Proceedings of the 13th International Blaise Users Conference

IBUC 2010

Baltimore, Maryland USA
October 19 – 21, 2010

Preface

This document contains the papers presented at the 13th International Blaise Users Conference held in Baltimore, Maryland, USA from October 19 – 21, 2010. The conference included three days of technical papers and presentations on the use of Blaise and related topics.

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

- Hilde Degerdal (Statistics Norway, Chair)
- Tom Anderson (Office for National Statistics, UK)
- Karen Bagwell (US Bureau of the Census)
- Gina-Qian Cheung (University of Michigan)
- Leonard Hart (Mathematica Policy Research, Inc., USA)
- Lon Hofman (Statistics Netherlands)
- Vesa Kuusela (Statistics Finland)
- Rebecca Lui (Statistics Canada)
- Jim O'Reilly (Westat, Inc.)
- Mark Pierzchala (USA)
- Fred Wensing (Australia)

Westat, as the host organization, has collected, edited, and printed these proceedings for the benefit of the conference participants and others.

IBUC 2010 was organized and hosted by Westat. The organizing committee was chaired by Jane Shepherd and Jim O'Reilly.

Table of Contents

Computer Assisted Recorded Interviewing (CARI)

Editing/Processing

Business Process Re-engineering using Blaise 4.8 API and Datalink

Mike Hart and David Kinnear, UK Office for National Statistics (ONS)

1. Introduction

In accordance with our remit to develop the use of Blaise within ONS, the Blaise Development Standards and Support team (BDSS) within ONS Social Survey (ONSSS) have been looking at ways of using the software to streamline our business processes. ONS Social Survey systems have largely been based on using Manipula in conjunction with survey-specific Blaise database storage. Whilst this technology is flexible and well understood, it does have the downside of requiring a continual commitment of trained resource in the production and maintenance of Manipula scripts, and it can be difficult to manage data across multiple surveys.

The Blaise API and Datalink functionality provide both an alternative to the reliance on Manipula and an opportunity to reduce the resource committed to manual interventions in processes such as report generation, manipulating Blaise databases, and sample file maintenance on longitudinal surveys. At the same time, this technology can allow survey teams greater control over their business applications.

This paper focuses on the example of Electronic Learning Questionnaires (ELQs) to show how BDSS are helping to re-engineer a specific business process and showcasing these aspects of Blaise within ONS.

2. Selection of business process for re-engineering

When assessing whether to use Blaise API or Datalink to re-engineer a business process, there were a number of factors to consider:

i) What would the impact be on other IT systems within ONSSS? Ideally we wanted to introduce an application that would not have a large impact on other processes;
ii) Was there a clear efficiency gain to be had from re-engineering?
iii) Technical knowledge. The application would be written from a small pool of resource from within ONS Social Survey, which would influence the range of possible technical solutions.

3. Electronic Learning Questionnaires (ELQ)

3.1 Background

When BDSS took control of the Electronic Learning Questionnaire (ELQ), it became apparent that the ELQ report writing system provided an excellent opportunity to demonstrate the possibilities of Blaise Datalink and API. The reasons for this were:

i) The current system requires users to formally request our Information Management (IM) department to manually run an application to create the ELQ report. This involves filling out an IM request form, introducing a bureaucratic overhead. Ideally, users should be able to run the reports direct from their PC's, with no IM intervention;

ii) The ELQ report system is isolated from other business processes, as it is only reading data from returned ELQ case data and outputting it into a report format;
iii) A complete solution could be provided using the available technical skills available within BDSS;
iv) A standardised ELQ report across all the surveys could be easily provided, replacing the different formats that currently exist.

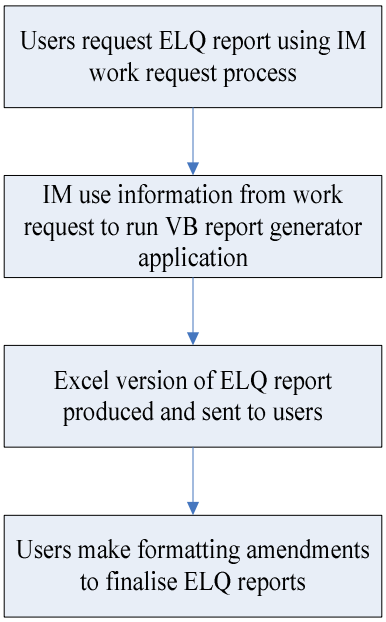
Field interviewers are sent a survey specific Blaise ELQ before they attend a survey briefing. The ELQ must be successfully completed for the interviewer to be eligible to attend the face-to-face briefing. The questionnaires are designed to test the interviewer's knowledge of a survey and key processes involved in working on it, and the interviewer has 2 attempts to answer each question correctly. If an interviewer answers a question incorrectly on the first attempt, they are directed towards the relevant section of their survey specific instructions. A report is generated which highlights the most problematic questions, which the trainer at the briefing uses to focus the session on problem questions and survey procedures.

For the purposes of this paper, the Household Assets Survey (HAS) ELQ has been selected. The HAS ELQ will test interviewer's knowledge on selected questions from the survey, covering pensions, mortgages and investments, as well as different scenarios that they may encounter. A list of questions included in the HAS ELQ can be found in Annex 1.

3.2 Current system for requesting ELQ reports

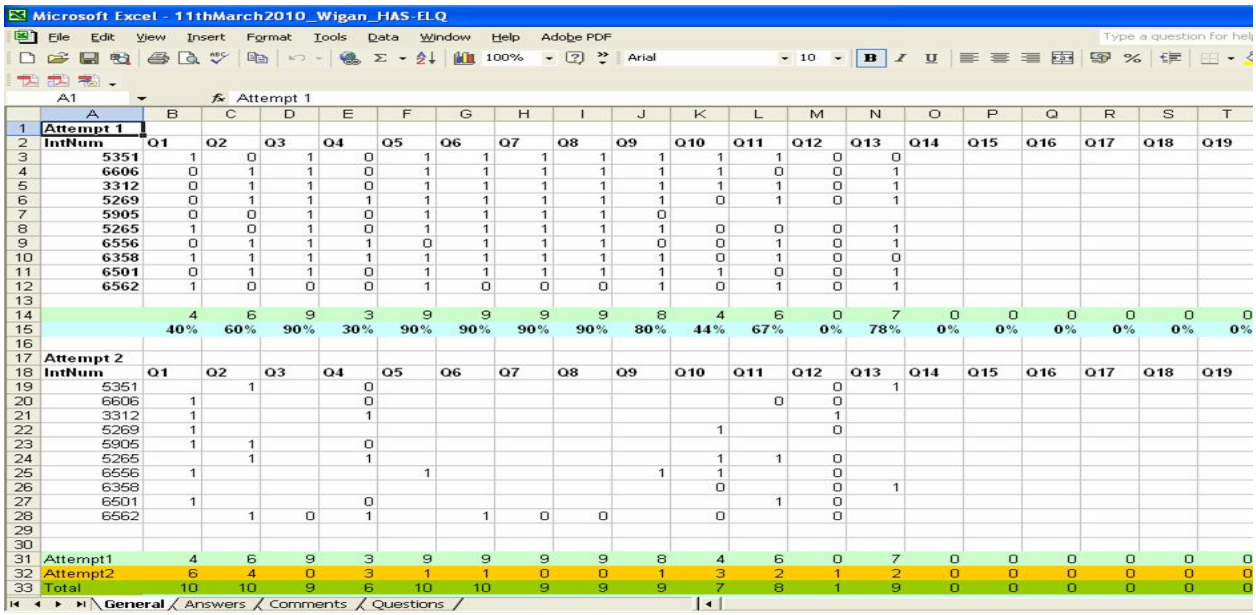
When an interviewer has completed their ELQ, the questionnaire is transmitted back to the office. The results from the ELQ are extracted by our Information Management Directorate from Blaise, and using a Visual Basic application which calls Manipula, the data is output to Excel spreadsheet format. Once the reports are produced, the Field Office team responsible for the survey will format the output into something that is appropriate for analysing. The process flow, shown in Figure 1 below, details how the ONSSS Field Office team produced ELQ reports.

Figure 1. ELQ report process



As the process flow demonstrates, there are a number of steps involved to produce the reports, each requiring some manual resource. An example report produced by our Information Management Directorate is shown in Figure 2.

Figure 2. Existing ELQ report



Microsoft Excel - 11thMarch2010_Wigan_HAS-ELQ

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Attempt 1																			
2	IntNum	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19
3	5351	1	0	1	0	1	1	1	1	1	1	1	0	0						
4	6606	0	1	1	0	1	1	1	1	1	1	0	0	1						
5	3312	0	1	1	0	1	1	1	1	1	1	1	0	1						
6	5269	0	1	1	1	1	1	1	1	1	0	1	0	1						
7	5905	0	0	1	0	1	1	1	1	0										
8	5265	1	0	1	0	1	1	1	1	1	0	0	0	1						
9	6556	0	1	1	1	0	1	1	1	0	0	1	0	1						
10	6358	1	1	1	1	1	1	1	1	1	0	1	0	0						
11	6501	0	1	1	0	1	1	1	1	1	1	0	0	1						
12	6562	1	0	0	0	1	0	0	0	1	0	1	0	1						
13																				
14		4	6	9	3	9	9	9	9	8	4	6	0	7	0	0	0	0	0	0
15		40%	60%	90%	30%	90%	90%	90%	90%	80%	44%	67%	0%	78%	0%	0%	0%	0%	0%	0%
16																				
17	Attempt 2																			
18	IntNum	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19
19	5351		1		0								0	1						
20	6606	1			0							0	0							
21	3312	1			1								1							
22	5269	1									1		0							
23	5905	1	1		0															
24	5265		1		1						1	1	0							
25	6556	1				1				1	1		0							
26	6358										0		0	1						
27	6501	1			0							1	0							
28	6562		1	0	1		1	0	0		0		0							
29																				
30																				
31	Attempt1	4	6	9	3	9	9	9	9	8	4	6	0	7	0	0	0	0	0	0
32	Attempt2	6	4	0	3	1	1	0	0	1	3	2	1	2	0	0	0	0	0	0
33	Total	10	10	9	6	10	10	9	9	9	7	8	1	9	0	0	0	0	0	0

There are several limitations to this report and the users would make extensive manual changes to it to make it suitable for distribution to the interviewer management teams. The main problem with the report was that there was no summary page identifying the key points from survey, it was not survey specific, it was cluttered, hard to read quickly and did not provide them with all the information they needed.

One major advantage of using Datalink to move the ELQ data into relational database storage is that it opens up the possibility of using off-the-shelf packages like Crystal Reports to produce tailored reports from the data.

4. The re-engineered ELQ reporting system

4.1 Objectives and benefits of the new ELQ reporting system

The main reason for re-engineering the ELQ reporting system is to reduce the amount of manual resource required to produce the reports. A successfully automated reporting system would enable Field Office users to have direct access to ELQ data from their desktops, and generate standardised reports from a standardised user interface. With more pressures on the resources of survey teams, automating more of their processes will enable them to continue delivering high quality outputs. Automating the systems will also release IM resource, enabling them to allocate more time to their core application support role rather than running business processes for customers.

Implementation of the new application would introduce a common format of ELQ reports across all surveys, reducing the time Field Office spend on standardising their outputs for analysis. It will also reduce the training requirements as Field Office staff move from supporting one survey to another.

Standard automated reports will enable quicker analysis of results, highlighting possible issues with questions. It will also enable the face-to-face briefings to be targeted at areas that the interviewers are struggling with, which should lead to better data being collected in the field.

The re-engineering project also provides an opportunity for the BDSS team to develop Blaise Datalink and API skills. This will allow us to take a fresh look at existing business processes within ONSSS, with the possibility of delivering cost effective applications to customers.

4.2 Developing the new application – Stage 1

The data from completed ELQ questionnaires returned by interviewers is already available in a merged Blaise database, created as part of the IM caseflow system. This will be the source of the ELQ data to be used in the reporting system.

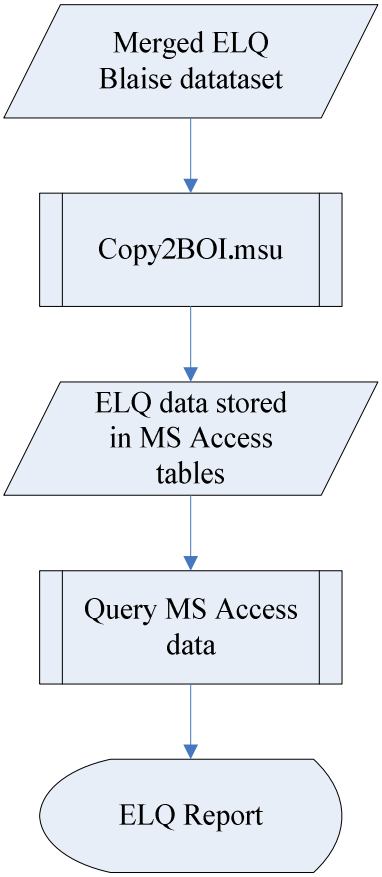
We decided on using Visual Basic to write the application, with Microsoft (MS) Access as the data storage method being queried by the application. These decisions were based on the relevant skills that were readily available within the BDSS team.

A BOI file was created to establish a link between the Blaise ELQ datamodel and the ELQ MS Access tables. The MS Access tables are based on the Blaise ELQ dictionary, and the data partition type selected was flat with blocks.

Using the merged Blaise ELQ dataset as the initial data source, the VB application runs a compiled Manipula script (Copy2BOI.msu) to copy data from here to MS Access tables. With the ELQ data now in MS Access, a simple SQL select query is run from within the VB application to retrieve the data. The application currently uses the in-built VB report writer to generate and display the data report on screen, with the option of printing. However, we are investigating using Crystal Reports to try and avoid maintaining VB code that makes direct reference to ELQ questions, thereby introducing more flexible report writing.

Variations on the SQL select query embedded in the VB application were introduced to enable data to be displayed either by briefing date of briefing location, depending on parameters input by users. The different outputs were achieved by use of different WHERE clauses within the SQL select queries.

Figure 3. ELQ reporting application process



After giving a demonstration of the first version of the application to field Office staff, the response was favourable. However, they ideally wanted to display not just the responses to the ELQ questionnaire, but also a summary page displaying those interviewers who had not yet responded, the number who had, the question causing the most difficulty at attempt 1, questions with in-correct answers after two attempts. In effect, a method of creating a dynamic sample would need to be integrated into the application, to enable responders and non-responders to be displayed on the ELQ reports.

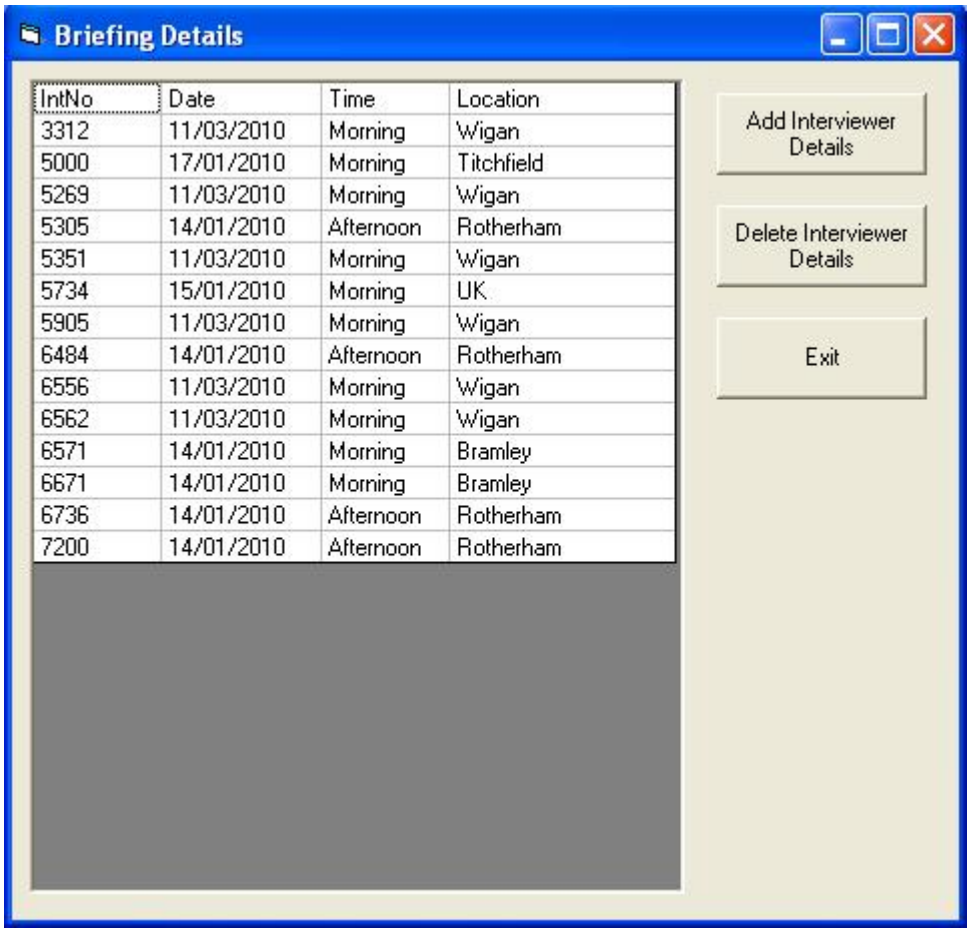
4.3 Developing the new application – Stage 2

To create the sample of interviewers for each briefing, a user interface was designed for the VB application. This interface would enable the user to enter:

i) The interviewer number
ii) Briefing date
iii) Briefing time i.e. morning or afternoon
iv) Briefing location

The briefing details entered through the user interface are stored in a text file. The Blaise API is then used in the application to copy the details stored in the text file into a Blaise database. This is achieved by using the ASCII file setting object and the copyfields method of the Blaise database manager object.

Figure 4. Creating the sample for interviewer briefings



A Blaise BOI file was created to enable a link between the briefing data input through the user interface, and the MS Access BriefingDetails table. Initially a full BOI file type was specified. However, testing revealed that if the data partition type was flat with blocks, when a record was deleted from the source Blaise dataset containing the briefing details, a runtime error in the VB application occurred at the point where the SQL DELETE statement is invoked. Consequently, the BOI file type was changed to Data only. With only a data table being created, the SQL DELETE statement successfully deleted the selected record.

On exiting the Briefing Details screen (shown in Figure 4 above), the application then runs a compiled Manipula script that updates the BriefingDetails.boi updatefile with the data input by the user.

The BriefingDetails table in MS Access, updated with the briefing data supplied by the user, can now be joined (using the primary key field IntNo) with the MS Access ELQ table containing the returned data. The SQL select statement embedded in the VB application makes use of the LEFT JOIN command,

enabling all selected interviewer numbers to appear in the final report, irrespective of whether they have returned data.

The results of running the re-engineered ELQ report system can be seen in Figure 5. In this example, the first 6 interviewers in the list have returned ELQ data for the date 11/03/2010. The date would have been a user supplied parameter, and is used in the construction of the SQL query embedded in the application. The ELQ data table in MS Access has an Administration block, which includes a date field, and this is used to retrieve the relevant cases filtered by the supplied date. The seventh case (9999) has not yet returned a completed ELQ questionnaire after attending the briefing on this date. However, the results of using the LEFT JOIN in the embedded SQL syntax can be clearly seen, as all interviewers being recorded in the BriefingDetails table as attending the briefing on 11/03/2010 are displayed. As a result, users can clearly identify non-responders.

