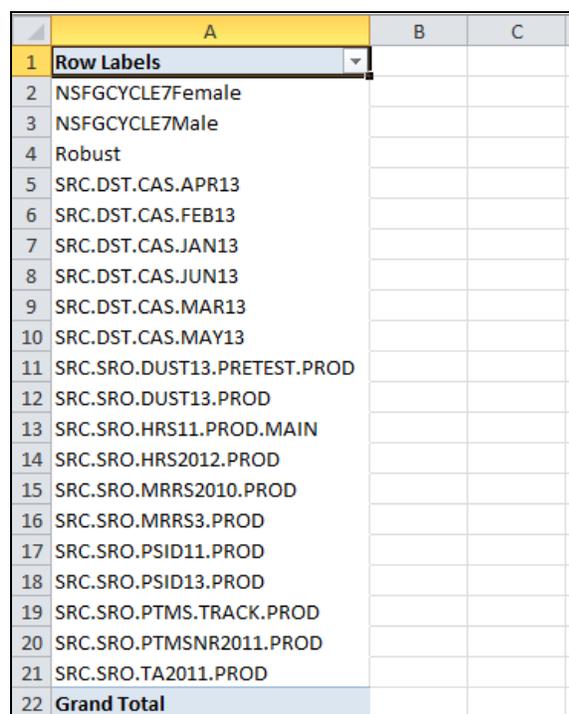
Figure 7. Pivot table in Excel once connected to the cube

4.2. Creating data sets

A full primer on the features and usage of pivot tables is beyond the scope of this paper. Briefly, pivot tables allow a wide range of features, including sorting, filtering, calculating, and even writing data back to the cube (if allowed by the administrator). Pivot charts can be used to add some visualization as well.

In Figure 7, the measures and dimensions are displayed in a collapsed state, which means that the various dimension attributes and specific measures are not visible. However, in practice, all of them would be fully expanded, and the strength of the pivot table design becomes apparent as one simply clicks on checkboxes to add, remove, or combine elements to the table. To give just a sense of the different combinations possible, we will present a few examples of data sets that could be created.

Let us imagine that questionnaire designers would like to assess the various items in the questionnaire. They would like to have some sense of average times spent within each question, the number of hard or soft errors that are encountered at each question, the number of times interviewers back up to the previous question, the number of times data is entered, and the number of times each question tends to be presented in a typical interview. With the OLAP cube, building such a report is a matter of just a few clicks of the mouse. For example, the first step would be to filter the cube to just the project of interest (assuming one did not want to compare across multiple projects). An example of the project list is pictured in Figure 8.



	A	B	C
1	Row Labels		
2	NSFGCYCLE7Female		
3	NSFGCYCLE7Male		
4	Robust		
5	SRC.DST.CAS.APR13		
6	SRC.DST.CAS.FEB13		
7	SRC.DST.CAS.JAN13		
8	SRC.DST.CAS.JUN13		
9	SRC.DST.CAS.MAR13		
10	SRC.DST.CAS.MAY13		
11	SRC.SRO.DUST13.PRETEST.PROD		
12	SRC.SRO.DUST13.PROD		
13	SRC.SRO.HRS11.PROD.MAIN		
14	SRC.SRO.HRS2012.PROD		
15	SRC.SRO.MRRS2010.PROD		
16	SRC.SRO.MRRS3.PROD		
17	SRC.SRO.PSID11.PROD		
18	SRC.SRO.PSID13.PROD		
19	SRC.SRO.PTMS.TRACK.PROD		
20	SRC.SRO.PTMSNR2011.PROD		
21	SRC.SRO.TA2011.PROD		
22	Grand Total		

Figure 8. Project list from OLAP cube

Once the project was selected, then the user could click on the Field Name dimension to see a list of all Blaise fields in that project's data model (Figure 9).

A	
1	Row Labels
2	SRC.SRO.PSID13.PROD
3	AddrPayment.Dust.DUSTINTRO[1]
4	AddrPayment.ProxyAddr.Addr1
5	AddrPayment.ProxyAddr.Addr2
6	AddrPayment.ProxyAddr.AptSte
7	AddrPayment.ProxyAddr.City
8	AddrPayment.ProxyAddr.Country
9	AddrPayment.ProxyAddr.InCO
10	AddrPayment.ProxyAddr.NamF
11	AddrPayment.ProxyAddr.NamL
12	AddrPayment.ProxyAddr.NamM
13	AddrPayment.ProxyAddr.State
14	AddrPayment.ProxyAddr.Suffix
15	AddrPayment.ProxyAddr.Title
16	AddrPayment.ProxyAddr.Zip
17	AddrPayment.RMailAddr.Addr1
18	AddrPayment.RMailAddr.Addr2
19	AddrPayment.RMailAddr.AptSte
20	AddrPayment.RMailAddr.City
21	AddrPayment.RMailAddr.Country
22	AddrPayment.RMailAddr.InCO
23	AddrPayment.RMailAddr.NamF
24	AddrPayment.RMailAddr.NamL
25	AddrPayment.RMailAddr.NamM
26	AddrPayment.RMailAddr.State
27	AddrPayment.RMailAddr.StateAbbrev
28	AddrPayment.RMailAddr.Suffix

Figure 9. Field list from OLAP cube

From there, the user would simply click on the various measures they would like to add to the table, and within literally a few seconds, the data set would be created (Figure 10).