Figure 5. The end result

ELQ Report Printed on: 19 August 2010 16:00

Briefing Date: 11/03/2010

Attempt 1

IntIlum	Q01	Q02	Q07	Q08	Q09	Q10	Q11	Q13	Q14	Q16	Q17	Q18	Q19	%
3312	0	1	1	0	1	1	1	1	1	1	1	0	1	77
5269	0	1	1	1	1	1	1	1	1	0	1	0	1	77
5351	1	0	1	0	1	1	1	1	1	1	1	0	0	69
5905	0	0	1	0	1	1	1	1	0					38
6556	0	1	1	1	0	1	1	1	0	0	1	0	1	62
6562	1	0	0	0	1	0	0	0	1	0	1	0	1	38
9999														
	33%	50%	83%	33%	83%	83%	83%	83%	67%	40%	100%	0%	80%	

The new report also allows users to easily identify the most problematic questions, which can then be addressed at the interviewer briefing.

Annex 1

Question	Question Text	Possible Question Responses
1	The HAS covers a number of topics relating to assets and debt. Which of the following is NOT within the scope of the HAS?	- Business Success - Government spending patterns - Consumer credit - Value of accounts and investments held overseas
2	You are about to start an interview (at a household interviewed at a previous wave) and you find the case on your laptop is corrupt and cannot be used. Do you;	- Use a training case to carry out the interview - Open a second household on another address - Leave and ask for the case to be re-scattered - Swap with an ineligible address

Question	Question Text	Possible Question Responses
3	The DWP uses the HAS to find out?	- About peoples business assets - About wealthy individuals and investment decisions - The value of peoples property - If people are saving for retirement
4	What is 'The Boost'?	- The £10 voucher to help response - The respondents who join a household - New addresses added to the sample - Multi households addresses
5	How many addresses are in a follow up quota from a previous wave?	- Less than 13 - 13 - More than 13 - 26 - Variable
6	The OSM agrees to interview but is not the HRP. The HRP and spouse refuse to take part in the survey. Do you?	- Interview the OSM and take proxy details for the HRP/Spouse - Interview the OSM only, as we are following individuals - Code the case as a refusal to interviewer - Code the case as ineligible
7	You expect to find the Smith family at the sampled address, but you find it empty and boarded up. The first thing you should do is;	- Contact the FEL and ask for the respondents new address - Outcome code the address as you found it, vacant - Try and find out where the respondents have moved - Code the case as moved away, address unknown
8	You find a 15yr old at the previous wave, now 17yrs old has moved out and is now flat sharing with five friends a couple of streets away. Should you;	- Open a second household and make a note of the new address for future allocation - Exclude, no longer a household member at the sampled address - Open a second household and interview the 17yr old and all his friends at the new address - Open a second household and interview the 17yr old only, as an original sample member
9	What are the extensions of time or extra visits rules on the HAS?	- No approval is required - Agreement is required from the Field Office/Survey Management area via your Field Manager - Agreement is required from your Field Manager - None permitted
10	When would you open a second household for a follow-up address?	- When the original household has moved to a second address - When some of the people have left the address - When another address/household in the quota is corrupt - When you discover a multi-household
11	Which of the following should you do if a respondent asks for clarification of an Opinion question (denoted by an * at the beginning of the question)?	- Rephrase the question - Repeat the question as it is written - Give your own interpretation of the question - Remain silent in hope respondent will give an answer
12	A household has taken out a loan secured on their main residence in order to help purchase a city-centre apartment. The payments they make partly cover interest and partly pay off the capital of the loan. How should this be coded at MType (Type of mortgage)?	- An endowment mortgage or loan - A repayment mortgage or loan - Another type (not listed in the coding frame for this question) - None of the above - this loan should be recorded in relation to the other property

Question	Question Text	Possible Question Responses
13	At GCOll 'Do you (or anyone in your household) own any collectibles or valuables - such as antiques, artworks, stamps etc - including items stored or kept elsewhere?' the respondent tells you that they have a number of antiques all kept at their main residence. They also have some valuable paintings in their second home in Spain. Which, if any, of these items should be included in their estimate of current market value at GcolV/GClVb?	- None - Just the antiques - Just the paintings - The antiques and the paintings
14	Thinking about occupational pension schemes, which of the following options of occupational pension schemes applies when a respondent says the value of their pension relates to the value of contributions made over the years as well as their salary in the final years before retirement?	- Money-purchase scheme - Salary-related scheme - Hybrid scheme - None of the above
15	Still thinking about pensions, which of the following statements reflects the true definition of a Group Personal Pension (GPP)?	- It is an occupational pension provided by the employer - It is an occupational pension scheme organised by an employer but provided by a financial institution - It is a type of private pension - It is a special type of stakeholder pension
16	When collecting information on savings and investments, a respondent tells you that they have a Basic Post Office Account. What sort of account should this be included as at FinvTy ('Types of accounts and investments you currently have')?	- Current account - Saving/deposit account - Individual Savings Account (ISA) - None of the above
17	When completing the financial assets section of the questionnaire, a respondent tells you that the Offset Mortgage covered earlier in the survey has a savings account element. What sort of account should this be included as at FInvTy ('types of accounts and investments you currently have')?	- Current account - Savings/deposit account - Investment Bond - Other financial asset
18	When interviewing a married couple you are told that the husband and wife hold a credit card account jointly in both names. Their daughter is an additional card holder. Against whose name should the value of any outstanding debt on the account be recorded?	- All - The husband and the wife as joint account holders - The husband or the wife, whoever is interviewed first - The husband or the wife, whoever is interviewed first, and the daughter"
19	Which of the following examples would qualify as a FULL individual interview on the HAS?	- The individual interview has been completed up to and including 'Trusts' - The individual interview has been completed up to and including 'Inheritance' - The individual interview has been completed up to and including 'Non-mortgage debt' - The individual interview has been completed up to and including 'Employment income'

Using Google® API's and Web Service in a CAWI questionnaire

Gerrit de Bolster, Statistics Netherlands, 27 September 2010

1. Introduction

From the survey department of Traffic & Transport in Statistics Netherlands came the question if it's possible to use Google® Maps in their CAWI questionnaire for pin-pointing locations. To determine this, a prototype was developed in which the use of Google® Maps was realised. It appeared that not only the Google® Maps API was needed but also the language API and the Maps Web service. This paper describes how this was done.

I must warn you that this paper is about the technology involved. I tried to explain this technology in a most simple way, but I understand that not everybody is able to grasp my solution. This paper is meant for those who want to do more with BlaiseIS than the basic things and I hope it shows that there are a lot of possibilities to extend BlaiseIS questionnaires with proprietary functionality.

Although they specifically asked to investigate the feasibility of using Google® Maps for searching on locations, the survey department of Traffic & Transport never asked the data collection department to introduce this solution in their questionnaire. One of the reasons they came up with later is that the use of Google® Maps would increment the burden for the respondent filling in the questionnaire.

2. Searching locations

The Internet questionnaire for the survey on road transport from the department of Traffic & Transport contains a lot of questions about locations. The location of the home base of the vehicles is asked as well as the place of departure and destiny of the trips made by these vehicles. Furthermore the respondent must fill in the locations where a load was picked up and where it was delivered. In the current questionnaire the respondent can use for these locations lookups on countries and places. Once a country is selected, the lookup for places (towns, cities) is filtered by this country so only the places within this country will appear in the lookup list. As the survey is on international transport, foreign countries are included. As it was expected that respondents will have trouble to spell the names of the foreign places correctly, a trigram-search was applied for the places. If available the respondent could also provide a zip code of the location asked.

Once the filled in questionnaire is received, the given locations are translated into coordinates (latitude/longitude). For this (automatic) translation a web service from a commercial provider is used. If a location could not be translated using this web-service, it should be done manually. As it appeared this happened more than was expected costing too much human resources. Even in some cases the web-service used was not accurate enough.

For these reasons they asked to investigate if it was possible to produce coordinates in the questionnaire selected by the respondent by using a "maps" interface. Of course, it was obvious to try to use Google® Maps as it is the most well known solution in this field.

3. BlaiseIS questionnaires and JavaScript

Blaise Internet questionnaires are based on the principle of client/server. That means that a respondent is entering data on the client side (with the browser on his own computer) and sending that to the server (in our office). This data is subsequently processed on the server by BlaiseIS components and Active Server Pages (ASP's) producing an updated screen that is sent back to the client. The way the data is processed is

determined mainly by the so-called rules in the Blaise data model. The screen that is sent back to the respondent is written in the universal Internet language HTML, normally enriched with JavaScript.

If the respondent is activating a lookup of any kind, the resulting window (screen) should appear in his browser so he can manipulate it and select the text or code necessary. That means it should appear on the client side and not on the server side. Furthermore, the selected text or code should be written to the input line of the question involved, visible for the respondent in his active screen. In other words: client side.

Within the world of (classic) Internet there is only one global accepted way to make your Internet page have some dynamics: the use of JavaScript. Although today there are other means possible (Java-applets, Flash, Silverlight) JavaScript is still the most used solution. To write data from one window to an input line in another window with JavaScript it is necessary to have 2 things: a pointer to that other window and a name (identification) of the input line. If a window was opened by an action in another window, the pointer to that "parent" window is automatically available (in JavaScript it's called "opener"). But how about the name of the input line? In BlaiseIS every visible field in the questionnaire is given a name at the moment the screen is constructed and sent to the browser of the respondent. It is impossible to get this name before e.g. at design time. The only way to obtain this name is, at the moment the window is opened in the respondents' browser to invoke a JavaScript function to scan the HTML source of the active window. This function was developed already before for another type of search (hierarchical) and stored in a small JavaScript library that is automatically included in every questionnaire using the caption of a (camouflaged) label in the Blaise Menu File (.bmf). The function is invoked by a JavaScript command that is activated the moment an image is loaded at the opening of a window. This command is included in the question text of a separate question in the Blaise sources that has only one visible part: an image in the shape of a "Search" button!

The same function is used to give that "Search" button an index number so it is included in the so-called Tab-order. That is the order in which the parts of a browser window are focussed using the Tab key. Once it is focussed it can also be activated by a key and not only by the mouse.

4. Create your own search facility

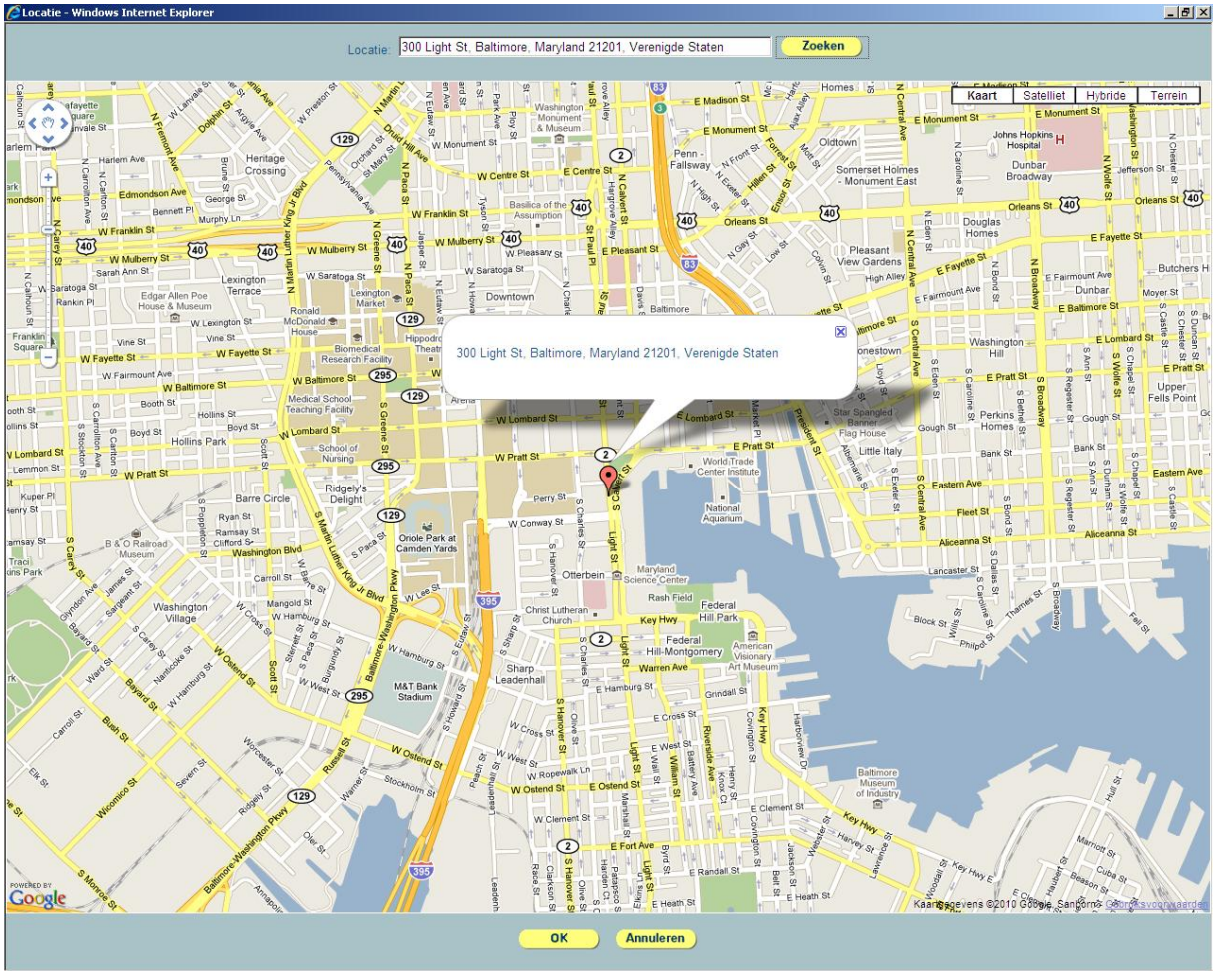
So how do you create your own search facility for your BlaiseIS questionnaire? As was described in the previous section, using JavaScript you can open a new window with the possibility to write data to an input line in the active (visible) page of the questionnaire. This window is opened by clicking on a button image in the questionnaire. As a result it is opened referring to the URL of that window. This URL can e.g. be an URL of an Active Server Page (ASP). In an ASP you can run a script (VBScript by default) invoking Blaise(IS) or other installed components. By the way, BlaiseIS consists partially of ASP's. To create your own search function, you must therefore create your own ASP and include it in your BlaiseIS questionnaire adding it to the file section of the Blaise Internet Specification file (.bis). In this ASP you can manipulate data as much as you want, e.g. opening a Blaise database on the data server and reading records from it or writing records to it. Applying this concept already several special (but generic!) functions have been created including the location search with the Google® Maps API.

As can be understood it is necessary to have the use of JavaScript switched to "on" in the browser if you want to use this special search function. Fortunately almost everybody has JavaScript switched on as without it the Internet is a very dull place. Although Google® still lets you use a simplified form of their Maps facility, it is a rigid solution that is very difficult to use. Just in case, if a respondent is one of the remaining few without JavaScript activated in his browser, he is informed that this search function is not working.

5. Google® Maps API

On the Internet there is a lot of information available regarding the implementation of Google® Maps functionality in a web application. See the section with references at the end of this paper for the home page of Google® Maps in English (also available in other languages). This was the way we learned how to create the location search with the Google® Maps API in JavaScript. The first location search function was created with Google® Maps API version 2 as it was the supported version at that moment. It was obvious that version 2 would not be the supported version much longer, so the solution was converted to version 3, what is the supported version today. There are a lot of examples available on the website making it quite easy to understand the different commands and functions and how to implement them. Basically the use is for free unless you want to include in a commercial environment. However, you can have a contract with Google® allowing you to use it in a different way. This is described in the Terms Of Service (TOS) also available on the Internet. As this location search has not been taken into production until now, Statistics Netherlands have not yet a contract with Google® for this use at the moment. As I understood we do have a contract with Google® for another type of use: that in publications with a geographical nature.

If the documentation is still not enough to answer all your questions, there is also a very active Internet forum on which Google® developers are trying to help you.



The Google® Maps API is a set of JavaScript commands and functions that enables you to manipulate a map in the same way as it is used on the website of Google® itself. You can use navigation buttons, zoom-buttons and switch from map to satellite and so on. Too much to mention. More important is the fact that you can place a marker on the map and move it over the map. To this marker you can attach a small window with information (address) where the marker is standing on the map. Finally, you can even retrieve this address information as it is stored in an object (cluster of data). This object contains a very complicated tree of address parts. It took quite some time to figure out how to retrieve from the object the information that was needed: the street address, the town and the country. This information can than be written to the input line in the BlaiseIS questionnaire by the respondent by pressing an OK-button that was added to the bottom of the screen.

In some cases the marker could be placed on coordinates that did not produce a valid address. Obviously this was the case in the middle of the ocean. But not only there! Google® appeared to be "political sensitive". Certain regions on our planet where there is still a discussion on-going about ownership and such are also producing empty addresses or non-specific addresses. Examples are Kosovo, Western Sahara and Northern Korea. Maybe with the exception of Kosovo, these regions are normally not involved in answers on questions from our questionnaires. No Dutch trucker has been seen there and they are also not very popular holiday resorts.

6. Languages: a nightmare!

Everything appeared to be very easy. There were hardly any problems to get things working using the examples on the website of Google®. This "happiness" lasted until version 1 of the location search was tested! Moving around the maps of the world all kind of strange characters appeared in the address part of the window. Not all of it, but just parts of the address. This happened although the language parameter was specifically set to Dutch (nl). Depending on the country the addresses were shown in Greek, Chinese, Arab or Cyrillic characters. Quite unreadable for our respondents. Although there are not many transporting firms in The Netherlands providing transport services by road to China, Greece and the Balkan countries are within reach. Furthermore, these foreign addresses were stored in UNICODE. This is a character set using 2 characters for one. Blaise, however, only supports the basic ANSI code set, one character at a time. Writing these addresses to the input line in the BlaiseIS questionnaire resulted in unreadable characters.

But, no panic, Google® also supports an API with which you can translate words and sentences. This nice (also JavaScript) API even includes an auto detect of the language of the text to be translated. However, this auto detect does not work when the text to be translated is a mix of different languages. So the language parameter was set to "local" producing the address including the name of the country in the local character set. It appeared that our good friend Murphy was working for Google® too. Suddenly the addresses were shifted all over the world. Cities in Croatia were moved after translation (according to Google®) to Australia, the city of Athens, Greece was translated into Athens, USA and so on. It was not Google® Maps that gave the wrong addresses; it was the translation API that mixed up things.

The problem was finally solved with a mixture of functions, a combined approach. In Google® Maps the language was set back to Dutch producing an address in different languages. In the translation command the language of the text (address) to be translated was set to the specific language based on the country code retrieved from the address object. Not all the countries were included, only those with different character sets. Even after this change some countries with a small number of different characters (Slovenia, Croatia, etc.) kept producing strange translations of addresses. Finally these countries were removed from the translation set and a function converting their "special" characters to characters within the ASCII character set was added. As a last resort a final addition was made. In some rare cases it could happen that there were still some "strange" characters left over (missed by all the other conversions). In

that case the coordinates were retrieved from the address object and written back to the input line of the questionnaire.

It worked! The language nightmare was over.

7. Google® Web service

As was mentioned in section 2 the goal was to retrieve coordinates, not addresses. The address object produced by Google® Maps contains these coordinates (latitude/longitude). But respondents are not really trained in reading coordinates. Therefore, in the questionnaire the location produced by Google® Maps should be a readable address. Besides, in Internet questionnaires the search option is optional. If a respondent is sure of the address he can decide just to type it in without invoking the search function. The conversion to coordinates should therefore take place after the question has been filled in. In other words: on the server in our office.

Besides the interactive Maps API, Google® also provides a web service to convert addresses to coordinates. A link to more information about this web service can be found on the homepage of Google® Maps (see Section 10 References). A web service can simply be invoked by calling its URL with some parameters. It normally responds by sending back an xml-stream. To call an URL from a BlaiseIS questionnaire ca be done using a so-called HTTP-request. For that purpose an Alien procedure was created calling a .NET component. In a Microsoft® environment a class is available for this purpose. This is how it looks:

```
<Runtime.ComVisible(True)> _
    Public Sub GetResp(ByVal pURL As BlAPI3A.Field,
      ByVal pParam As BlAPI3A.Field, ByVal pResp As BlAPI3A.Field,
      ByVal pStatus As BlAPI3A.Field)
        Dim objRequest As New WinHttp.WinHttpRequestClass
        objRequest.Open("GET", pURL.Text + pParam.Text, False)
        objRequest.Send()
        pResp.Text = objRequest.ResponseText
        pStatus.Text = objRequest.StatusText
        objRequest = Nothing
    End Sub
```

By making the question in which the address from the search function is written (or directly typed in by the respondent) so-called “critical”, the rules on the server are invoked when this question looses the focus. In these rules the web-service is called through the Alien procedure and returns an xml stream in a Blaise string field containing the coordinates. This string is subsequently “ripped” with standard Blaise commands to get hold of the coordinates. Using a status field warnings or errors can be given when an address could not be converted. As it is server based, the call to the web service is going outside to the Internet. The fire-walls should therefore be configured to let this call pass through.

8. Look and feel

Using the functions of the Google® Maps API the search function is made as user friendly as possible. A respondent can double click on any place on the map and the marker jumps to that place showing the address in the info window. It is also possible to “grab” the marker with the mouse and drag it to a new place. Furthermore, by clicking on the search button on the top of the search screen, the screen is centred on the marker wherever it is situated. The whole map can be moved by dragging it with the mouse or using the navigation buttons. As Google® Maps is a very used function on the Internet, it looks quite familiar and respondents will probably not have any problem using it.

Finally the search window is given the same standard look and feel of the other BlaiseIS search functions as we designed them at Statistic Netherlands.

9. More functions

The same technology that is used for the location search is applied for other search functions too. At this moment a hierarchical search function is available which is designed for large coding tables like the Combined Nomenclature as used in the European Intrastat survey. Lately a keyword search was added that can e.g. be used for coding occupations. The concept of this popup search function makes it possible to develop new functions in a short time. Finally based on the same concept but applied in a different way a function for a popup question was added to the "family" that can also be used as an alternative for the remark function.

10. References

Google® Maps homepage: http://code.google.com/intl/en/apis/maps/index.html

Google® Maps terms of service: http://code.google.com/intl/en/apis/maps/terms.html

Google® Language homepage: http://code.google.com/intl/en/apis/ajaxlanguage/documentation/

Google® Forums homepage: http://www.google.com/support/forum/

CATI Followup Application: Resolving Communications Links Between Applications Using XML

Joseph Cummings, Statistics Canada

Abstract:
Designing a Blaise application to be used in a Multi-Mode environment has resulted in a strong tool that can be used for CATI, CADE and failed edit follow-up. Having Blaise interface with external systems in this Multi-Mode environment has given us some interesting design challenges in terms of methods for transferring the data between the systems and evaluating the completeness of the data.

This presentation reviews the solutions we originally designed and where we see ourselves moving to in the near future. Within each of these phases we will review the pros and cons of each strategy and of any lessons we have learned in the attempt. Some of the interesting challenges and considerations we need to face involve managing the separation of different collection methods between different groups of users within a distributed environment. We also need to weigh the pros and cons of distributing the samples between different collection applications or managing users from different locations through one centralized sample.

The original design involved electronic collection over the web with the data stored in an SQL database. The SQL database had a mapping table within it where each field collected could be mapped to a field within the Blaise database. The collected data would be parsed nightly and loaded into the Blaise application where it was evaluated for completeness and accuracy.

The current design is similar but extracting the data into an XML format has replaced the parsing tool and has improved the speed of loading the data into the Blaise application. In addition, the evaluation method has evolved to be more robust and versatile. The application has also been improved within its Multi-Mode functionality by introducing the solutions we have for evaluating the loading of external data to the Data Capture flows and evaluation.

Computer Assisted Interview Testing Tool (CTT) - a review of new features and how the tool has improved the testing process.

Russell Stark and Rebecca Gatward – Survey Research Center, University of Michigan

1. Introduction

The CAI Testing Tool (CTT) is an application developed by the University of Michigan's Survey Research Center to manage and facilitate efficient testing of Blaise questionnaires.

CTT is a comprehensive tool designed to be used by all involved (programmers, project managers, and clients) in questionnaire testing and development from beginning to end. As Dascola et al. explain in their paper, presented at the 2007 International Blaise Users Conference, CTT was designed to improve the quality of CAI instruments through standardized testing procedures, reduce the cost of testing, and increase access to information about the CAI development process through preset and adhoc reports.

The tool has now been in use for around three years and a new version of CTT has recently been developed. This paper will outline recent updates to the testing tool, which include the capability to test Blaise IS questionnaires, will examine how the tool has changed the testing and development process, and will attempt to quantify how it has improved the quality of CAI instruments and reduced testing costs.

2. Background

Development of the CAI Testing Tool began 'in-house' in 2006 and the first production version was being used by project teams in 2007. The tool was developed with the aim of standardizing testing, increasing the overall quality of CAI through standardized testing procedures, reducing testing costs and increasing access to information about CAI development through standard and ad-hoc performance metric reports.

3. Key components of the CAI Testing Tool

The CTT consists of six main components, these are summarized below, a more detailed description of the tool is provided in Dascola et al (2007).

3.1 Automated testing tool

The CTT is a fully automated tool used to maintain a bug log for Blaise applications. This is a program that 'follows' the tester while they run through a Blaise application. The tool keeps a keystroke log, can capture screen shots, and automates the bug log and bug log item prioritization. Pre-loads (scenario information) can be specified to ensure all aspects of Blaise instruments are tested.

3.2 Pre-load builder

The CAI Testing Tool also includes a pre-load builder, which is used to enter or load any data required to test the instrument. The pre-load builder allows us to create a new or update a pre-load for a revised instrument, or delete an existing pre-load via an administrative data entry screen.

3.3 Random case generator

The random case generator feature tests questionnaire routing using test cases that have been created using random data. Using the output the programmer can determine if all the questions in the instrument have a response. The programmer can then investigate if the blank fields are due to a routing or programming error. The Random Case Generator includes some 'intelligence,' with preloaded information guiding what variables are not randomly generated, to allow for focused generation of case data and testing of targeted paths. This feature helps ensure every piece of the questionnaire has been

tested. The module produces an Excel file called frequency.csv, which lists the questions alongside the number of times each question has been answered.

3.4 Management

The 'manage problems' screen works as a functional dashboard to deal with issues. Its general purpose is to review and manage the bugs or enhancements reported during testing. The bottom half of the screen is primarily used by the Blaise programmer. Here they can view all information about the issue and update information about each problem for example, the problem status, open a screenshot or audit trail, and enter the date when a problem is fixed. This screen displays all problems, regardless of status. The addition of a capability to sort, move and filter columns is a great improvement in the newest version. Blaise and BlaiseIS problems are all listed on the 'manage problems' screen in the same way.

3.5 Reporting

CTT has a built-in reporting function providing the tester, coordinator and programmer the ability to generate specific reports. This helps each level of the testing staff to focus their efforts effectively. The ready to test report is designed primarily for testers. This report shows the tester which problems have been repaired, their locations and other relevant data. The tester can use this report as a guide to what to re-test and to confirm that problems have been solved. The 'Not Fixed' and 'Not Fixed High Priority' reports are designed for the Blaise programmer. This allows them to one-click a report of what they currently have to work on. We also display a number of summary reports that provide information on test performance metrics (e.g. Problems found, problems fixed problems remaining etc.). These are used by managers and testing coordinators to monitor testing progress.

3.6 Administration

The Administrative tasks tab in CTT contains a variety of upper-level functions. Managers or coordinators can add projects and set-up files for testing, assign testers to projects, attach specific priorities to different problem types, import master preloads and assign access rights. These administrative tasks are grouped in the Admin tab and then are shown as large icons on the 'ribbon'.

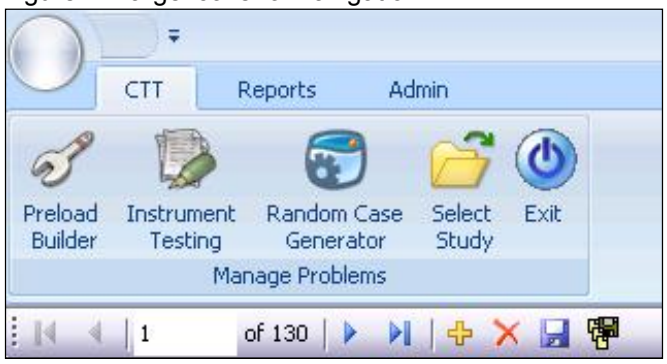
4. Recent developments to the CAI Testing Tool

Since 2007, we have continued to develop the CTT. The enhancements made are in response to feedback from users and the need to keep up to date with new developments in Blaise. The key developments are described below.

4.1 Updated user interface

Since the first version of the CTT was developed, we have updated the overall look of CTT using the now familiar 'Ribbon' interface. It allows for a smoother user interface, using large icons for navigation (figure 1) instead of dropdown menus from the title bar. We have also introduced tabs, allowing similar tasks to be grouped and separated. Users have reported that they find the new user interface and its updated look easy to navigate.

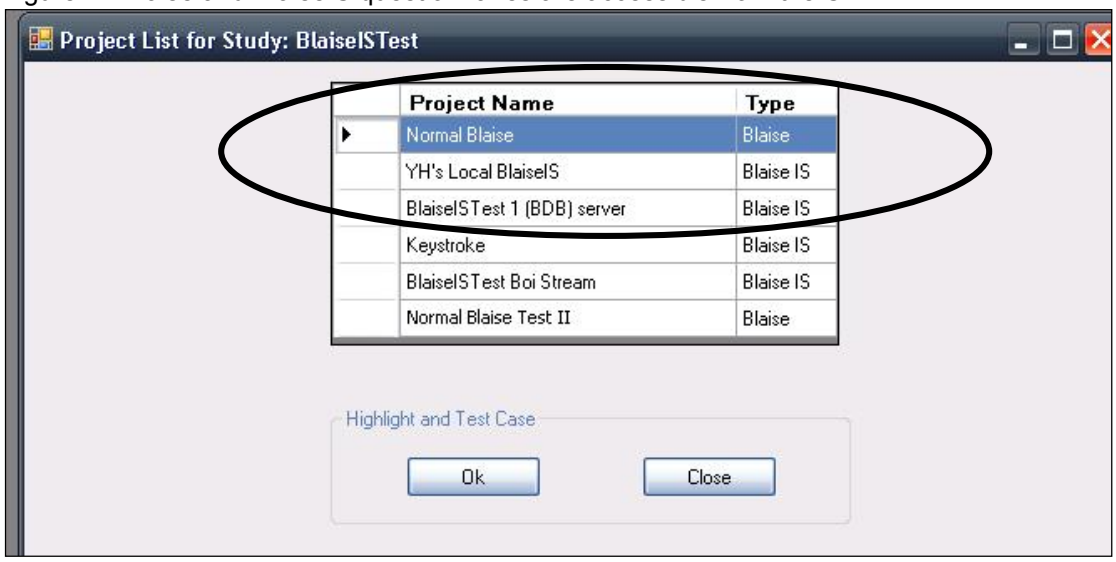
Figure 1. Large icons for navigation



4.2 Blaise IS

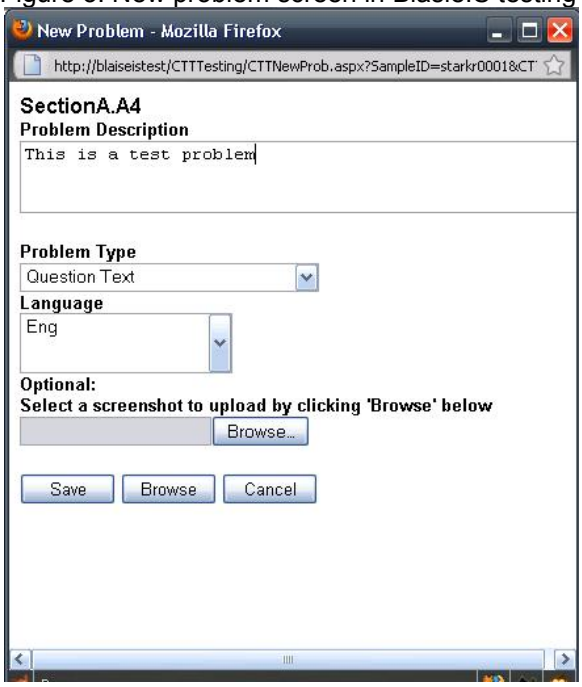
A recent addition to the functionality of CTT, is the ability to test both BlaiseIS web-based instruments and 'standard' Blaise instruments simultaneously (figure 2). The Blaise IS component has been designed to follow the functionality of the general Blaise testing tool as close as possible.

Figure 2. Blaise and BlaiseIS questionnaires are accessible from the CTT



The report problems screen, which users see most often, is nearly identical in both versions, as is the short-cut key (F8) used to call this screen (figure 3). This reduces the need for any retraining for users who were trained on the original system or the need for separate trainings for general Blaise and BlaiseIS testing.

Figure 3. New problem screen in BlaiseIS testing



The BlaiseIS testing tool has been designed to work in all major web browsers (IE, Mozilla Firefox, Chrome, and Safari). After a user has selected an instrument to test, they are prompted to select whichever browser they prefer (and have installed) (figure 4).

Figure 4. Choosing a Web browser for BlaiseIS testing

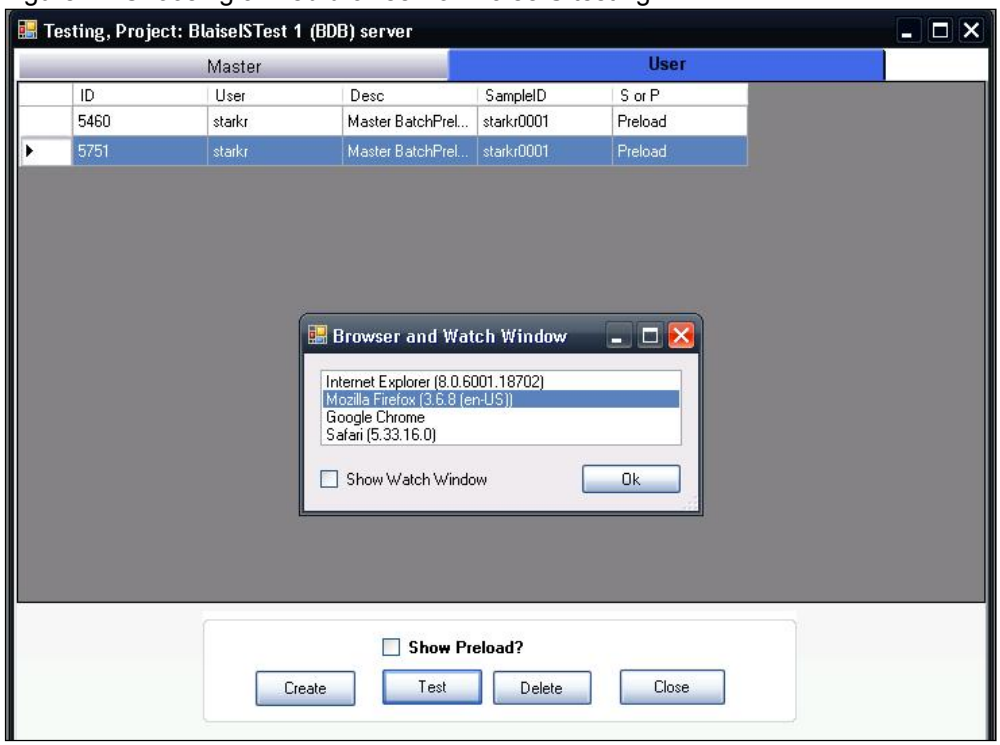
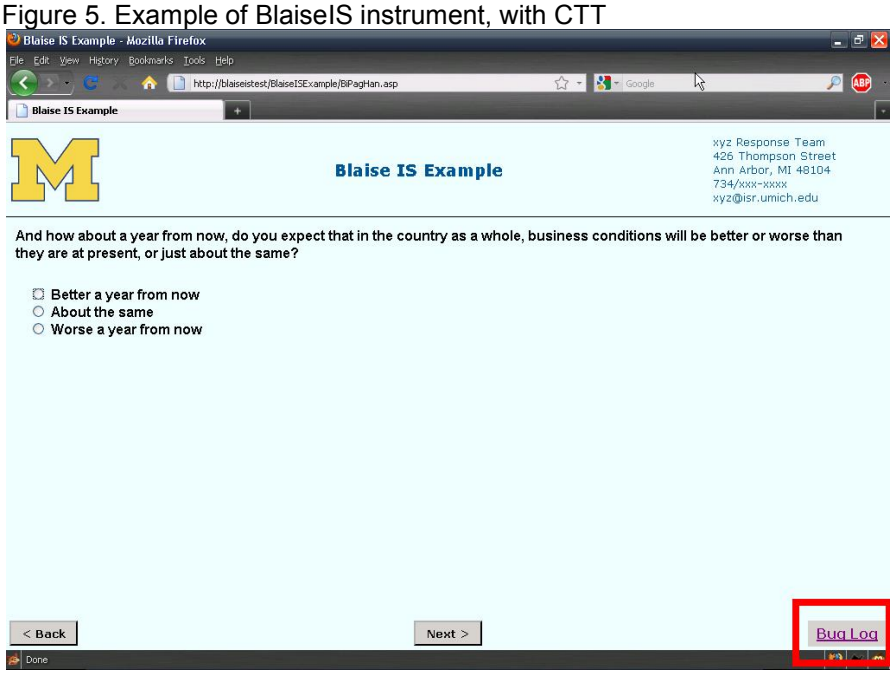


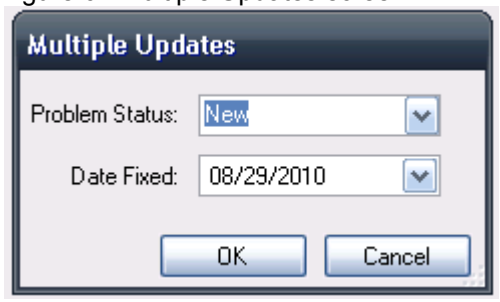
Figure 5. Example of BlaiseIS instrument, with CTT