A		B	C	D	E	F	G
1	Row Labels	AdtRow_Count	Error Hard	Error Soft	Timing_FieldAvg	Move Backup	Data Entry
2	SRC.SRO.PSID13.PROD	4435538	8211	1552	0.1605	280221	3333232
3	AddrPayment.Dust.DUSTINTRO[1]	2102	15	0	0.3391	107	1832
4	AddrPayment.ProxyAddr.Addr1	92	0	0	0.1886	10	75
5	AddrPayment.ProxyAddr.Addr2	102	0	0	0.0381	15	28
6	AddrPayment.ProxyAddr.AptSte	97	0	0	0.0248	13	8
7	AddrPayment.ProxyAddr.City	96	0	0	0.0695	15	70
8	AddrPayment.ProxyAddr.Country	1	0	0	0.0658	0	1
9	AddrPayment.ProxyAddr.InCO	81	0	0	0.0235	8	4
10	AddrPayment.ProxyAddr.NamF	86	0	0	0.0598	9	69
11	AddrPayment.ProxyAddr.NamL	88	0	0	0.0533	16	65
12	AddrPayment.ProxyAddr.NamM	94	0	0	0.0239	11	22
13	AddrPayment.ProxyAddr.State	84	0	0	0.0474	9	65
14	AddrPayment.ProxyAddr.Suffix	78	0	0	0.0188	8	2
15	AddrPayment.ProxyAddr.Title	80	0	0	0.1149	3	41
16	AddrPayment.ProxyAddr.Zip	83	0	0	0.1022	8	67
17	AddrPayment.RMailAddr.Addr1	8536	0	0	0.1096	296	1836
18	AddrPayment.RMailAddr.Addr2	8965	0	0	0.0282	848	88

Figure 10. Field-level summary report

At that point, the user could simply uncheck the “Field Name” dimension and select some other dimension, such as the Interviewer Name or Sample Id, to almost instantaneously change the grouping of the same measures. (Note in Figure 11 that the first column has changed from field name to sample Id). In this way, users can point and click various combinations of dimensions and measures to very quickly create data sets that could then be analyzed in other programs.

	A	B	C	D	E	F	G
1	Row Labels	AdtRow_Count	Error Hard	Error Soft	Timing_FieldAvg	Move Backup	Data Entry
2	SRC.SRO.PSID13.PROD	4435538	8211	1552	0.1605	280221	3333232
3	0004151	672	3	0	0.1766	34	547
4	0004192	475	0	0	0.2286	16	392
5	0004312	432	0	0	0.2092	13	342
6	0004712	693	1	2	0.2748	36	581
7	0005001	685	0	0	0.1878	18	574
8	0005002	477	1	0	0.1814	24	374
9	0005003	678	1	0	0.1869	38	532
10	0005311	688	0	0	0.1384	88	479
11	0005511	592	1	0	0.2119	24	492
12	0006001	680	0	0	0.2121	25	550
13	0006002	645	3	0	0.1889	21	528
14	0006003	407	1	0	0.2389	17	283
15	0006141	610	0	0	0.1793	17	506

Figure 11. Sample ID-level report

As mentioned above, visualization can also be added to assist analysis. Figure 12 shows a chart that was created in less than a minute, showing a comparison of average interview length over time between two groups of sample.

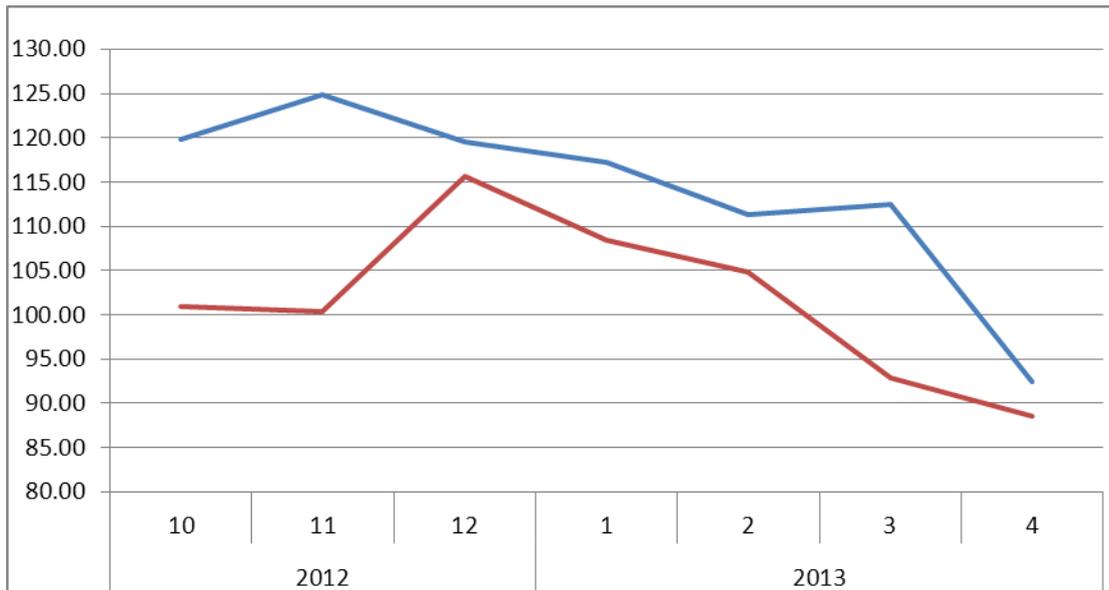


Figure 12. Pivot chart of average interview length over time between two sample groups.

5. Further Development and Conclusion

5.1. Strengths and Weaknesses of the OLAP approach

Using the OLAP cube approach for analyzing and reporting on ADT data offers some very compelling advantages. First and foremost, OLAP cubes are extremely flexible, inviting data exploration and new query ideas through an interface that does not require advanced programming skills. It additionally takes the burden of processing times and complex data joins off of the end user, so that they can concentrate on quickly answering the questions of interest. Cubes are also very customizable, offering the ability to define limited views, hierarchical relationships, and options to pull together data from a variety of sources. They are also accessible with familiar tools, such as Excel. Cube data can very easily be used as the basis of “dashboards” or other data visualization tools with key metrics to help keep the pulse of data collection. Finally, since the data are stored in aggregated form, it is easy to strip out all personally identifying information (PII) so that access to the ADT data can be granted more widely than it otherwise might.

However, OLAP cubes are not without their own set of constraints and limitations. While it seemed very well suited for this particular application, building and maintaining an OLAP cube is also a complicated endeavor, and can easily be overkill for data sets that do not necessarily need to be heavily restructured, combined, or aggregated. While they remove a lot of burden from the analyst, that burden is generally shifted to the IT support staff or database administrators (i.e., whoever designed and maintains the cube). This can lead to a situation where certain aspects of the data structure are necessarily under centralized control, and changes to the cube, while usually relatively easy after the initial design phase, could still be subject to potential bottlenecks.

Additionally, cubes can be deceptively easy to use, especially when working with tools such as Excel pivot tables. It is easy, for example, to build a data set or comparison from the underlying cube. What is more difficult is to carefully think through the goals of the analysis, or what the various calculations mean from a practical perspective. While the tool is flexible, it can also be overwhelming to users, at least until some initial familiarity is developed.

Finally, while it is a significant advantage to be able to join ADT data to other sources and store it in one central location, this can also present challenges related to harmonization among different studies that may have each included a slightly different set of variables, or stored the same variables in different locations or named them differently, and so forth. Attempting to create a one-size-fits-all cube to hold all projects can consequently make the ETL process extremely complicated, as project-specific variables are recoded into standardized variables.

5.2. Potential Enhancements

The ADT OLAP cube is still a relatively young phenomenon at SRO, and it will be interesting to see how it grows and evolves over time, or indeed if other cubes are constructed to summarize other kinds of paradata altogether. What seems most likely in terms of short-term future enhancements are refinements and/or additions to the list of measures and dimensions, as well as the addition of some other sources of data, such as interviewer timekeeping and interview evaluation or verification data. Additional projects will continue to be loaded into the cube, and further thought will need to be given about data harmonization among studies. In the long term, perhaps the ADT cube could grow into just one aspect of a much larger cube that captures most, if not all, of the paradata we collect.

Finally, it is worth mentioning that many off-the-shelf products are available to connect to and work with OLAP cubes once they are deployed. Many of these products are web-based, and once installed on a web server, can provide a similar point-and-click user interface, but perhaps more easily accessible. Depending on the needs for this level of access, we may consider such web-based tools in the future.

6. References

Devonshire, J., Liu, Y, & Cheung, G. (2012). Blaise Audit Trail Data in a Relational Database. *Complete Volume of 14th IBUC 2012*. (38-46) Paper presented at the 14th International Blaise Users Conference, London, UK.. NatCen

Evelson, B. (2008). Topic Overview: Business Intelligence. Retrieved from Forrester Research database.

Nicolas, G. (n.d.). Oracle Olap. GerardNicom Weblog RSS. Retrieved Aug. 2013. From http://gerardnico.com/wiki/database/oracle/oracle_olap

7. Appendix

Table 1. Structure of table ADT OLAP cube (Measures, Dimensions, and Attributes)

TYPE	MEASUREGROUP	NAME	MEASURE_DESC
Measure	Adt_Row	Last Processed Date	Date of last ETL data refresh
Measure	Adt_Row	AdtRow_Count	Count of all rows of ADT data (i.e., field entries).
Measure	Adt_Row	Blaise Help	Count of Blaise Help instances.
Measure	Adt_Row	Ctrl L Set Lang	Count of Ctrl-L (set/change language) instances.
Measure	Adt_Row	Ctrl_D	Count of Ctrl-D ("Don't Know" hot key) instances.
Measure	Adt_Row	Ctrl_R	Count of Ctrl-R ("Refuse" hot key) instances.
Measure	Adt_Row	Data Entry	Count where field LEAVE_VALUE <> field ENTER_VALUE
Measure	Adt_Row	Data Prepopulated	Count where field ENTER_VALUE <> Empty
Measure	Adt_Row	Enter_Date_Max	Max of Enter Date.
Measure	Adt_Row	Enter_Date_Min	Min of Enter Date.
Measure	Adt_Row	Enter_Time_Min	Min of enter Time.
Measure	Adt_Row	Error Esc	Count of "Escape" responses to Blaise error.
Measure	Adt_Row	Error Hard	Count of Hard Blaise error instances.
Measure	Adt_Row	Error Jmp	Count of Jump responses to Blaise error (i.e., jumps to another question).
Measure	Adt_Row	Error Soft	Count of Soft Blaise error instances.
Measure	Adt_Row	Error Supp	Count of "Suppress" responses to Blaise error.
Measure	Adt_Row	Field Recorded	Distinct Count of recorded fields/questions.
Measure	Adt_Row	FieldVstNum_Max	Max Blaise field visit number. (i.e., number of times the field was visited)

Measure	Adt_Row	FormVstNum_Max	Max Blaise form (instrument) visit number. (i.e., number of suspends)
Measure	Adt_Row	Keystrokes_FieldAvg	Average number of keystrokes for the Blaise field.
Measure	Adt_Row	Keystrokes_FieldStDev	Standard deviation of keystrokes for the Blaise field.
Measure	Adt_Row	Keystrokes_Sum	Count of keystrokes.
Measure	Adt_Row	Media Start	Count of times media was started from within Blaise.
Measure	Adt_Row	Mouse Click	Count of times the mouse was clicked from within Blaise.
Measure	Adt_Row	Move Backup	Count of times iwer backed up to previous field.
Measure	Adt_Row	Q Help	Count of QXQ Help instances.
Measure	Adt_Row	Remark Entered	Count of times a new remark was entered.
Measure	Adt_Row	Remark Modified	Count of times an existing remark was modified.
Measure	Adt_Row	Set Lang	Count of times the language was set on this row.
Measure	Adt_Row	Suspend	Count of times a suspend was initiated on this row.
Measure	Adt_Row	Timing_Btw_Field_Avg	Average time (sec.) spent between leaving one field and entering the next.
Measure	Adt_Row	Timing_FieldAvg	Average time (min.) spent in the Blaise field.
Measure	Adt_Row	Timing_FieldMax	Max time (min.) spent in the Blaise field.
Measure	Adt_Row	Timing_FieldMedian	Median time (min.) spent in the Blaise field.
Measure	Adt_Row	Timing_FieldMin	Min time (min.) spent in the Blaise field.
Measure	Adt_Row	Timing_FieldStDev	Standard deviation of time (min.) spent in the Blaise field.
Measure	Adt_Row	Timing_Sum	Sum of time (min.) spent in the Blaise field.
Measure	Distinct_Counts_Ratios	Distinct_Field_Count	Distinct Count of Blaise fields.
Measure	Distinct_Counts_Ratios	Distinct_Field_Count_lwAvg	Average number of distinct fields within an interview.
Measure	Distinct_Counts_Ratios	TotalRows_DistinctField_Ratio	AdtRow_Count/ Distinct_Field_Count
Measure	Distinct_Counts_Ratios_More	Distinct_DataEntry_Count	Count of distinct fields for which data was entered.
Measure	Sample_Line_Row	Enter_Date_Range	Number of days between first and last ENTER_DATE
Measure	Sample_Line_Row	Keystrokes_lwAvg	Average number of keystrokes for the Interview
Measure	Sample_Line_Row	Keystrokes_lwStDev	Standard deviation of keystrokes for the Interview
Measure	Sample_Line_Row	PSID_Postlw_Tasks_Time_lwAvg	PSID-specific timing variable.
Measure	Sample_Line_Row	PSID_R_Burden_Time_lwAvg	PSID-specific timing variable.
Measure	Sample_Line_Row	PSID_Total_lw_Time_Avg	PSID-specific timing variable.
Measure	Sample_Line_Row	Sample_Line_Count	Distinct count of sample lines.