4.3 Manage problems

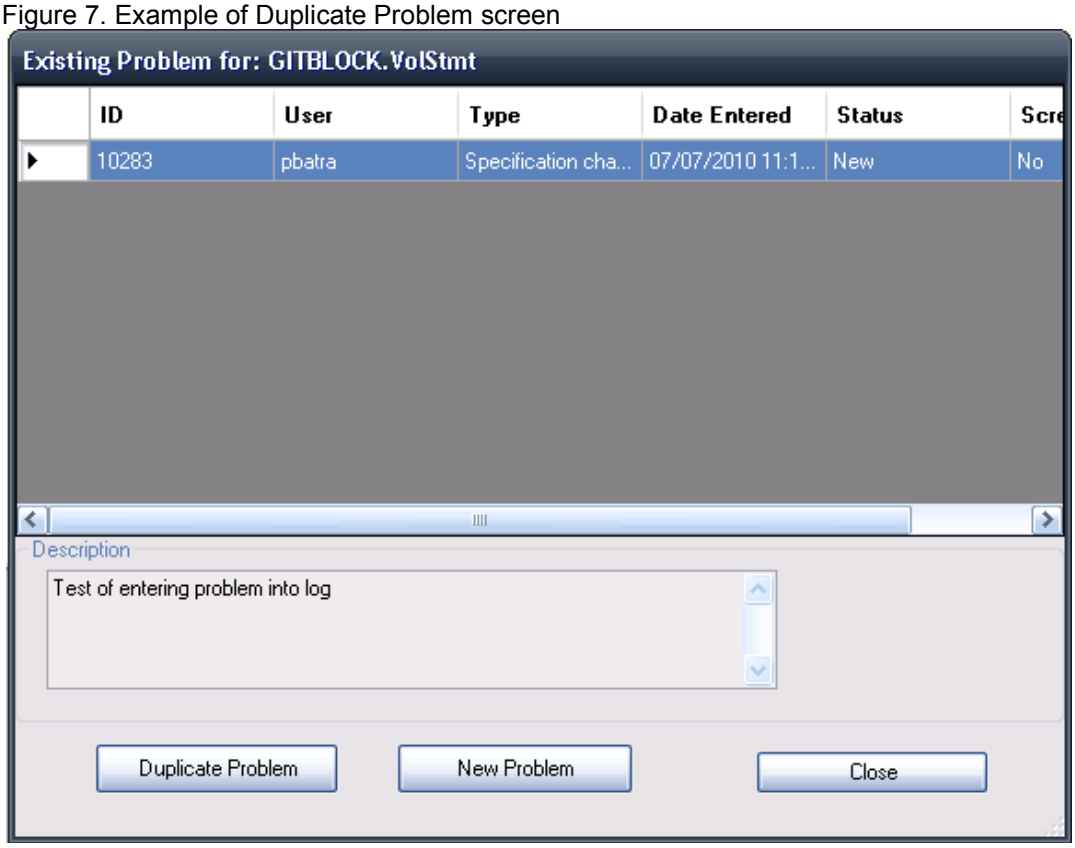
Many updates have been made to the Manage Problems screen. A wish list item from the first version of CTT was the ability to sort and filter information based on user requirements. This option is now functional, even allowing the user to resort each column to their needs. This main screen also now features a functional screenshot button. This permits the user to open a screen capture of the problem in question. Feedbacks from CAI programmers lead to the development of the Multiple Updates button (figure 6). This allows the programmer to update the statuses of multiple issues without having to save each time they updated a line.

Figure 6. Multiple Updates screen



User's access levels now determine what each level of user sees on the Manage Problems screen. Access level 1 (tester) cannot see the problems listed by default. This was put in to place to help keep the interface clean and uncluttered for novice testers. Access levels 2 (Testing Coordinator) and 3 (Programmer) allow users to perform Administrative tasks as well as view all issues listed.

Figure 7. Example of Duplicate Problem screen



5. How the CAI Testing Tool has changed the testing process

The CAI Testing Tool has been used extensively within the Survey Research Center since the first production version was completed in 2007. In order to understand how the CTT has changed the testing process, it would be useful to briefly describe how testing was carried out before CTT was developed. Pre-CTT, comments were recorded on one or multiple Excel spreadsheets. This meant that a lot of time was spent reconciling logs and trying to determine what the tester was trying to convey. Too much time was spent working on 'bug logs' and not enough on testing or developing the instrument. CTT integrates all of the functions for Blaise testing we need into one software package, and automatically gathers the testing information and outputs it to formats which can be used by all.

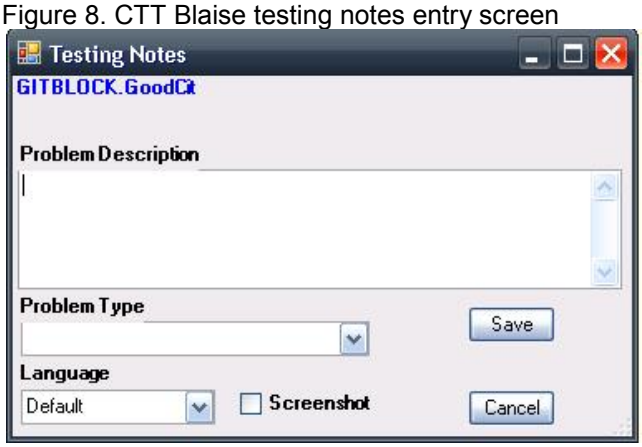
In this section of the paper we describe how CTT has improved the testing experience from each of the three main users perspective: tester, programmer and manager.

5.1 Tester

5.1.1 Direct access to the CTT

The main advantage of using the CTT for the tester is the ability to access the tool and record comments directly from the Blaise questionnaire (figure 8). Using a short-cut key, they can record comments with no need to jump between applications. The testers comments are saved directly into one master 'bug' log , the tool 'stamps' the comment with the field name/path, records the testers username and attaches any additional data – for example, a screen shot to the comment.

Figure 8. CTT Blaise testing notes entry screen



5.1.2 Existing problem

The 'existing problem' feature, eliminates duplication of effort – the tester is notified if a comment has already been attached to a field – they can then check if the existing check is the same as the one they have (figure 9a and 9b). When appropriate testers can then click a 'duplicate problem' button while testing inside Blaise, affirming they have recognized the problem also, but do not need to make separate note for it.

Figure 9a. Duplicate problem screen (Blaise)

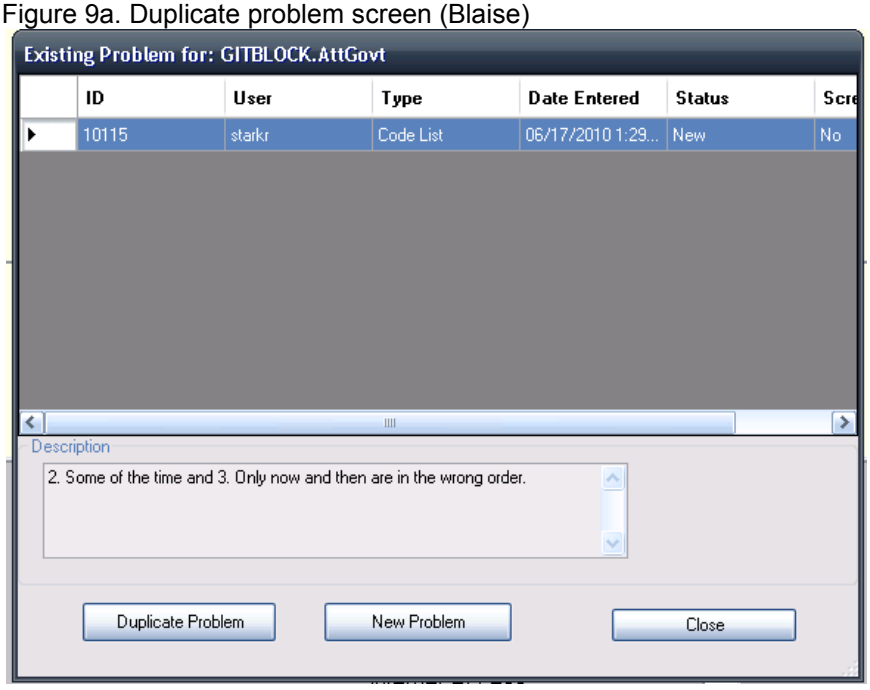
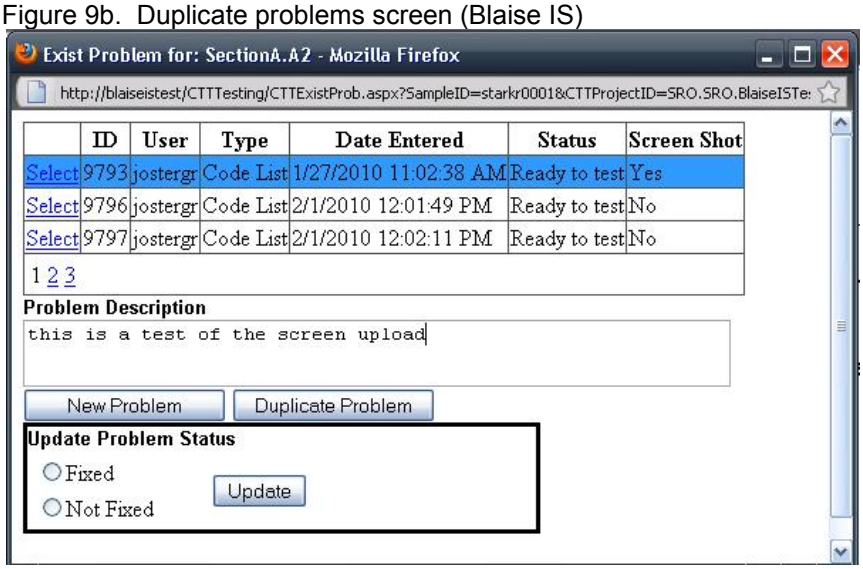


Figure 9b. Duplicate problems screen (Blaise IS)



5.1.3 Screen shots

Testers have found the screen shot option is particularly useful when commenting on screen layouts because it reduces the amount of text they are required to enter to describe the problem, in addition the programmer is able to see how the screen layout appears without entering the Data Entry Program (DEP) themselves.

5.1.4 Central creation of testing cases

The ability to create 'master scenario' testing cases has also saved time. Those with particular access levels can create 'master' cases and make them available to more than one tester. The tester can then select and save the cases to their own 'testing area'. This feature reduces time spent specifying scenarios and for testers recreating them, it also helps ensure all testers are using the correct testing parameters.

In further benefit of using a shared pool of master cases is that the cases can be prefilled to particular points in the questionnaire, this allows the tester to concentrate on elements of the questionnaire they have been assigned to test rather than wasting time getting to the point of testing. This is especially useful when testing effort has been divided between sections of the questionnaire or when clients wish to concentrate their testing on the questions they are funding.

5.1.5 Remote testing

One of the aims of the CTT was to allow remote testing. This facility has proved to be useful with not only off-site testers, such as field interviewers, but also with clients.

5.1.6 Sharing test cases

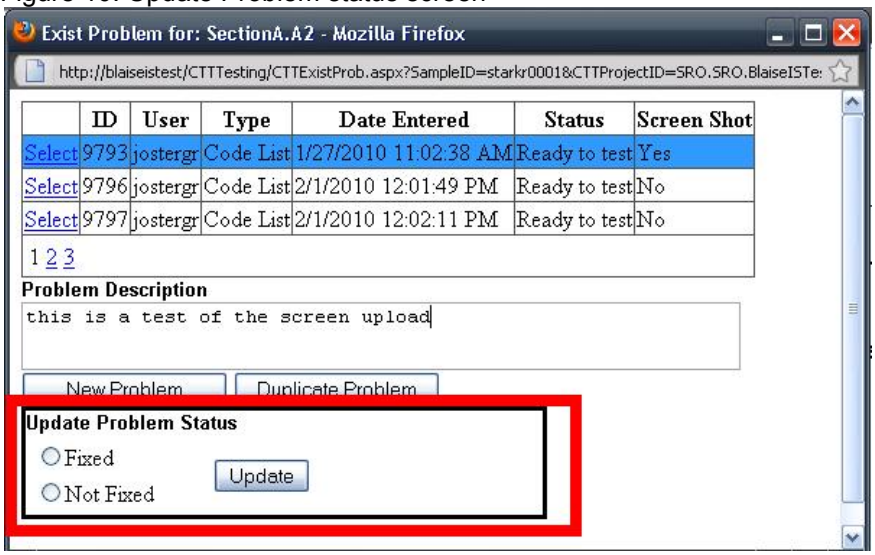
A further feature that has saved testers' time is the ability to share cases or scenarios with others. In the CTT, testers can save cases they have created whilst testing and those with a higher access level can create and save 'master' cases for others to use when testing. This eliminates the need for others to spend time recreating a reported.

5.1.7 Targeted retesting

When retesting the questionnaire the tester is able to access the 'Ready to Test' report. This report, in Adobe .pdf format, outputs a form displaying problems that the programmer has changed the problem status from 'New' to 'Ready to test' after they have repaired the problem. This lets the tester focus on re-

testing the specific problems in the instrument. After the tester has determined that the problem has been fixed, the tester will enter that information via the Update Problem status check box (figure 10).

Figure 10. Update Problem status screen



5.2 The programmer

5.2.1 Consolidating Comments

For the programmer, the key advantage is that all comments are stored in a consolidated list, in a standardized format with other details, such as field name and tester username attached. The programmer no longer has to decipher emails, Excel spreadsheets, scraps of paper, or remember to act on comments provided face to face. Having comments all in one place ensures programmers have all the information in one place and streamlines their editing time.

5.2.2. Specifying reports

A further key feature is the reporting function in the CTT. The programmer can either view or print standard reports or specify them. For example, programmers can select just the problems they have been assigned to fix or a general list of unfixed problems, and then work from a printed copy.

5.2.3 Identifying problems

Programmers have found that simply being able to view and manipulate the list of problems helps identify similarities in problems and can lead to further common problems being found. CTT gives programmers the more confidence in an instrument because they know it has been tested thoroughly and can release it with more confidence.

5.3 The Manager

5.3.1 Control of the testing process

The CAI Testing Tool enables the manager to have control over the following aspects of the testing process.

- Access to survey questionnaires or questionnaire versions is controlled by username. This ensures that the intended version of the questionnaire is used for testing and testers do not waste time working on an incorrect version.

- The manager can also control which aspects of the questionnaire are tested or the scenarios used by creating master cases.
- If necessary, the manager can control which problems are fixed first by prioritizing cases.

5.3.2 Setting up scenarios for testing

Setting up 'master' testing cases once has also streamlined the testing process for the manager because the scenarios do not need to be specified to the testers and only need to be specified once.

5.3.3 Managing problems

Storing all the testing information in one database is also beneficial to the manager partly, again, because of the efficiency of just needing to consult one list, but mainly because problems can be prioritized, sorted, and allocated to programmers to fix within the database.

5.3.4 Information about the testing process

Reports generated from the CTT allow the manager to access information on progress at any point throughout the testing process.

6. Conclusion

In summary, the CAI testing tool has made the testing process more efficient – saving time and thus money and improved the quality of CAI questionnaires.

The tool has fulfilled all the objectives set out to justify its development. Using the CTT ensures that all projects are following a standardized approach to testing and teams are not wasting time developing their own databases in which to collate comments. The tool has saved time for all those involved in the testing process. The CTT also stores and allows access to documentation about the entire testing process, providing a source of data about one project and the ability to run reports easily. The tool also acts as a repository for information about future enhancements and could provide information that could be used to influence development of future projects or the questionnaire development process.

From our experience CTT has facilitated a smoother (and more harmonious) testing process with all those involved working in the same environment accurate and appropriate information can be shared easily among the programmer, tester, and manager.

We have found the CAI Testing Tool to be a useful tool that reduces the burden of testing for all involved in the process. Although the quality of a questionnaire can only be as good as the time spent testing it--this tool allows the tester to concentrate their efforts on releasing an instrument with minimal errors.

References

Dascola, M., Pattullo, G. and Smith, J. (2007): CTT: CAI Testing Tool Proceedings of the 11th International Blaise Users Conference, 2007, Annapolis.

Pattullo, G., Stark, R. and Batra, P. (2010) Testing Tool Manual Survey Research Center, University of Michigan.

Automated Regression Testing of BLAISE INTERNET: A Case Study

Karen Brenner and Sudha Manoj, Westat

A commercial off the shelf (COTS) testing product, TestPartner, is being used to automate test cases for Blaise Internet projects. This paper will review the pros and cons of automated testing and present considerations in making a decision about whether automation is the best option for a particular project. We will look at how we implemented automated testing and discuss issues that Blaise Internet presented and how they were resolved.

Automated Testing Lifecycle

This paper focuses on Automated Testing for comprehensive regression testing changes in a Blaise Internet instrument. Like the overall software development lifecycle (SDLC), test automation has its own lifecycle consisting of the following six steps:

1. Level of Effort Analysis: The decision whether to automate testing is based on analyzing the field length of the project, system stability, available resources, test environment, tool compatibility, and other factors. A number of significant benefits and false expectations of automated testing are discussed later in this paper.

2. Test Tool Acquisition: A test tool can be developed or purchased after comparing and evaluating tools on the market. TestPartner (by MicroFocus) is the standard automation tool currently used by the Westat Testing Unit; it has been successfully implemented on Westat projects.

3. Automated Testing Introduction Process: An overall test process and strategy needs to be in place before introducing automated testing for any new project. For example, documented system changes must be passed to testing to update the version of suites of scripts and expected results.

4. Test Planning, Design, and Development: This phase includes setting up a preliminary schedule and test environment guidelines, and developing the automated scripts. The Testing Team leads this phase of the work with staff equipped with a second desktop PC for developing and tuning the automated scripts. This phase includes close collaboration with the developer and/or tool tech support to help manage how the tool integrates with the system.

5. Execution of Tests: The test environment needs to be in place as planned. The automated scripts must have dedicated PCs on which to run. Dedicated PCs for automated testing are totally independent of development PCs, so that they can provide "unpolluted" platforms on which to test. Thus, they should be as close to the end user configuration as possible. It may be desirable for testing platforms to have increased processing speed and memory. This consideration has to be weighed against testing on a system that is identical to the end user system. The tester will execute automated scripts and provide test results for evaluation.

6. Test Program Review and Assessment: Test program review and assessment activities need to be conducted throughout the testing lifecycle, to allow for maintenance of scripts and continuous process improvement activities.

Significant Benefits of Test Automation

Once it has been determined that automation is appropriate for a project, the team can look forward to the following benefits:

1. Increased depth and breadth of regression testing.
2. Elimination of long, repeatable manual test cases. (Although automation doesn't eliminate manual testing, it does replace lengthy repeatable test cases so that testers can focus on particular issues.)
3. Reduction of the schedule.
4. Unattended testing. Automated tests can run unattended and overnight.
5. Improved quality of the test effort.
6. Improved assessment of system performance. Besides allowing for greater consistency and test coverage, automation can be used to ensure that the system's performance requirements meet or exceed user expectations.

False Expectations for Automated Testing

After considering the automated testing life cycle, the benefits of automation must be weighed against false expectations:

Automation can be implemented at any time

Automated testing requires a long-term project with an instrument that is relatively stable, that needs to have run a specified set of regression tests on a regular basis that have predictable results. If the project tries to implement automation prematurely, while a system is still being developed, it will increase the maintenance required to keep suites of scripts running that can provide useful feedback about software issues.

Automation can replace manual testing

Automated test tools should be viewed as *enhancements to manual testing*; they will not develop a test plan, design and create test cases and procedures, and execute the tests. The test team will never achieve 100% test automation of an application, because it is not possible to test all combinations and permutations of all inputs. An experienced tester will certainly need to execute tests during exploratory testing that were not predicted when writing test cases.