Measure	Sample_Line_Row	Timing_IwAvg	Average time (min.) spent in the Interview
Measure	Sample_Line_Row	Timing_IwMax	Max time (min.) spent in the Interview
Measure	Sample_Line_Row	Timing_IwMedian	Median time (min.) spent in the Interview
Measure	Sample_Line_Row	Timing_IwMin	Min time (min.) spent in the Interview
Measure	Sample_Line_Row	Timing_IwStDev	Standard deviation of time (min.) spent in the Interview
Dimension	N/A	Block	Blaise Block name
Dimension	N/A	Data_Model_Name	Name of the data model
Dimension	N/A	Data_Model_Date	Release date of data model version
Dimension	N/A	Field	Blaise Field name
Dimension	N/A	FieldVstNum	The visit number of the Blaise field.
Dimension	N/A	FormVstNum	The visit number of the Blaise form (instrument).
Dimension	N/A	Leave_Field_Cause	The cause of leaving the current field (e.g., ENTER, move back, suspend, etc.)
Dimension	N/A	Project	Project name
Dimension	N/A	Sample_Line	Sample line ID
Attribute	Dim: Field	Custom_Var1	Custom-defined variable representing any combination of Blaise fields.
Attribute	Dim: Field	Custom_Var2	Custom-defined variable representing any combination of Blaise fields.
Attribute	Dim: Field	Custom_Var3	Custom-defined variable representing any combination of Blaise fields.
Attribute	Dim: Sample_Line	Interviewer_Hierarchy	Drilldown from Project, PM, PC, TL, and interviewer levels
Attribute	Dim: Sample_Line	Day of Week	Integer that corresponds to the day of the week (from Iw result date); 1=Mon
Attribute	Dim: Sample_Line	DRI Consent	Whether consent was provided to DRI recording (1=yes)
Attribute	Dim: Sample_Line	DRI Recorded	Whether the sample line was recorded (1=yes)
Attribute	Dim: Sample_Line	Evaluation Score	From OLIVE database; evaluation score.
Attribute	Dim: Sample_Line	HRS Cohort	Text label of HRS cohorts
Attribute	Dim: Sample_Line	HRS Language	Text label of HRS language
Attribute	Dim: Sample_Line	HRS Pref Mode	Text label of HRS preferred mode
Attribute	Dim: Sample_Line	Instrument Type	SELF, EXIT, P-EXIT
Attribute	Dim: Sample_Line	Interviewer Name	Interviewer Name (without hierarchy)
Attribute	Dim: Sample_Line	Iwer Experience	Iwer experience level (should be standardized across projects)
Attribute	Dim: Sample_Line	Iwer Field SSL	Interviewer designation, field vs. lab (SSL)
Attribute	Dim: Sample_Line	Iwer Gender	Text label of interviewer's gender.
Attribute	Dim: Sample_Line	Iwer Timezone	Timezone of iwer's home location.
Attribute	Dim: Sample_Line	Month	Integer that corresponds to the month of iw (from Iw result date).
Attribute	Dim: Sample_Line	Project Name	Name of project (only for SurveyTrak projects).

Attribute	Dim: Sample_Line	PSID cell complete	Indicator of whether PSID interview was completed by cell phone.
Attribute	Dim: Sample_Line	PSID Sample Type	Text label of PSID sample type.
Attribute	Dim: Sample_Line	Release	From SurveyTrak, tSample_Line.sRelease
Attribute	Dim: Sample_Line	Resistance Flag	From SurveyTrak, tSample_Line.bRCIndFlag
Attribute	Dim: Sample_Line	Result Code	From SurveyTrak, tSample_Line.sResultCodeID
Attribute	Dim: Sample_Line	Result Date	From SurveyTrak, tSample_Line.dResultDate
Attribute	Dim: Sample_Line	Sample Line	From SurveyTrak, tSample_Line.vSample_LineID
Attribute	Dim: Sample_Line	Year	Year of interview (from lw result date)

Our experience in producing the CAWI questionnaire for the survey "Statistical Report on the Careers of Doctorate Holders (CDH)"

Vladimir Slemenšek, Marko Sluga, Statistical Office of the Republic of Slovenia

1. Introduction

Due to the enforced cost reduction and the need to improve the efficiency, the Statistical Office of the Republic of Slovenia decided to start in 2013 a project with CAWI mode for the survey "Statistical Report on the Careers of Doctorate Holders (CDH)". Until now, the CDH survey has been carried out only through CATI and paper questionnaires. It was necessary to specify how the web questionnaire will look like, how it will be accessed, will it have to be adapted for online use issues, combining its data with data from the CATI questionnaire, etc. In short, the project which we have undertaken, will result in the fact that we will ultimately achieve a standard of manufacture and design of future CAWI and Mixed-mode surveys.

2. Objectives of the project

The objectives of the project were as follows:

- To create a web domain for accessing the CDH survey,
- The CDH survey must be accessible only with a username and password and it must be behind a firewall,
- To create the questionnaire that will be suitable to meet web demands,
- To preserve all properties of the existing CATI questionnaire (introducing »Mixed-Mode«),
- To achieve the standard for the design of future online surveys,
- To record all the problems that we will encounter during development and solve them.

3. Our current CATI questionnaire for the CDH survey

Our current CATI questionnaire consists of the following blocks that gather the information on doctorate holders:

- Block bTel with preliminary questions for determining whether the selected person is appropriate for the survey,
- Block BNonResponse for storing information about the reasons, why a person did not wish to participate in the survey,
- TAppointment table for storing information to be seen by the interviewer on the call screen,
- Block A: PRELIMINARY QUESTIONS - determining the appropriateness of the selected person to be surveyed,
- Block B: Training to achieve a doctorate,
- Block C: Primary employment or Work as of 31 December 2012,
- Block D: Additional employment or Work as of 31 December 2012,
- Block E: Satisfaction with primary employment or primary work, and knowledge, properties and behaviour important for primary employment,
- Block F: Previous employment,
- Block G: International Mobility,
- Block J: Planned migration out of Slovenia, and
- Block K: Experiences related to career.

4. Customizing the questionnaire for CAWI

The CATI questionnaire has a standard layout, but for online surveys it is not too attractive and functional (Figure 1). Furthermore, its blocks with questions also contain help texts in green, intended only for CATI interviewers.

C1. Kakšen je bil vaš zaposlitveni status na 31. 12. 2012?

Preberite vse možne odgovore. Anketiranec lahko izbere samo en odgovor.

- 1. Bili ste zaposleni.
- 2. Bili ste samozaposleni.
- 3. Delali po drugi vrsti pogodbe (npr. po pogodbi o delu, avtorski pogodbi itd.).
- 4. Bili ste brezposelni.
- 5. Bili ste upokojeni, študirali ste, dela niste iskali itd.

B7l	<input type="checkbox"/>	C1a	<input type="checkbox"/>
B7m	<input type="checkbox"/>	C1	<input type="checkbox"/>
B7n	<input type="checkbox"/>	C2a	<input type="text"/>
B7o	<input type="checkbox"/>	C2b	<input type="text"/>
B7p	<input type="checkbox"/>	C3	<input type="text"/>
B7q	<input type="checkbox"/>	C4	<input type="text"/>

Figure 1: Screen capture of the CATI questionnaire for the CDH survey

Therefore, we have adapted Mode Library for the CAWI questionnaire (Figure 2).

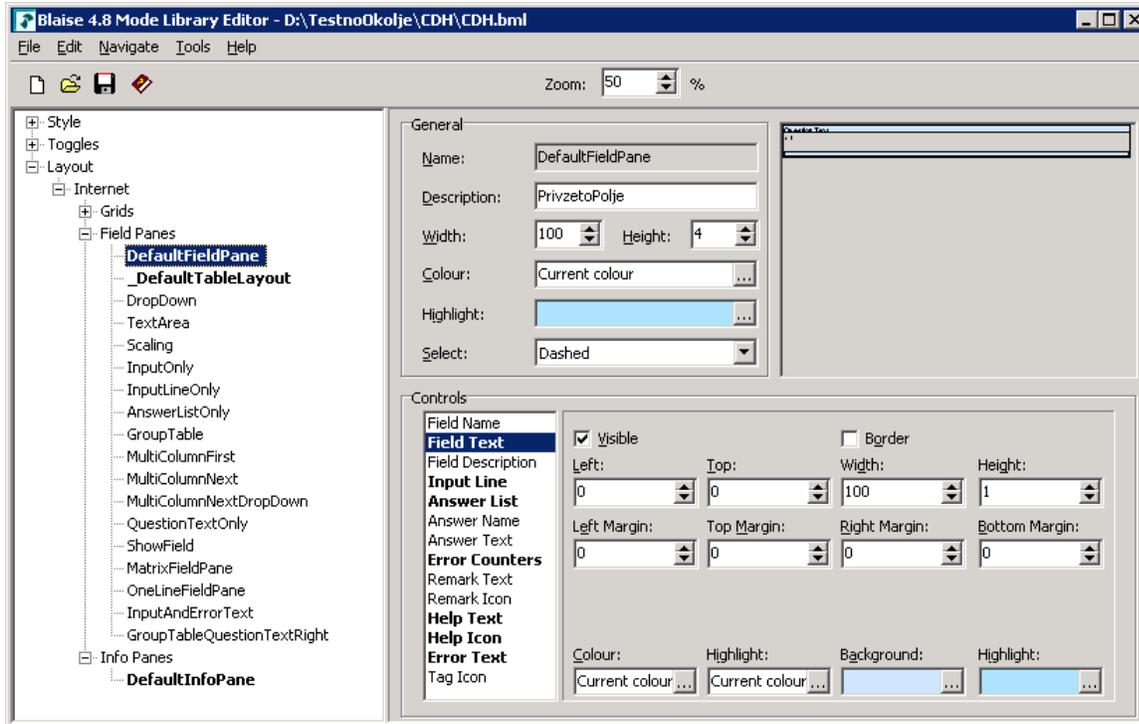


Figure 2: Property settings for DefaultFieldPane

In Blaise Menu Editor for the website, we have made two panels: header for the top and navigation for navigating through the questionnaire at the bottom. The website was designed in a way that the header is always visible on top with the logo of the Statistical Office on the left hand side and the title of the questionnaire on the right hand side, next to the logo. The logo of the Statistical Office is set as a flat button. Clicking on the logo opens the official homepage of the Statistical Office of the Republic of Slovenia (Figure 3). The navigation panel with Forward, Back, Exit and Save buttons was placed at the bottom of the webpage. Background for both panels was set to the colour that was already used in the Mode Library.