Automation is easy

Vendors sell an automated tool by exaggerating its ease of use, usually referring to it as a "record/playback" tool. Automation is actually more complicated than that, because recorded test scripts must be enhanced manually with code to make the scripts robust, reusable, and maintainable. To be able to modify the scripts, the tester must be trained and become an expert on the tool's built-in scripting language.

One tool does it all

Currently, one single test tool will not fulfill all testing requirements for most projects. Different tools are used for regression, file comparison, load, and other aspects of testing.

Immediate test effort reduction

In addition to the learning curve associated with applying automated testing to a project, test automation requires that the team pay careful attention to automated test procedure design and development. The automated test effort can be viewed as its own software development life cycle, complete with the planning and coordination issues that come along with a development effort. Time is invested up front to organize, plan, script, and debug.

Selecting an Automated Tool

There are many tools in the market which support automation. TestPartner is the testing tool we selected to automate our regression testing process at Westat. Like most automation test tools, you can create scripts automatically using TestPartner's record facility. When recording your actions, the responses of the application you work with are translated into TestPartner scripts. TestPartner can record just keystrokes, mouse moves and clicks. It works best at an object level –- identifying objects by name. Your actions are translated into simple commands. TestPartner lets you quickly record and execute test scripts. The tester can then modify test scripts to include hand-coded programming that cannot be recorded, to enhance the scripts and make them easier to maintain. All automated test tools will require this hand-coding to make scripts robust, even though tools are generally marketed as "record and playback."

In a basic out-of-the-box Blaise for Windows script, TestPartner is not able to uniquely identify the objects on the screen, since Blaise for Windows does not supply a unique attribute to the windows control. One option we've implemented with a Blaise for Windows application is to build a web front end for a Blaise survey, that uses the Blaise database, and then easily build automated test scripts using the HTML web interface.

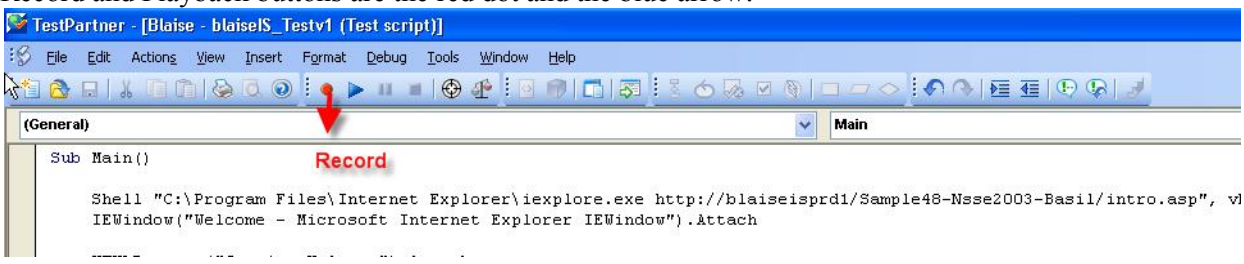
TestPartner works well with Blaise Internet and raises very few issues.

Example of how the tool works

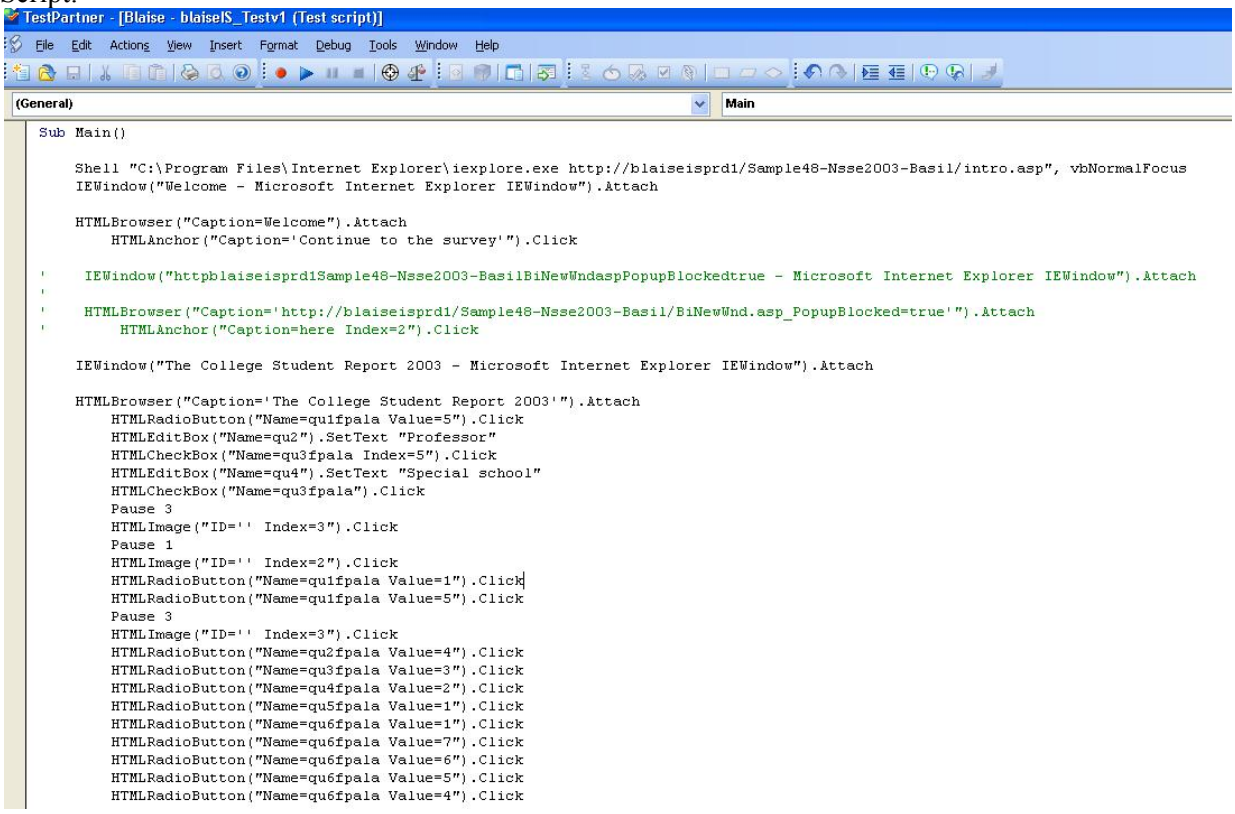
For demonstration purposes, we have version 1 of an instrument which is relatively small and has already been determined to be a candidate for automation. First, we need to record the script by going through the instrument manually. Each object in the instrument has an object name with unique properties. The object name is displayed in the script when the tool generates the code. The script generating options we use record the tester's actions such as keystrokes and mouse clicks as they are applied to the objects on the screen. These options make the script more understandable when read and allow us to more easily maintain it. Once the script is recorded the first time, we can play it back. After the script runs, the results will be displayed in the results window. If the script runs through correctly, the results show as passed. If the instrument has any changes or does not run, the script fails and the results are displayed as failed.

Here is an example of a script running on version 1 of the instrument.

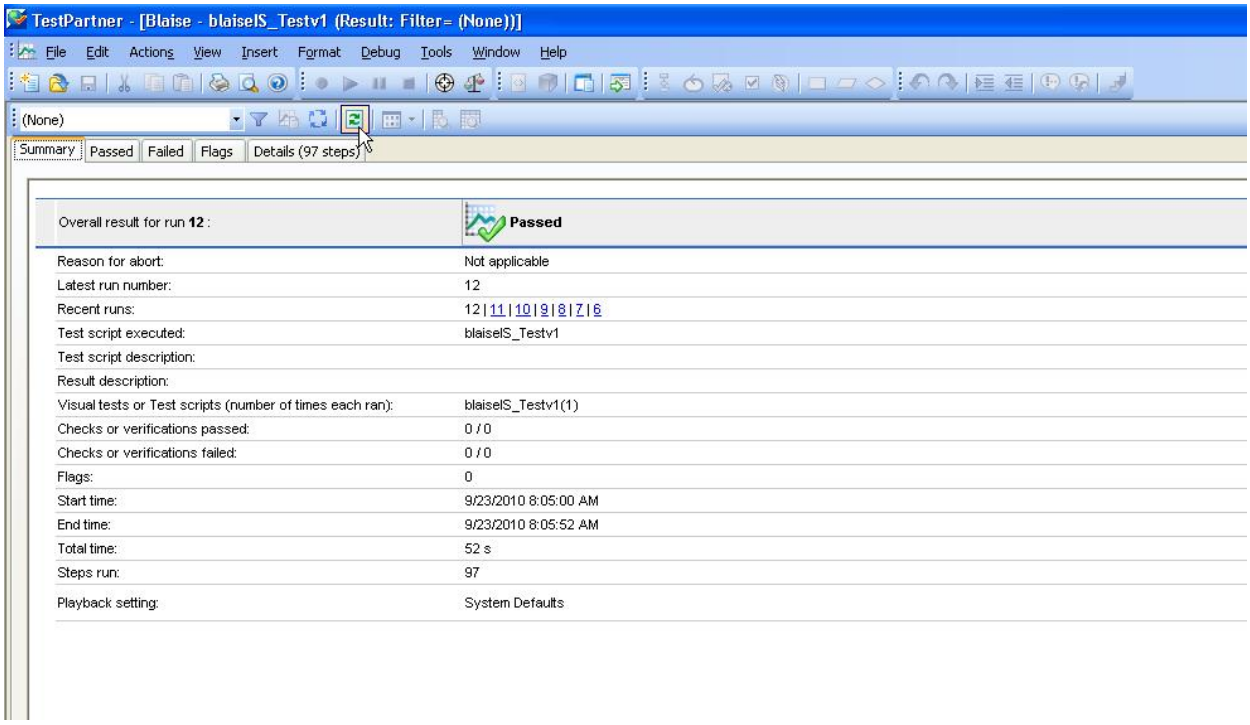
Record and Playback buttons are the red dot and the blue arrow:



Script:



Results Window:



If a script stops during playback, we know that there is an issue. When it is determined to be a defect, we enter the issue in our standard bug tracker. If it is not a defect, for example an intentional change to the instrument, we have to modify the script to make it run. This requires maintenance of the scripts as the versions of the instrument keep changing. It is not very difficult, but scripts have to be updated on an ongoing basis. One can edit the code manually to comply with the screen or re-record portions of scripts to incorporate the new information and allow the script to continue running.

First, it is important to note that automation tools have a variety of checks that can be used for more test coverage. Without checks, you're only testing navigation. Adding checks lets you confirm that the application is validating input, performing calculations correctly, saving data reliably, and reporting accurately, by comparing actual responses to expected responses and reporting any discrepancies.

Here are some of the checks that can be used in TestPartner.

Bitmap Checks: These checks compare a bitmap in the target application with one that was previously defined. Bitmap checks are used to verify the appearance of toolbars, the desktop, and other windows that contain non-textual information.

Content Checks: A content check enables you to verify the contents of controls that the tool supports. Currently, tables and list controls in a Windows-based or Web-based application are supported. List controls include list boxes, drop-down lists, combo boxes, and various "tree" type controls, such as those used in Windows File Manager and Windows Explorer.

Field Checks: Field checks enable you to conduct specific types of text comparisons, including date and time, of individual fields you select in a window or area.

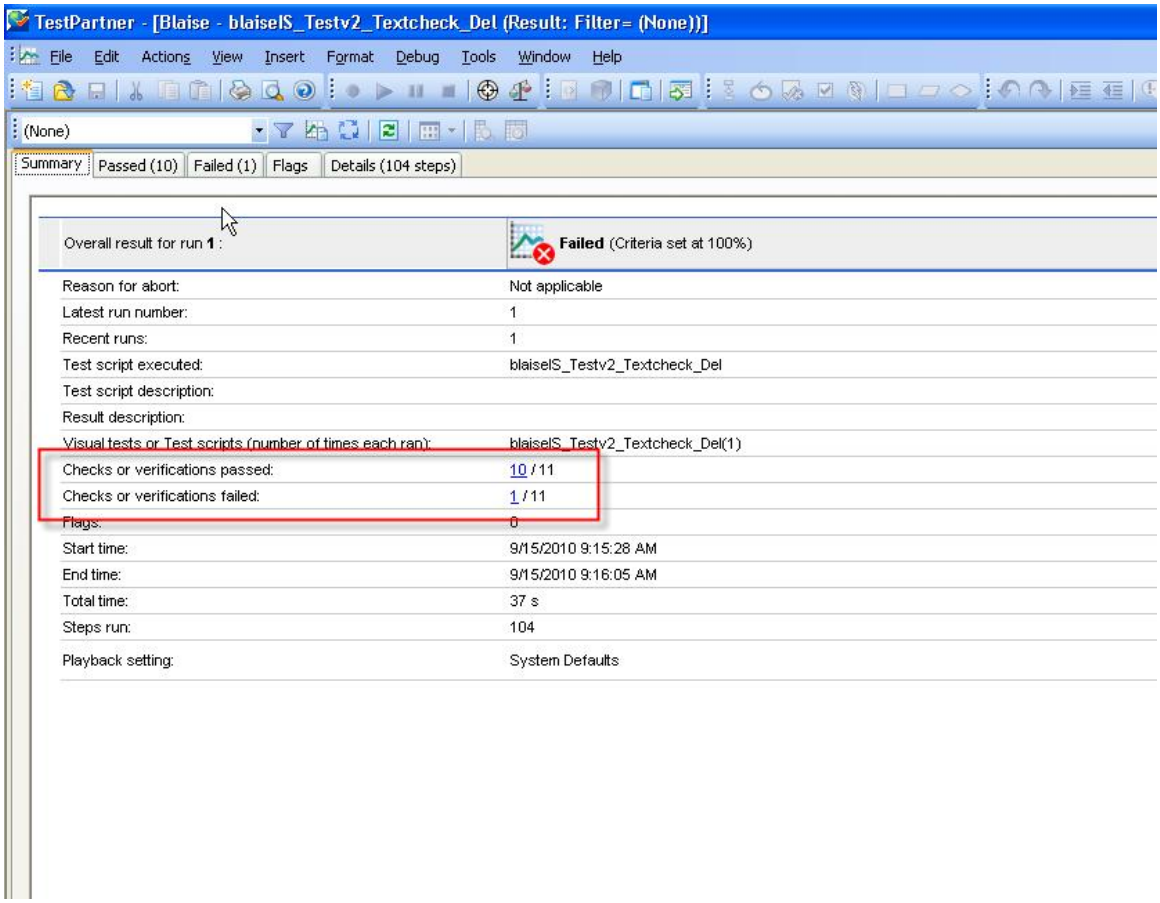
Text Checks: These checks can be used to verify the text on a screen. You can check an entire window and mask the area(s) that contain variable data, such as the current date or anything else that you want to ignore during the check. One can add a check by simply selecting the area of the window that contains the text to capture.

In the following examples, we will be examining an instrument where a field has been added to the page and a response list has been modified. We want to look at how to set up the script to spot various types of changes and how to maintain the scripts.

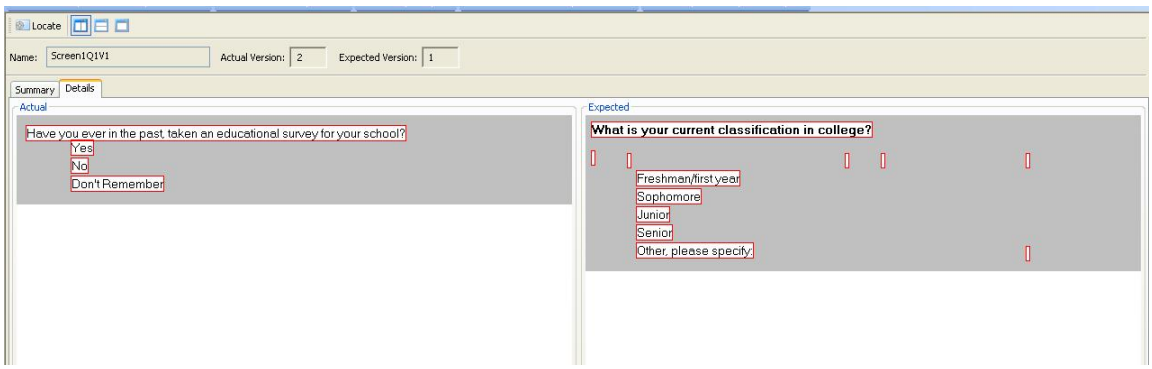
Text Check Example

In this example, version 1 of the instrument asks a Yes or No question on screen1, while version 2 of the instrument lists classifications in college. We captured the text and ran the same check on version 2. The script is expecting to find certain text in a specific location, and notices that there is a discrepancy. Since the text does not match, the check fails and is displayed on the results screen.

Failure Screen:



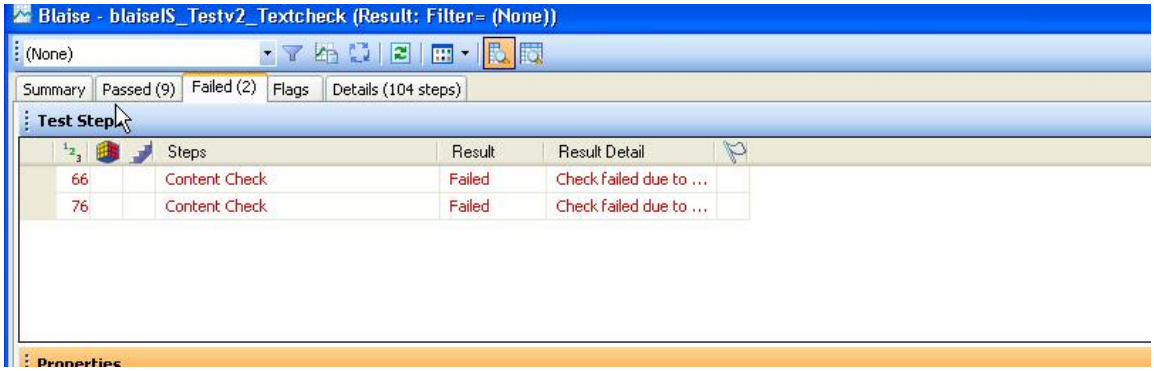
When you click on the failed check, it takes you to the details page, and the differences of the text are displayed as shown in the screenshot below.