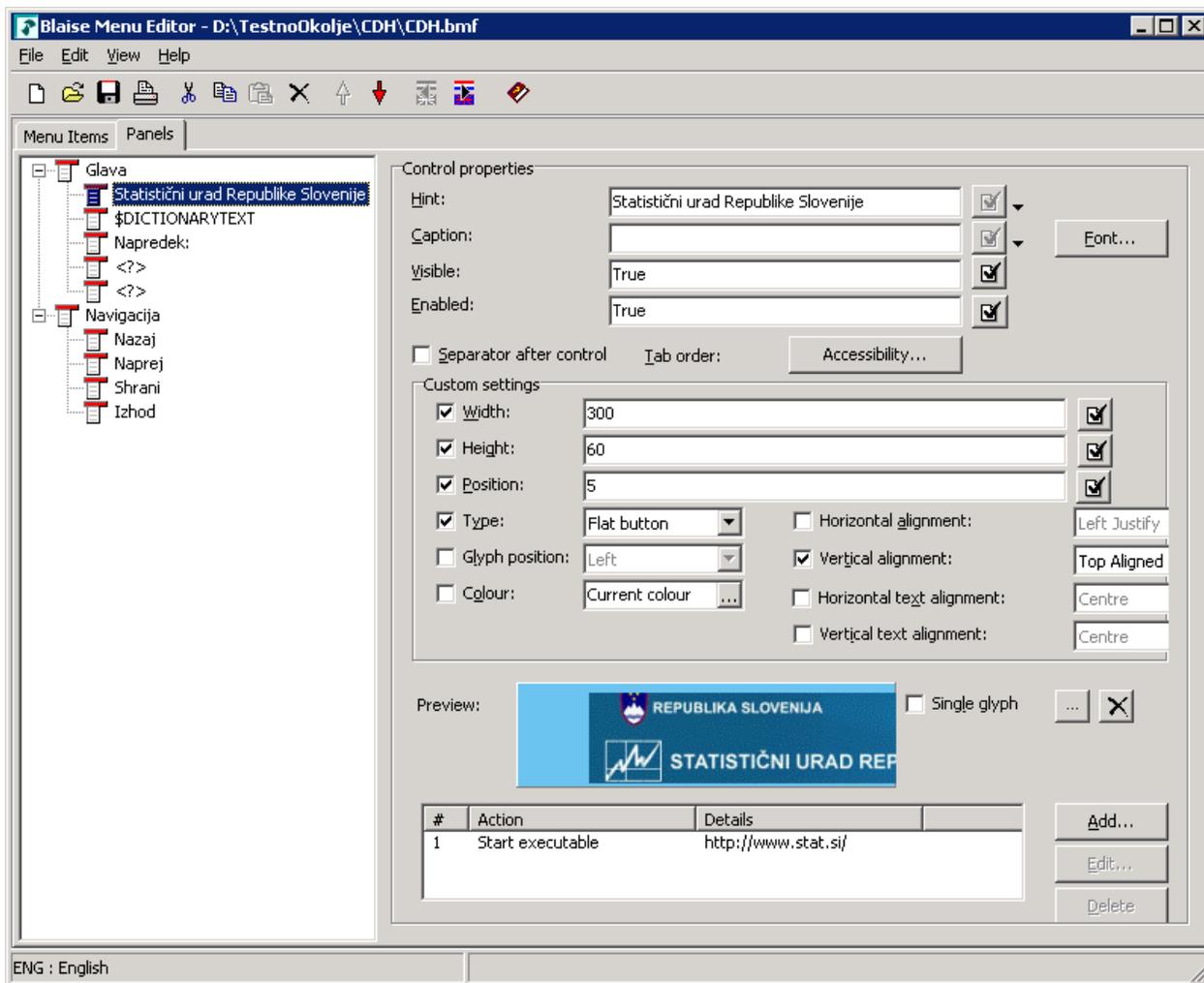


Figure3: Panels for the CAWI questionnaire

Our wish was that the questionnaire would be used in the Mixed-mode. The next goal was that the content of the questionnaire is identical regardless of the mode in which it is used. The problem was, that the display of the contents in CATI or CAWI was completely different:

- Very rarely at CATI was it necessary to use the command NEWPAGE, because the interviewer promptly reads to the respondent a question, fills in the answer and then moves to the next question. In the CAWI questionnaire, questions with fields for entering answers are optimally shown and listed on pages (in blocks with questions the command NEWPAGE is often used). It was preferred that the need for the use of the vertical slider is reduced to the minimum.
- To easier distinguish between the question text and the answer text, the questions in CAWI are shown in bold text without interviewing instructions, which were displayed in green. If such texts were present in the CATI questionnaire, we changed them into the texts for helping the respondent. These texts were presented in italics. We did not want to show those texts as hints.

We had to decide whether we will ask ourselves in each block, at each question, in what mode we currently are, or we will ask ourselves about that only in our Datamodel and then call for blocks with modified questions for CATI or CAWI. We opted for the second option. We have stored a block with questions for CATI in the INC file, which we have named Bloka_CATI.INC, etc. The same block for the CAWI questionnaire was stored in the INC file without an extension _CATI (e.g. BlokA.INC). Questions in these files are modified in the way, that they are suitable for appearance in the CAWI questionnaire. For the switching between the modes, we have used a conditional symbol

named CATI_Mode (Figure 4), defined under Project | Options.... It behaves as being defined with \$DEFINE directive.

```

{-----
V anketi CDH - CATI in CAWI mode je vključen naslednji sklop (skupen obema):
-----}
INCLUDE "bPred.inc" {Blok bPred}
{IFDEF CATI_Mode}
{-----
V anketi CDH - CATI mode so vključeni naslednji sklopi:
-----}
INCLUDE "bTel.inc" {Blok bTel}
INCLUDE "BNonResponse.inc" {Blok BNonResponse}
INCLUDE "TAppointment.inc" {Tabela TAppointment}
INCLUDE "BlokA_CATI.inc" {Sklop A: UVODNA VPRAŠANJA - ugotavljanje ustreznosti izbrane osebe za anketiranje}
INCLUDE "BlokB_CATI.inc" {Sklop B: Izobraževanje za dosego doktorata}
INCLUDE "BlokC_CATI.inc" {Sklop C: Osnovna zaposlitev oz. delo na dan 31.12.2012}
INCLUDE "BlokD_CATI.inc" {Sklop D: Dodatna zaposlitev oz. delo na dan 31.12.2012}
INCLUDE "BlokE_CATI.inc" {Sklop E: Zadovoljstvo z osnovno zaposlitvijo oz. osnovnim delom in znanja, lastnosti}
INCLUDE "BlokF_CATI.inc" {Sklop F: Prejšnja oz. predhodna zaposlitev}
INCLUDE "BlokG_CATI.inc" {Sklop G: Mednarodna mobilnost}
INCLUDE "BlokJ_CATI.inc" {Sklop J: Načrtovana selitev iz Slovenije}
INCLUDE "BlokK_CATI.inc" {Sklop K: Izkušnje, povezane s kariero}
{ELSE}
{-----
V anketi CDH - CAWI mode so vključeni naslednji sklopi, prilagojeni za internet:
-----}
INCLUDE "BlokA.inc" {Sklop A: UVODNA VPRAŠANJA - ugotavljanje ustreznosti izbrane osebe za anketiranje}
INCLUDE "BlokB.inc" {Sklop B: Izobraževanje za dosego doktorata}
INCLUDE "BlokC.inc" {Sklop C: Osnovna zaposlitev oz. delo na dan 31.12.2012}
INCLUDE "BlokD.inc" {Sklop D: Dodatna zaposlitev oz. delo na dan 31.12.2012}
INCLUDE "BlokE.inc" {Sklop E: Zadovoljstvo z osnovno zaposlitvijo oz. osnovnim delom in znanja, lastnosti ter}
INCLUDE "BlokF.inc" {Sklop F: Prejšnja oz. predhodna zaposlitev}
INCLUDE "BlokG.inc" {Sklop G: Mednarodna mobilnost}
INCLUDE "BlokJ.inc" {Sklop J: Načrtovana selitev iz Slovenije}
INCLUDE "BlokK.inc" {Sklop K: Izkušnje, povezane s kariero}
{-----}
{ENDIF}