Content Check Example

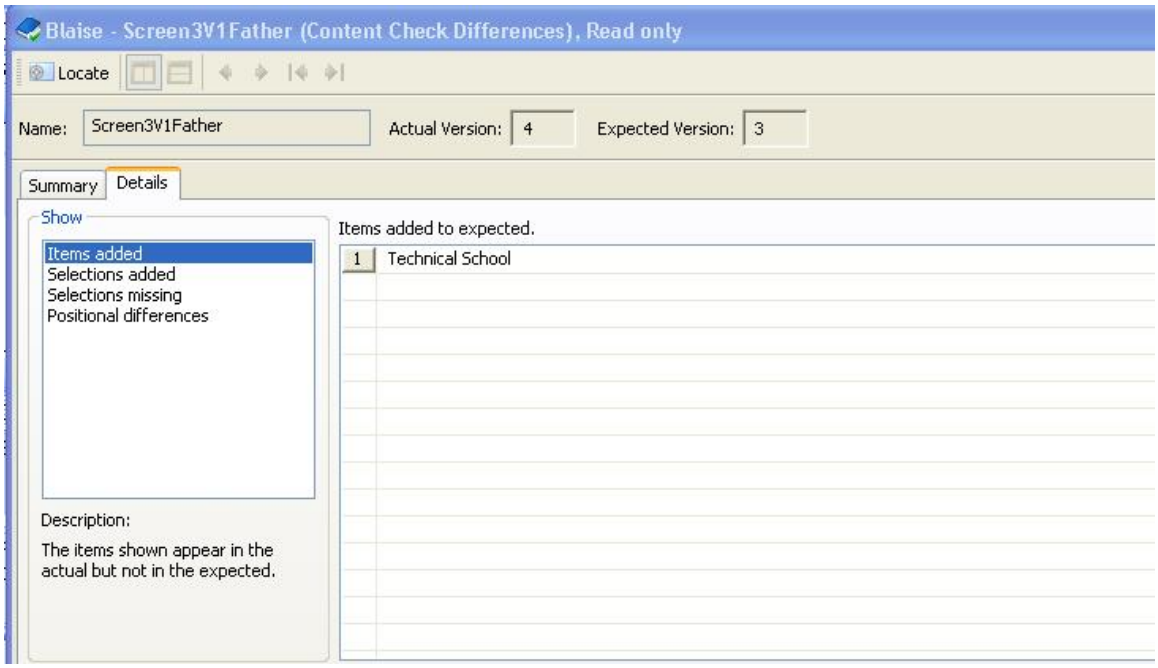
Content checks can be used to check the text of the items in a list or the number of items in a list, and if a list item is selected, the content check can be used to check the number of columns and rows in the table. One can add a check by simply selecting the field with the dropdown list to capture the items in the list.

In this example, version 1 of the instrument has eight items in the dropdown list on screen3, while version 2 of the instrument has been modified to have nine items. We created a content check in version 1 and ran the same check with version 2. Since the content of the items in the dropdown does not match, the check fails and it is displayed on the results screen.

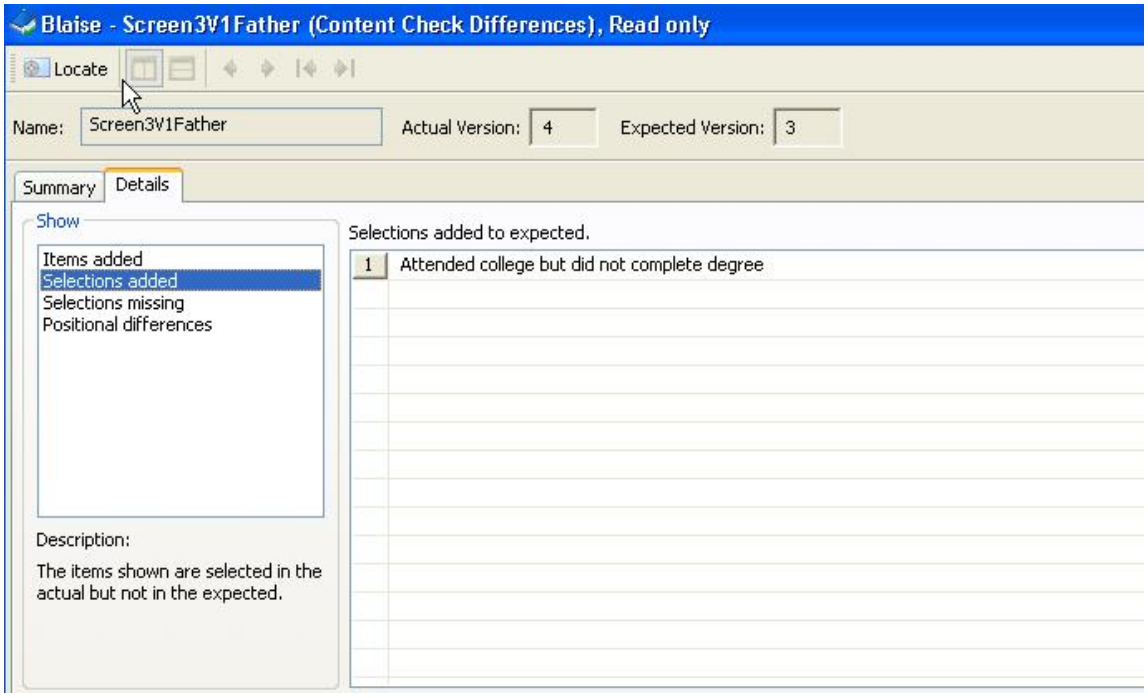


When you click on the failed check, it takes you to the details page and the differences in the content are displayed as shown in the four screenshots below. In this case, it shows any items added, selections added, selections missing, and any positional differences.

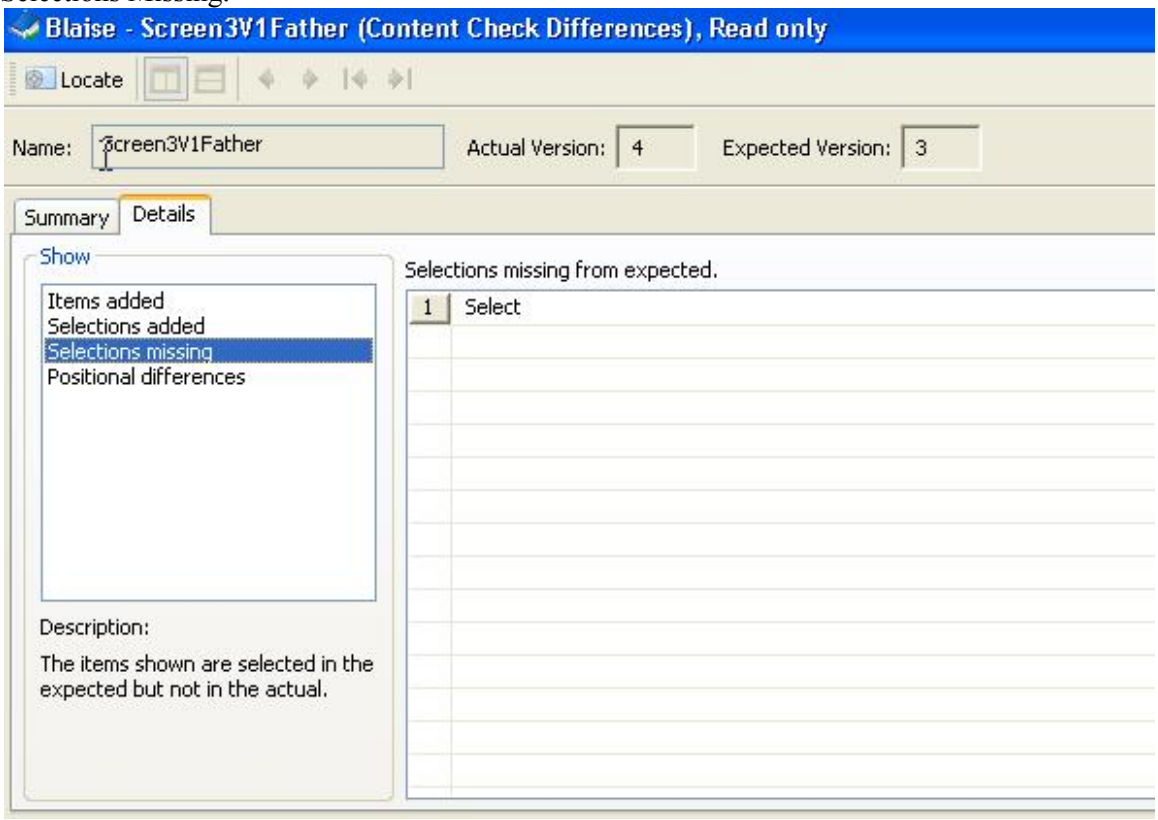
Items added:



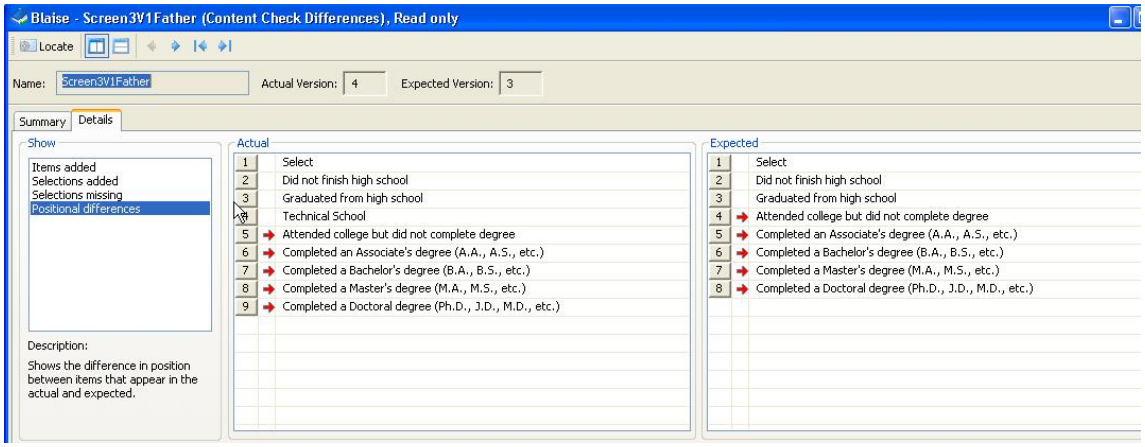
Selections added:



Selections Missing:



Positional differences:

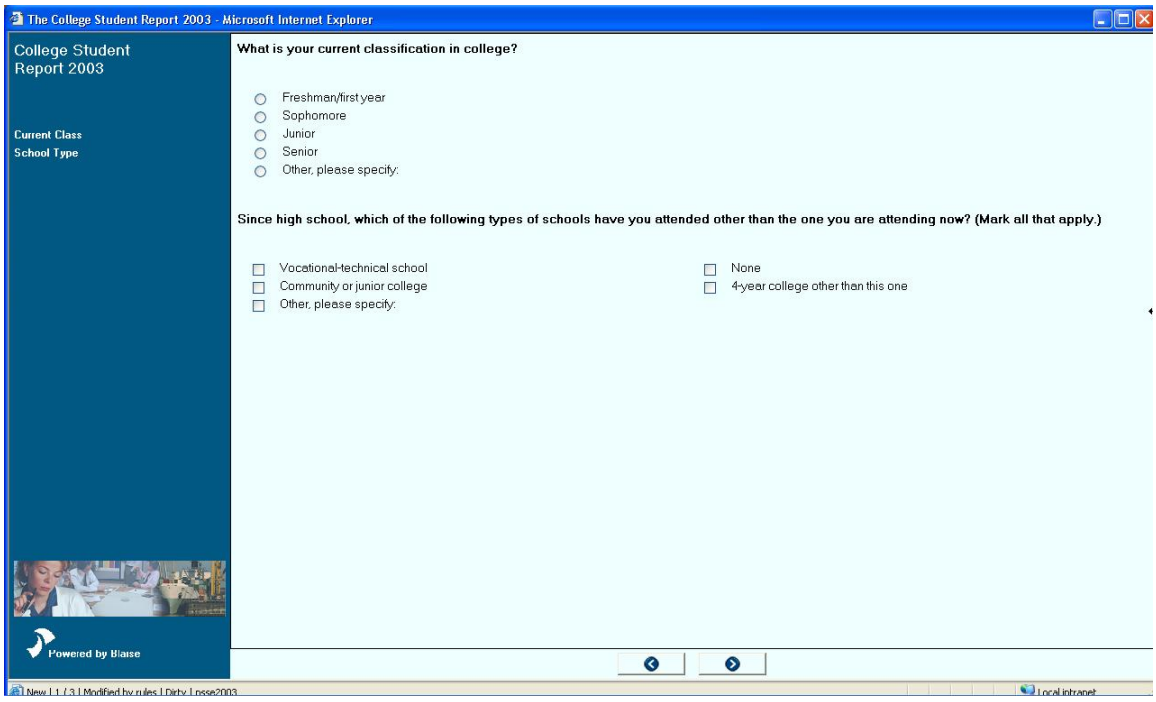


A script without checks would still run without errors as long as the navigation remained the same as originally scripted.

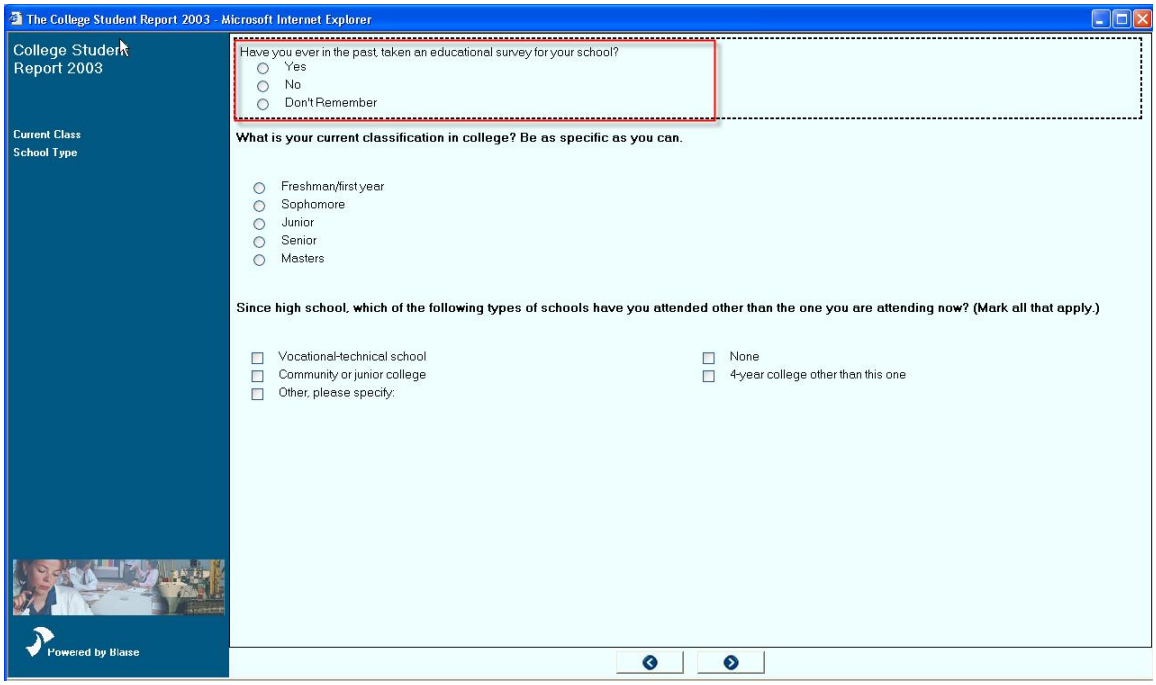
Example of an Issue with Script Maintenance – New Objects

The one area where we had additional maintenance issues using our tool with Blaise Internet was when there was a question (object) added to the instrument.

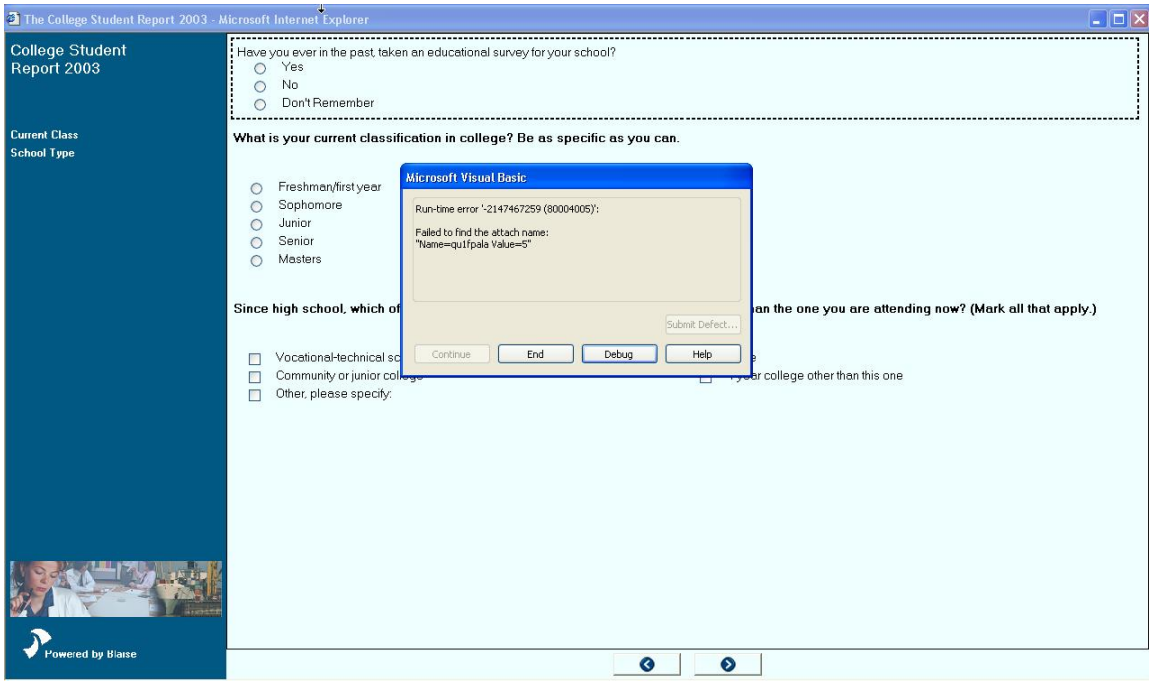
On this screen which is version 1, there are two questions.



The new version 2 has three questions. The first question seen in the red box has been added.



When we run the script during regression testing, the script stops when it gets to the new object. The script is expecting to be located on the classification field, but instead the focus is on the newly added survey field. Once it stops a debug screen appears as shown below.



When we click on Debug, it opens up the script and highlights the line where the problem exists, as shown in the screenshot below:

```
IEWindow("Welcome - Microsoft Internet Explorer IEWindow").Attach

HTMLBrowser("Caption=Welcome").Attach
    HTMLAnchor("Caption='Continue to the survey'").Click

'   IEWindow("httpblaiseisprd1Sample48-Nsse2003-BasilBiNewWndaspPopupBlockedtrue - Microsoft
'
'   HTMLBrowser("Caption='http://blaiseisprd1/Sample48-Nsse2003-Basil/BiNewWnd.asp_PopupBloc
'       HTMLAnchor("Caption=here Index=2").Click

IEWindow("The College Student Report 2003 - Microsoft Internet Explorer IEWindow").Attach

HTMLBrowser("Caption='The College Student Report 2003'").Attach
    HTMLRadioButton("Name=qu1fpala Value=5").Click
    HTMLEditBox("Name=qu2").SetText "Professor"
    HTMLCheckBox("Name=qu3fpala Index=5").Click
    HTMLEditBox("Name=qu4").SetText "Special school"
    HTMLCheckBox("Name=qu3fpala").Click
    Pause 3
    HTMLImage("ID='' Index=3").Click
```

The result is that when an object is changed (added or removed) on a Blaise Internet page, the properties of all pre-existing objects on that page also change. One can edit or re-record the script to add the missing question or edit the code manually to correspond with the new screen updates and continue the script.

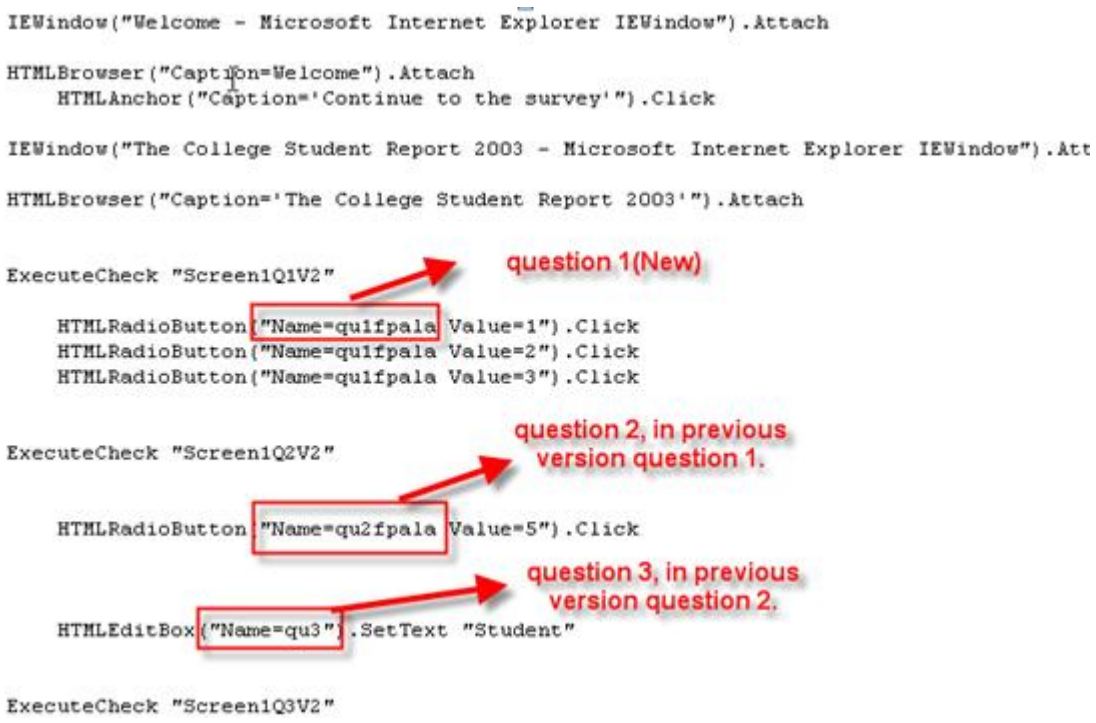
In our example, the script has the object name for question 1 as “Name=qu1fpala”, and the object name for question 2 as “Name=qu2” as seen in the screenshot below.

```
ExecuteCheck "Screen1Q1V1"


HTMLBrowser("Caption='The College Student Report 2003'").Attach
    HTMLRadioButton("Name=qu1fpala Value=5").Click

    HTMLEditBox("Name=qu2").SetText "Professor"

ExecuteCheck "Scr1Q2V1"

    HTMLCheckBox("Name=qu3fpala Index=5").Click
    HTMLEditBox("Name=qu4").SetText "Special school"
    HTMLCheckBox("Name=qu3fpala").Click
    Pause 3
    HTMLImage("ID='' Index=3").Click
    Pause 1
    HTMLImage("ID='' Index=2").Click
    HTMLRadioButton("Name=qu1fpala Value=1").Click
```

question 1

question 2

After the new question is added, the properties of all pre-existing objects on that page also change. Now you will notice that the object name for question 2 is now “Name=qu2fpala” instead of “Name=qu1fpala”.



This indicates that the object properties change on that particular page when objects are added or removed. Therefore, the script has to be re-recorded for that section or the user can manually change the code. The maintenance issue is made easier because in general there are not many objects on a Blaise Internet page.

Conclusion

In summary, there are many automated testing tools. This paper addresses our experience using one COTS product. We were able to use an automated testing tool to successfully perform regression tests on a Blaise Internet instrument. The regression testing is ongoing and will be considered for other complex Blaise instruments.

Moving to a Meta Data Driven World

Chris Seymour, Statistics New Zealand

1 Introduction

The recent international financial crisis has left the economies of many countries reeling as stock markets have fallen and financial institutions failed. Governments have been forced to intervene with rescue packages to ensure their countries' financial systems do not collapse.

With the extent and duration of the global financial crisis not yet known governments are now demanding more information, sooner, but with an expectation that additional funding will not be available to help it happen.

This message comes within an environment where the governments, such as New Zealand's have clearly articulated that public sector spending will be cut over the coming years.

> "Most Government agencies will receive no Budget increases over the next 3 or 4 years, as the Government moves to get back into surplus as soon as possible"

This '*tightening of the belt*' is requiring government agencies to rethink the programmes of work that provide value for money.

2 Changing the 'status quo'

Within National Statistical Office's (NSO's) the global financial crisis and reduction in state sector funding highlights a need to challenge the 'status quo'. This in turn is leading to significant changes in areas that have previously been considered sacrosanct. For example:

> "…the U.K. is getting rid of the 200-year-old head count because it's expensive, incomplete and out of date before it's even published… In a country struggling with a $235 billion deficit, the census might be seen as a luxury"

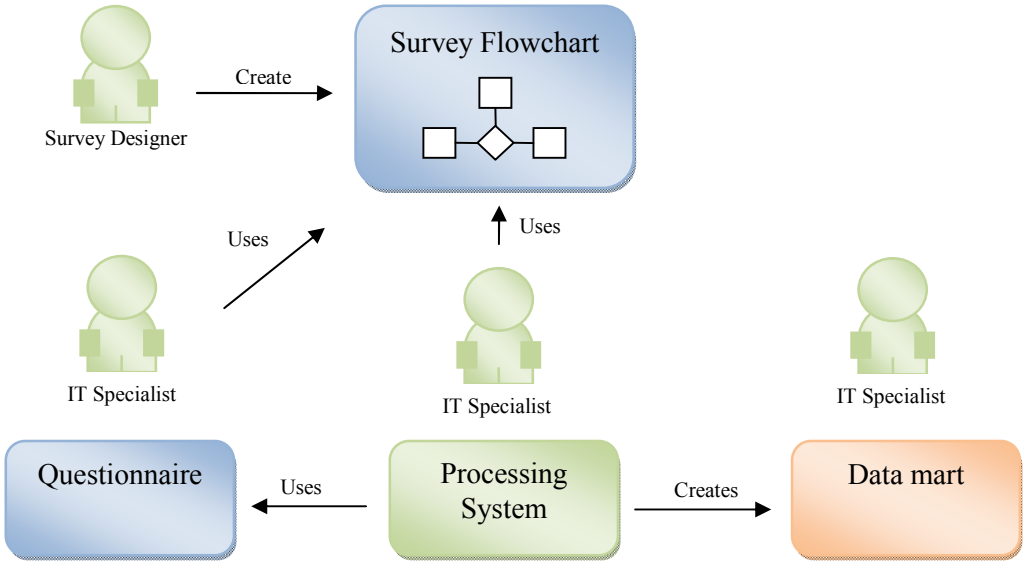
While some agencies are considering moves to cut programmes of work, economic analysts are predicting that there will be a surge in demand for more timely information. Agency independence and isolation has also meant that may NSO's operate in an environment which consists of numerous, often disconnected, technologies and systems which are difficult to bend to new methods of operation.

In order to stay relevant NSO's will need to be able to adapt and react faster to changing demands while operating at a lower total cost. To do so will require the use of new methods and technologies by which this can be achieved.

Within the data collection arena, which is an increasing expensive exercise for NSO's, cost and efficiency drivers are resulting in Statistics New Zealand reviewing all aspects of their collection activities.

> "Respondents to statistical surveys are increasingly difficult to contact, disengaged and reluctant. This raises collection costs and impacts on data quality."

The current state, which results in information being duplicated, in multiple phases of the Business Process Model (BPM) and within multiple technologies, each of which requires involvement from IT specialists, is now harder to justify and rationalize.



As such Statistics New Zealand is increasingly interested in the separation of content from technology.

3 Meta data driven systems

Advancements in technologies and standards are changing the way metadata is viewed by NSO's. A definition of metadata states:

> "Metadata is loosely defined as data about data"

No longer is metadata the information that must be collected to assist in understanding disseminated data; which has resulted in varying degrees of usefulness and completeness. Now metadata is seen as a corporate asset which can be used to drive the business process and has sparked a number of activities which are transforming the way data is being managed through the BPM.

Over the last two years Statistics New Zealand has focused on the development of metadata driven platforms within the 'process' space of the BPM. This approach has allowed us to construct a platform that can store any shape of data, while at the same time allow for change and growth in the information without the reliance on IT specialist.

The following sub sections present the core concepts as they relate to data, data modelling and statistical processing design (data transformations).

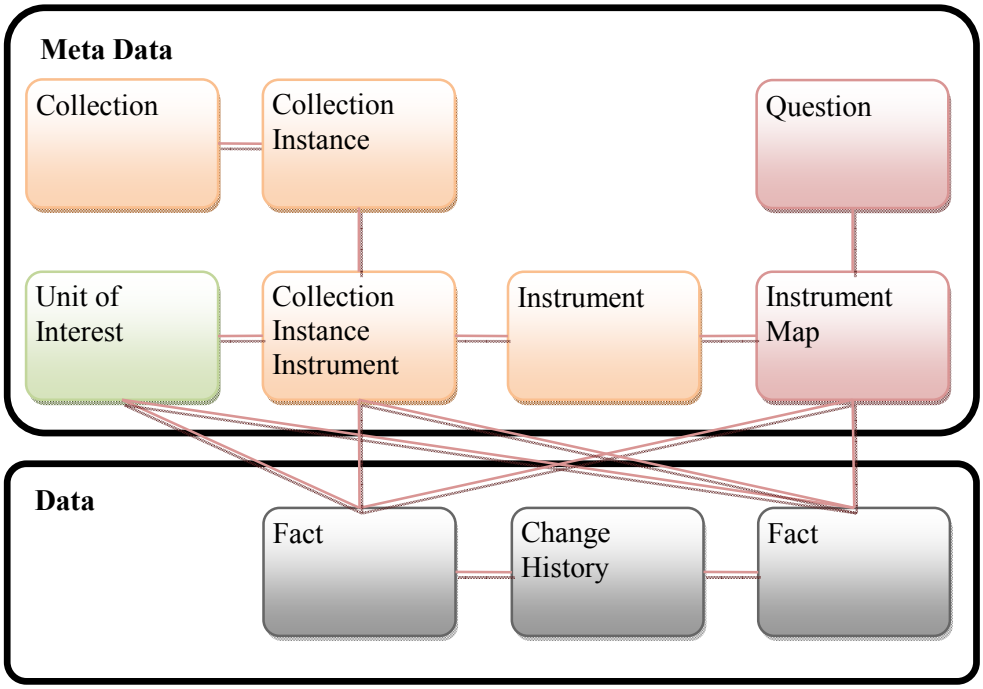
3.1 Concept Library

A concept library provides a central repository of information about the questions and variables that are collected through survey instruments and administrative datasets. It provides not only a mechanism to introduce standardisation of questions and variables, but also defines a minimum level of information capture.

In order to realise the real benefits of the metadata that could be captured new approaches were investigate for the storage of this information type; from which we turned to dimension modelling.

3.1.1 Dimensional Modelling

Dimensional modelling is a technique where the data structures are presented in a standard intuitive framework consisting of a central fact table surrounded by a number of dimension tables (i.e. a traditional star-schema). The fact table stores the measures which is the data that is of primary interest; with the dimension tables containing the descriptive information that give context to the facts.



3.2 Process Configuration

The configuration store provides a hierarchical repository for process configuration information. This information type is generally unstructured in nature and has been stored as xml documents which are directly interpreted by the processes they configure. Examples of configurable processes include standard statistical tasks such as: data editing, imputation and transformation, reweighting, tabulation, variance calculation, data storage and extraction, etc.

3.3 Routing Rules

The success of this work has prompted thinking within the collection space that over the next few years aims to deliver a flexible and configurable system that can meet our collection needs both now and into the future.

Of particular importance for a collection system will be the capturing of the questionnaire routing information - this type of data is heavily used within the processing phase to determine the eligibility status of the response for a respondent. Current tools force this information to be directly embedded in the instrument with no way of accessing or interpreting it. This results in the duplication of this information, often in multiple technologies, which becomes disconnected and prone to error through re-interpretation.

This has lead to the investigation of rule engine technologies with the aim of centralising the creation and storage of this information.

The combination of metadata and metadata driven tooling will enable the delivery of a collection platform for Statistics New Zealand which provides it users with the flexibility and control they require, while at

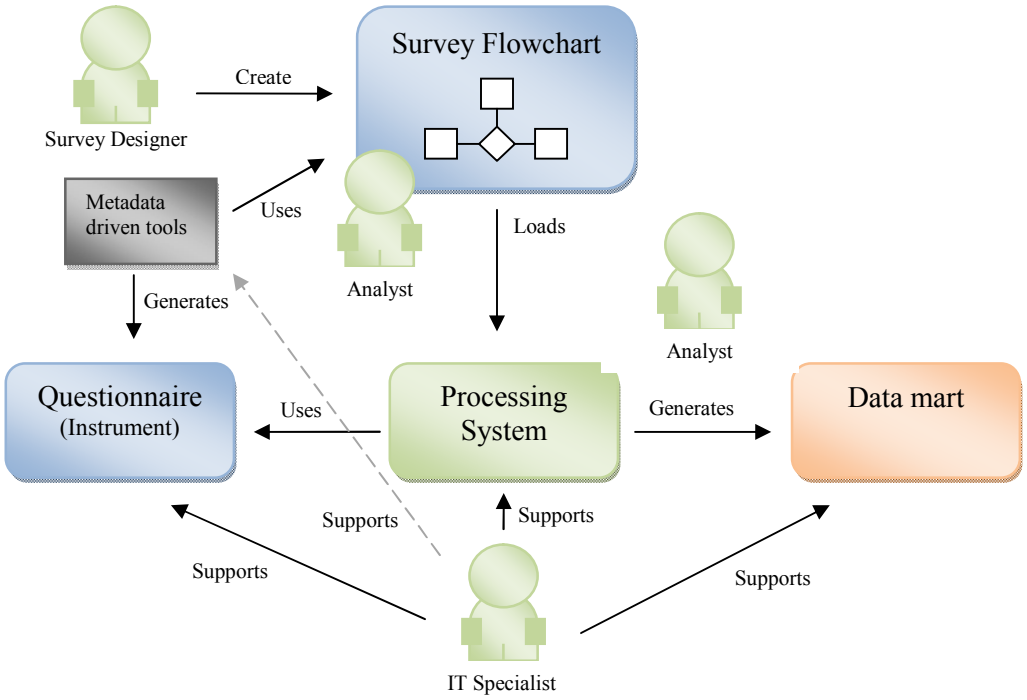
the same time delivering a lower total cost of ownership through reduced IT involvement over any collection's lifecycle.

4 Bringing systems together

The advancement in infrastructure elements such as metadata driven question libraries, configuration driven tooling and graphical routing technologies, coupled with the increasing development and adoption of data related industry standards, provide us with the confidence that the physical instrument can be separated from the information used to create it and the information it collects.

As this generation of tools begin to emerge, tools like Colectica, although immature and not yet capable of supporting the corporate arena, offer promising sign that this vision can be realised.

The vision:



4.1 Standards

One of the key factors behind these advancements is the emergence of standards aimed at the statistical world. With adoption on the increase, statistical organisations are positioning themselves to take advantage of the transparency and ease that standard bring.

Of the standard gaining favour, the Statistical Data and Metadata eXchange (SDMX) standard for aggregated data and Data Documentation Initiative (DDI) for unit record data are the lead candidates.

4.1.1 SDMX

> "SDMX is an initiative to foster standards for the exchange … of aggregated statistical data and the metadata…"

4.1.2 DDI

> "The Data Documentation Initiative (DDI) is an effort to create an international standard for describing social science data…designed to document data across the research data life cycle, from questionnaire design to data publication…"

Although SDMX is recognised as the leading candidate for the interchange of aggregated data between statistical organisations, it has been the adoption of DDI within the collection space that is providing excitement in achieving our collection vision.

4.2 International collaboration

Recently a number of NSO's have meet to investigate the setting up of an international collaboration group which will focus on the needs of statistical offices. The will enable NSO's to combine resources (people, budget and time) to deliver solutions (or part solutions) that can be shared by multiple agencies, which independently a single agency would not have been able to create.

Although this programme is in its inception it shows promise in delivering solutions tailored to the statistical world, which to date not, have not been available through traditional IT vendors.

Standards, such as SDMX and DDI, will play a key role in the success of the work undertaken by those agencies involved in the collaboration exercise.

4.3 Changing the cost model

The use of standards brings with it a number of other indirect benefits for statistical organisations; it enables new entrants into the arena.

As vendors and open source communities begin to produce offering in the statistical space, the cost models for agencies can begin to move from projects (capital expenditure) towards that of licensing products (operational expenditure). This shift provides a huge advantages to organisations and governments leaders as the total cost of operation can be determined up-front, which in turn enables better planning and forecasting of financial expenditure.

5 Direction

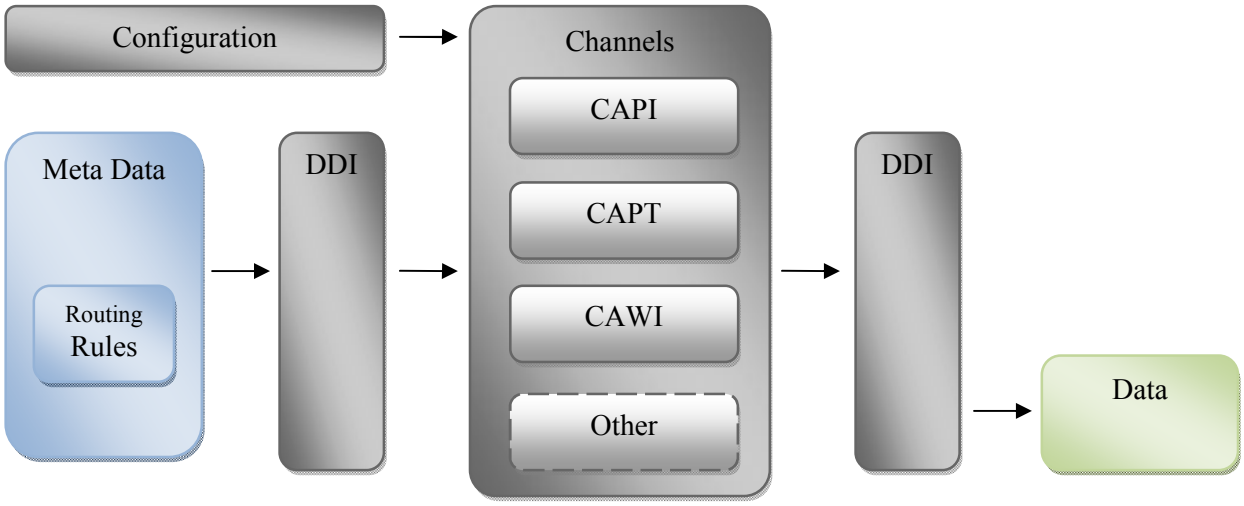
With reliance upon an increasingly time-poor, mobile and technically competent population, collection systems will need to be flexible.

The vision enabled through the use of metadata and standards is clear.

- Survey design is given back to survey designers
- Questionnaire development is standardised through the use of question libraries
- Instruments are generated, not hand crafted.
- Duplication is eliminated with the use if IT systems that directly consume the survey design
- Flexibility is enabled through choice of questionnaire delivery modes.

DDI is delivering a standard by which technology can, and is, being developed that will enable a true multi-model environment which caters for current technologies, and provides the potential to enable new modes to be seamlessly plugged-in with very little integration, cost and effort.