```

Figure 4: Part of code in Datamodel for switching between CATI and CAWI mode

This way of programming has enabled us to:

- Maintain all the code for CATI intact,
- Arbitrarily change the code and Layout for the CAWI questionnaire,
- Have each block with questions located in its own INC. file,
- Edit a block of questions faster and more transparently,
- Make the code in Datamodel shorter and much more transparent,
- Achieve a Mixed-mode Datamodel,
- Design a nice interface for the online survey (Figure 5).

REPUBLIKA SLOVENIJA
STATISTIČNI URAD REPUBLIKE SLOVENIJE

CDH

Sklop C: Osnovna zaposlitev oz. delo na dan 31.12.2012

C1a. Naslednja vprašanja se nanašajo na osnovno zaposlitev, ki ste jo imeli na dan 31. 12. 2012.
Ali ste bili na ta datum zaposleni pri/na ?

Da
 Ne

C1. Kakšen je bil vaš zaposlitveni status na 31. 12. 2012?
Preberite vse možne odgovore. Izberete lahko samo en odgovor.

Bili ste zaposleni.
 Bili ste samozaposleni.
 Delali po drugi vrsti pogodbe (npr. po pogodbi o delu, avtorski pogodbi itd.).
 Bili ste brezposelni.
 Bili ste upokojeni, študirali ste, dela niste iskali itd.

Izhod Nazaj Naprej Shrani

Figure 5: Example of CAWI questionnaire

5. Security and authentication

We wanted to restrict access to our web survey. So we have decided to use the approach from an example that is shipped with the Blaise system. It is usually located in folder "\\Blaise 4.8 Enterprise\Samples\Internet\Interview>Login". With this approach the authentication questions are separated from the questions of our web survey. In our case the respondent is asked for a username and password. Those two are compared with stored usernames and passwords in a Blaise database. If the username password combination is correct, the respondent will automatically be redirected to the desired questionnaire instead of to a receipt page. We will use pre-generated usernames and passwords, stored in separated Blaise database on the Web Server. For the page IntroLogin.asp we have adapted the content and changed the background colour to a shade of blue. We have added a legal notice about providing information as well as contact information for general and technical assistance (Figure 6).

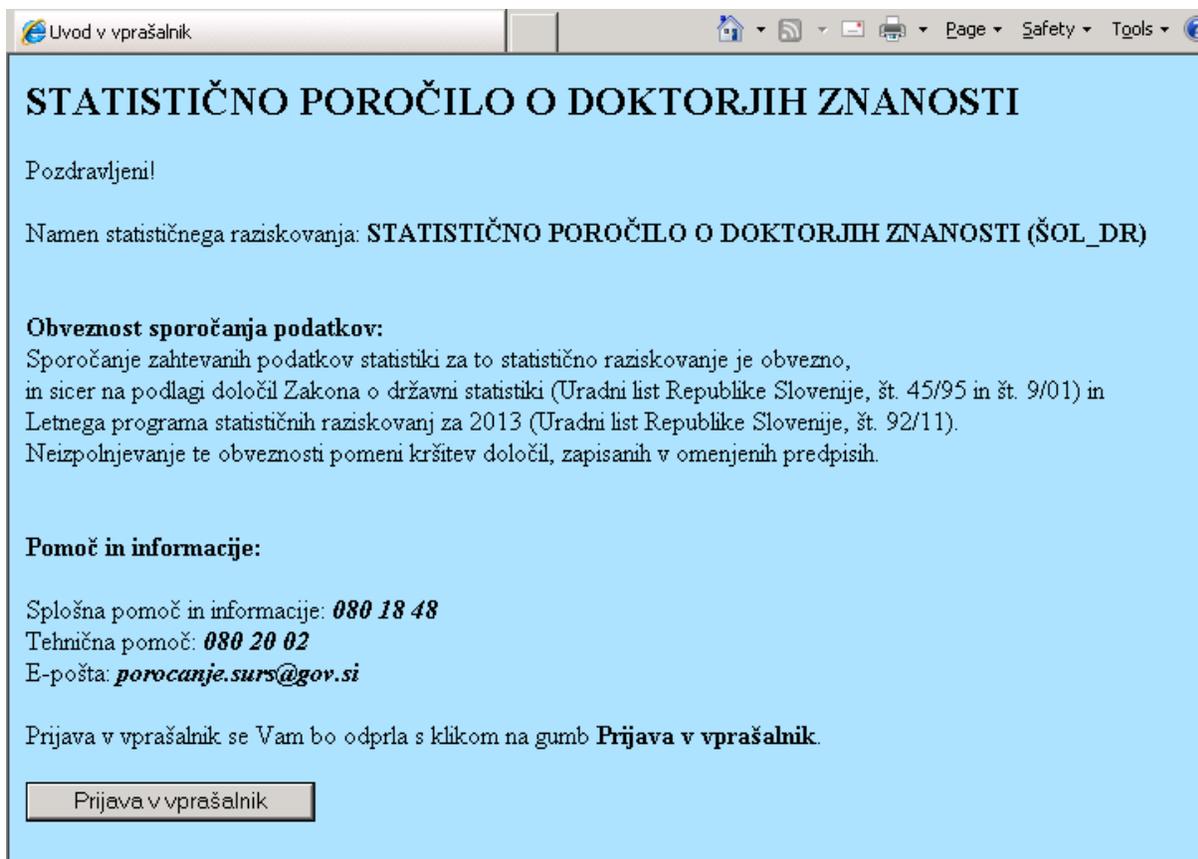


Figure 6: Modified page IntroLogin.asp

For the page GotoSecureInterview.asp we had to add the Slovenian language, translate texts into that language and assign SecureInterviewStartPage variable to CDH.asp. Adding the Slovenian language with translated text was also done for the file InvalidPassword.asp. For making the web survey available to the public, we agreed to publish it on a special subdomain called **cawi.stat.si**. The name of this subdomain was created with an eye to the future web surveys, which will be available there.

6. Translation of key contents into the Slovenian language

To have Slovenian text in other web surveys in the future, we decided to translate key contents in stylesheet files biHTMLWebPage.xsl, biHTMLWebPage.xsl, biHTMLDialog.xsl, in all system pages, error pages, in ReceiptPage.asp, AbortPage.asp and in BiInterviewStarter. First, it was necessary to add the Slovenian language as a new language and then translate English terms. This was quite time consuming. All translated files were saved in a separate folder named PredlogeSURS. This folder will then be copied and used in any future project. In files ReceiptPage.asp and AbortPage.asp we basically did not make many changes. It was necessary to translate the text into the Slovenian language and properly adapt those two files (with UTF-8) for displaying the contents of the questions with special characters that are common in the Central European region.

7. Our focus is the future

At carrying out this survey we intend for the first time to collect information from the online questionnaire (in addition to telephone and postal questionnaire). Unfortunately, a pilot survey will not be implemented this year. Only cognitive testing of CAWI and CATI questionnaires (the latter due to the inclusion of some additional questions in the questionnaire) will be conducted. Because the ongoing survey has a sequential course (CAWI, CATI, PAPI), it is necessary to accurately predict the distribution of "supporting" materials. We have determined

and adapted a set of activities and deadlines (time schedule for the implementation of CDH 2013), and some of the arrangements related to designing and testing the questionnaire and implementing the survey.

Cognitive testing will be carried out first:

- For the online questionnaire (CAWI) and the telephone questionnaire (CATI);
- On 10 male and female doctors (5 on online questionnaire and 5 on telephone questionnaire), that are employed at SURS;
- In the period between 26 June 2013 and 12 July 2013;
- Prior to the testing, a questionnaire for recording the findings will be prepared;
- If after the test no significant and substantive changes to the questionnaire are made, we are planning to include responses of participants in the final data.

Data collection will take place with the combined method in three successive steps:

1. Online survey (conducted from 26 August 2013 to 11 September 2013). Access to the questionnaire will be open until 16 September 2013.
2. Telephone survey (conducted from 18 September 2013 to 30 September 2013 or 2 October 2013 - the deadline is provided for analysis),
3. Postal survey (conducted from 11 October 2013 to 24 October 2013). During this period we will re-open the access to the online questionnaire.

Steps and timeline for conducting the data collection will be adjusted if necessary. This will depend on the number of responses for each mode of data collection.

Dispatching accompanying materials:

- Invitation letters (or e-mails to those for whom we are able to obtain e-mail address) will be sent to respondents together with a password and a notice about the deadline for completing the questionnaire on the same day, 26 August 2013. In the invitation letter that will be sent by post there will be no mention that a notification will also be sent by e-mail, while the e-invitation will state that the invitation letter was sent by post.
- On 04 September 2013 a reminder will be sent about the deadline for completing the questionnaire (for those for whom we were able to obtain e-mail address, they will be sent only by e-mail;- to the rest they will be sent by post). In the reminder it will be mentioned that after this date, if they are not able to complete the online survey, they will be contacted by telephone.
- At the end of the telephone survey we will send to all those who did not respond to the telephone questionnaire or online questionnaire a mail questionnaire (together with invitation letter with a password and a link to the online questionnaire and an envelope with the stamp for sending the questionnaire back to SURS). In the notification letter it will be mentioned that a link to the online questionnaire is reopened and the questionnaire can also be completed online.
- After the last deadline for sending the mail questionnaire back to SURS, we will consider if it will be necessary to send another reminder.

The CDH 2013 survey is a preparatory phase of optimization and modernization of processes for interviewing people and households. The implementation of this survey with online questionnaire represents only a fraction in the mosaic that is necessary for introducing CAWI to SURS. You do need to look at it as one in a number of experiments (tests of individual activities) that have yet to be put into full implementation before implementing CAWI to SURS. We will be invest considerable effort to adapt questionnaires for the online survey.

It has been decided that after the CDH 2013 survey we will produce policies and guidelines for the initiative to carry out the pilot for introducing CAWI to SURS.

8. Conclusion

Many people think that we set ourselves quite a big goal:- i.e. to design a CAWI questionnaire, preferably in a Mixed-mode, with the possibility to switch between CATI and CAWI modes. It was necessary to adjust the entire CATI questionnaire to the new requirements for appearance, functionality and security. It was necessary to set a subdomain to the official domain of our Statistical Office and to secure access to the web survey. It was also necessary to adjust Blaise key contents to the Slovenian language and its peculiarities. The result is that we designed a functional and eye appealing web survey, to which the respondent must sign in with a username and password. During this phase of the project there is still some uncertainty about how to implement some issues. For example, at the moment it is not yet agreed, under what rules completed records will be transferred from the server to the location for collecting data from other sources (which is not determined yet). Also CATI and CAWI do not have exactly the same database structure that would allow us to facilitate easy switching between CATI and CAWI and eliminate the step of adapting the content of one or another in-to a common database. For future surveys it also raises the dilemma of how respondents will be informed about the survey (by mail, by email or by both) and how usernames and passwords will be sent to them.

But we managed to bring the appearance of our CAWI questionnaire (header, navigation, background, appearance of question texts, appearance of help texts and system notifications) one step closer to standardization and the key contents that will be used in future questionnaires. However, there is no progress if there is no remaining challenge also for the future.