Metadata and configuration will be the information required to feed into these types of system that provides the flexibility for users to tailor the collection environment to their needs from which to create and manage the instruments, using the mode which they require, for a given survey.



6 What about Blaise?

So where and how does Blaise fit into our collection vision for the future?

As stated above, Statistics New Zealand's current questionnaire development processes and systems are unsustainable in the current climate, and the answer to this question will largely depends on the functionality within the upcoming BlaiseNG release.

Three possible scenarios exist:

6.1 Blaise meets DDI

The best possible scenario is that Blaise introduces functionality that enables the tool to be DDI compliant. This would provide the best possible combination of flexibility and standardisation within a single tool from which to create survey instruments.

6.2 Wrap it up

This would require the building of software that could externalise the blaise data model using a standard's based model such as DDI. This would allow the instrument 'content' to be developed independently, but still use the Blaise DEP to capture the response from a respondent, limiting the impact on existing IT investments. As a number of NSO's use Blaise it is envisaged that this work could be done through the NSO collaboration group to be offered back to the international community.

However, while this scenario offers the potential of flexibility and standardisation when creating a survey instrument, the architected solution could be brittle as it relies on an as yet unknown data model, outside the control of the community, and is likely to change over time.

6.3 Move on

The last scenario limits Blaise to a viewing 'engine' only as new tools are transitioned into the collection environment. The survey instrument would be produced as a standards based document which is then translated into the language of the tool required to display it.

Initially Blaise, with the instrument generated through software like 'Colectica', would still be used to do this. As more tools emerge that can consume a standards compliant document and collection support systems were redeveloped Blaise would begin to be phased out.

Although the path ahead for Blaise at Statistics New Zealand is still not set, it is clear that tools which are metadata driven and standards compliant will play a significant part in the makeup of our collection system.

7 Summary

The global financial crisis has put added pressure onto governments and government agencies to reduce cost; in particular leading statistical offices to begin rethinking the methods, justifications and practices used to create statistics in an environment that is demanding more, sooner.

Within Statistics New Zealand metadata is playing an increasing part in the generation of systems that use rather than just collect this type of data and are proving to be more flexible and resilient to change.

The use of standards, such as DDI, are seen as a key enablers both in terms of aligning technology and allowing IT vendors into the market, which will result in new cost models that offer the potential to lower the overall cost of ICT for statistical agencies.

A DDI or standards aware Blaise has the potential to be at the forefront of survey development tools in an environment where significant change is underway.

8 References

Collections 2020 Paper, Andrew Hunter, 16/07/2010

(Hon) Bill English - Finance Minister, http://www.parliament.nz/en-NZ/PB/Business/QOA/4/b/9/49HansQ_20100422_00000002-2-Budget-2010-Value-for-Taxpayers-and-Better.htm, 22 April 2010

Adam White, http://www.time.com/time/world/article/0,8599,2005245,00.html#ixzz0wol9NEMQ, 21 July 2010

Keith Chung, Input Data Environment (IDE) Design Overview, November 2007

The DDI Alliance, http://www.ddialliance.org, August 2010

The SDMX Sponsor Committee, http://sdmx.org/?page_id=10, August 2010

Resolving Question Text Substitutions for Documentation Purposes Using the Blaise 4.8 API Component

Jason Ostergren, Helena Stolyarova, and Danilo Gutierrez, The University of Michigan

Overview

The Health and Retirement Study (HRS) is a national longitudinal survey on the health concerns and economics of aging and retirement. The HRS utilizes a CAI instrument for biennial interviews of one to three hours in length given to around twenty thousand participants. One of the prominent features of the HRS CAI instrument is that it makes uncommonly heavy use of text substitution in its question wording. In some HRS questions, the entire wording consists of text substitutions, and oftentimes multiple substitutions are used in succession and/or layered inside of others. While this situation provides for a more dynamic and streamlined interview experience, for the purposes of documentation it presents a steep challenge for anyone attempting to resolve and document the various permutations of question wording.

There are various reasons why HRS finds text substitutions so necessary. At a basic level, they allow HRS to provide exact wording to the interviewer rather than offering a parenthetical that forces the interviewer to choose the correct words on the fly (e.g. "What is your husband's name?" vs. "What is [your/his/her] [husband's/wife's/partner's] name?"). Some substitutions are very closely tied to particular bits of data like gender and tend to be fairly simple. Others reflect more complicated and dynamic data gathered from multiple variables or the flow of a particular interview. What is more, these various kinds of substitutions can be layered on top of each other to produce even more complicated structures.

HRS has tried a number of schemes to resolve these text substitutions in the past. These range from manually tracing the logic of the substitutions to parsing the Blaise instrument source code in various ways. These past attempts have each proved deficient in some way. While no scheme is ever likely to be perfect, by making use of improvements in the Blaise API, HRS has now written a program that can resolve these text substitutions accurately and efficiently. The key advance involves tracing variable assignments through parameters between blocks and procedures using the API.

Text Substitutions in the HRS

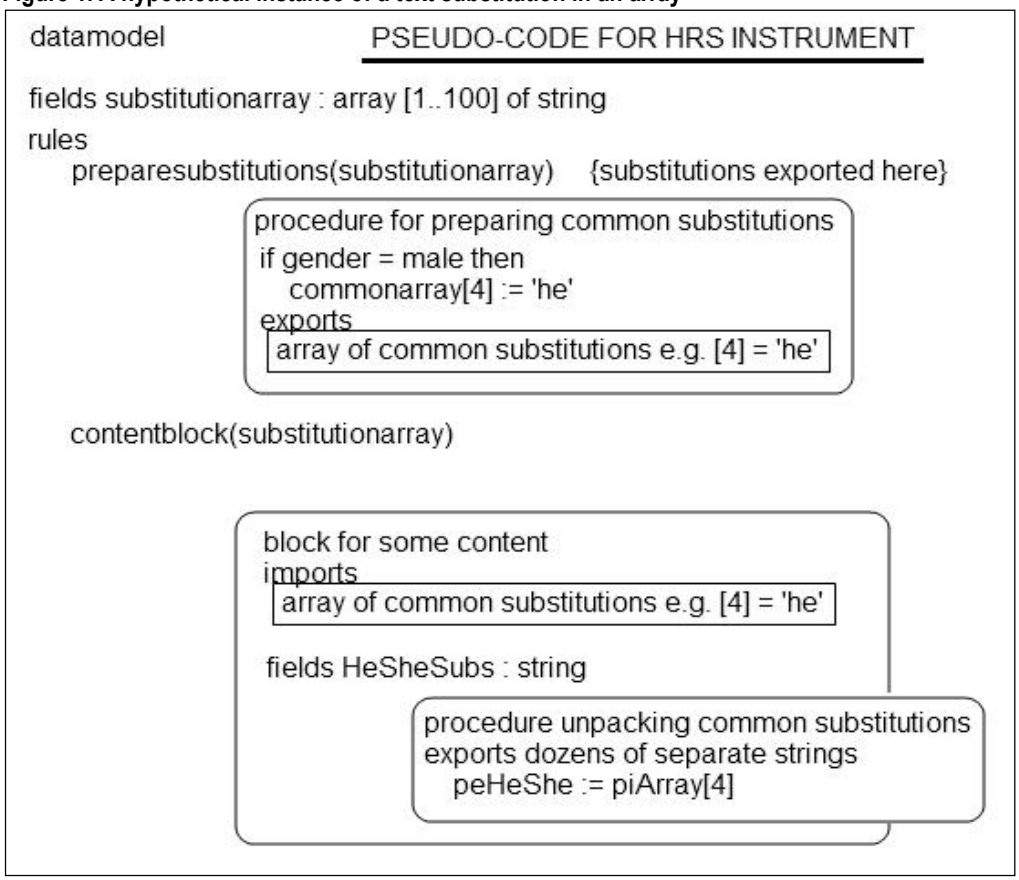
To see how this works, it is first necessary to understand how HRS handles text substitutions in the code of its instrument. Depending on how one counts, the present iteration of the HRS instrument contains between 6,000 and 15,000 text substitutions in its primary interview language. When Blaise was adopted as the platform for the HRS interview in 2002, an array of problems related to text substitutions presented themselves and the solutions chosen continue to drive the structure of our instrument. In particular, HRS chose to move all code for text substitutions into procedures and pass the necessary arguments and the text output as parameters. Procedures were chosen over blocks in part because they do not require maintenance of instance fields to call them and because HRS had no desire to store the internal variables used in generating the text substitutions in the database.

The problems which HRS confronted fell mostly into three categories which included the degree of clarity of the flow code, the ease of translation, and performance. The HRS instrument possesses a high degree of complexity in its flow logic and thus code readability has been a real problem in the struggle to debug and maintain the instrument. Pulling all of the text substitution code out of the flow rules helps to keep these rules readable to the programmer, decreasing the likelihood of flow bugs. Additionally, HRS has Spanish language translators who work in the code but are not as proficient with Blaise as the main programmers. Sectioning off all the text substitutions into procedures which are placed together near the

top of each include file has made translating this text somewhat easier and minimized the incidence of flow bugs being introduced by a translator.

Performance was also a serious problem in the early versions of the HRS Blaise instrument, in large part due to an excess of generated parameters, some of which were in effect global text substitutions – that is common substitutions referring to the respondent's spouse or partner, for example, which are used across different content areas. To improve performance in the handling of these common substitutions, HRS built a procedure which was called at the beginning of the instrument that packaged up the 40 or so strings into an array which was exported to a field at the datamodel level. This array was then passed into each content block (of which there are approximately 25) and another common procedure was used to unpack this array into a set of substitution strings for use in that block (see figure 1). This meant there would be no generated parameters from outside content blocks while also requiring only one field to be passed into each block, as opposed to up to 40 separate substitution strings.

Figure 1. A hypothetical instance of a text substitution in an array



Although all of this makes the instrument more reliable and efficient, the results of this scheme when it comes to handling metadata are mixed. In some ways, sectioning off these substitutions makes them easier to digest manually for the staff tasked with documenting them, but this is not at all an easy task given the size of the HRS. For automated documentation systems, this has proved a nightmare because it has simply been difficult to trace the content of a substitution among all the blocks, procedures and parameters though which it may have passed.

When it first adopted Blaise, HRS found itself almost immediately in need of ways to resolve text substitutions. Question wording with accurate substitution text was needed for IRB approval,

documentation, translation, quality assurance and other purposes. Manual attempts by programmers to document substitutions, such as placing the resolved text in the question text of the substitution variable proved to be impractical to maintain. It also proved too difficult for non-programmers assigned to document the instrument to determine the possible outcomes of the substitutions in the web of parameters through which they passed.

Resolving Text Substitutions: General Approach

Ideally, a program which resolves substitutions properly would operate as described in the following paragraphs. It would walk through the statements of the Blaise instrument in order, stopping at questions which are asked and which contain carets in the question text of the language being resolved. It would begin by correctly parsing the name of a substitution variable out of some question text, done by matching text following the carets which indicate substitutions. Some care needs to be taken here and also at later points to parse this correctly keeping in mind potential sources of error such as the fact that fully qualified variable names may be used and that characters like periods at the end of sentences may be adjacent to the variable name.

Once the variable containing the substitution is named, the correct instance of that variable must be identified since identically named variables can exist in different blocks. The program must begin searching for the variable in the immediate block. If no such variable is present, it must then begin to search upwards through the parent block and on to its parents as necessary. If the variable was a parameter, the program must identify the name of the variable passed in by the calling block and begin the process again based on that variable. If a fully qualified variable name is provided at any point that ends the search.

Once the actual substitution variable is identified, the program must find and evaluate any values assigned to that variable. Once processed, these become the resolved content of the text substitution – for example 'husband/wife/partner' – and can be plugged into the documentation of the question text at the location of the variable. In a simple case, there may be only a few strings such as 'husband' and 'wife' to locate. In more complicated cases, variables may be nested in the assignments which themselves have to be recursively traced and resolved and then incorporated into the larger resolution process of their parent substitutions. Additionally, it is necessary to filter out assignments for texts in languages other than the one presently being resolved (in HRS there are six "languages" – three each of English and Spanish for respondents, proxies, and proxies for deceased respondents– which each may have specialized substitutions).

The process of finding these assignments involves walking through all statements looking for direct assignments to that variable as well as assignments coming from export parameters of blocks or procedures. When parameter assignments occur, a recursive process of identifying assignments to the export parameter in question inside the other block or procedure begins. Assignments can also come from fully qualified references to the variable or from generated parameters. Finally, in some cases in HRS, assignments can come from parts of the instrument which are not even on the route at the point where the question with the substitution being resolved was asked (but which could nonetheless appear in the actual interview). Incidentally, HRS has made no attempt to account for this last eventuality, even though it occurs frequently in the first section of the HRS instrument due to the complexity of HRS and the way the rules engine works.

Beyond the mechanics of tracing the substitutions as described above, various additional obstacles exist which have to be handled. For example, some substitutions simply display a user input value without ever being assigned a value in the rules, which means there is nothing from the rules to use. However, if description texts exist in these situations, it makes sense to substitute them. Another problem is handling

array indexes which can make matching variable names difficult, for starters. It can be useful to try to filter the results by array index if a definite number is given, but it is often the case that a counter variable is provided, or worse that more than one array index is involved or that some expression such as 'blockA[i – 1].Q1' is used. HRS found that it was not worth tracing any indexes more complicated than a simple number. It also seems advisable to avoid tracing mostly irrelevant variables like counters when resolving text substitutions.

Implementation: Blaise API Use

HRS has been using the Blaise API to gather metadata since 2001, but only with more recent versions of the API has HRS been able to use it to write a program along the lines described above. The API provides a `RulesNavigator` object which can be used to walk through the entire statement hierarchy of the instrument. It also provides field metadata for statements that put a field (including blocks and procedures) on the route, which includes items such as question text that can be searched for the carets indicating substitutions. The functionality of this is well known and will not be described here, but the information that is gleaned from it is the key to this process. What follows is a description of the new kinds of data that must be gathered from the `RulesNavigator` in order to resolve text substitutions effectively.

Given the large number of places in the process where tracing parameters is critical, it is vitally important to glean information about parameters from the API. Three pieces of information are vital. These are the name of the parameter inside a block or procedure (that is, its name in the parameters section) and the corresponding name of the variable outside which is passed in as an argument to the block or procedure call. Finally, it is necessary to know the direction of the parameter (import, export or transit) – that is, whether the data passed by the parameter is flowing into or out of the block or procedure in question.

To obtain this information, it is necessary to look at the `StatementText` and the `Field` object for the statement calling the block or procedure. If there are parameters involved, the `StatementText` string will contain the text "{Parameterlist:}" followed by a comma delimited list of names of the parameters on the outside of the block or procedure call. By splitting this list by commas, an array can be obtained which will correspond to the order of parameters listed in the `Fields` collection *inside* the block or procedure. These `Fields` can be obtained from the method `get_DefinedFields()` on the `Field` object used above. Then it is necessary to loop through this set of fields and obtain each `LocalName`, which will provide, in order, the inside names that correspond to the variable names from the "Parameterlist." While looping through the set of `Fields`, it is vital to collect their `ParameterKind`, which will require casting them to `IBlaiseField2`. It may also be helpful to verify that they are parameters by checking that `FieldKind == BlFieldKind.blfkParameter`.

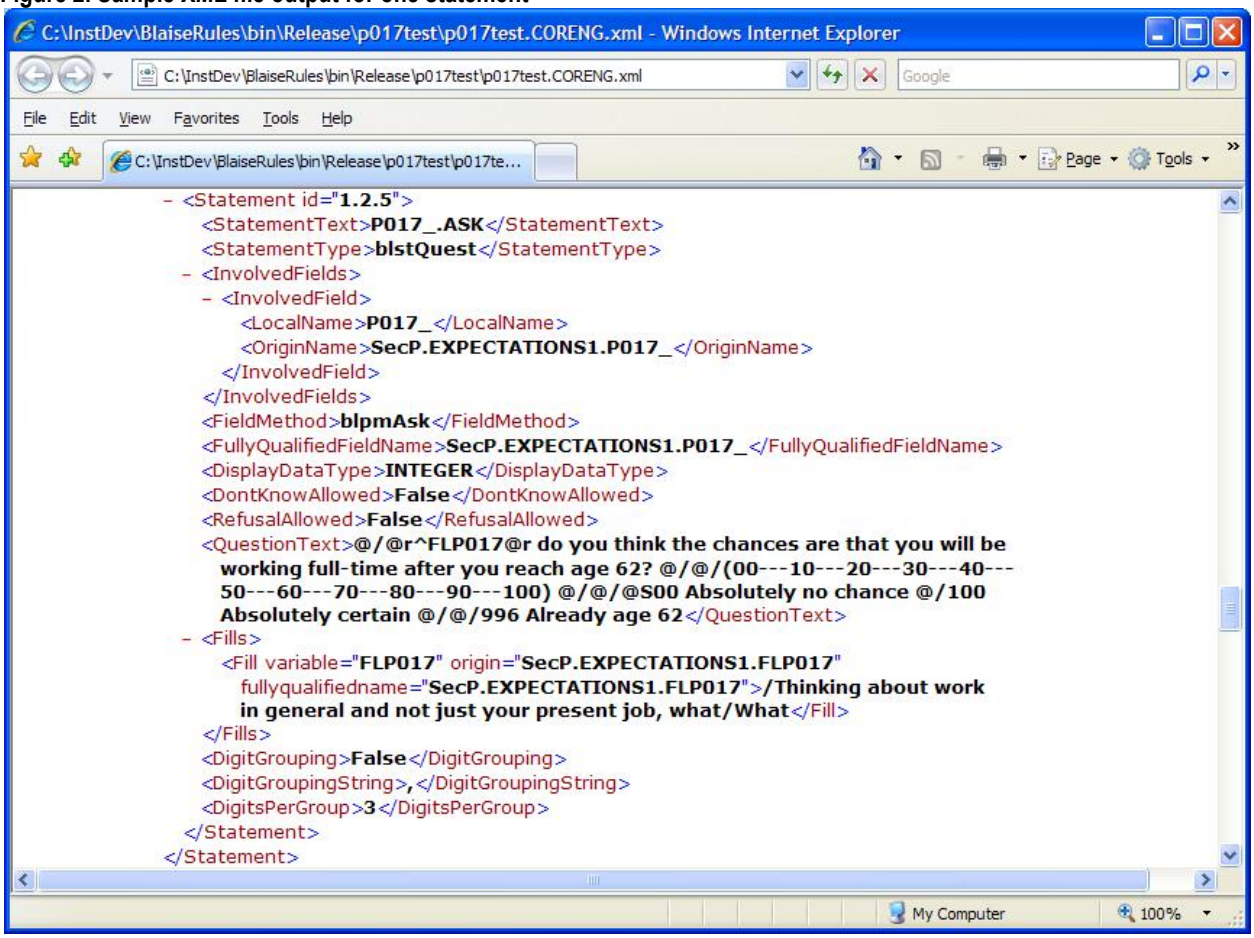
With these data about parameters in hand, it is possible to trace the originating variable of a text substitution upwards through layers of parameters by matching the outside parameter and continuing the search in the parent block. It is also possible to track down assignments to a variable by matching a variable receiving an assignment from the export parameter of a block or procedure with that export parameter so that the assignments to that export parameter can be tracked down in turn.

Finally, in rare cases the API functions in such a way as to prevent tracing these substitutions. Basically, the problem is that a procedure called from inside a loop at any structural depth lower than the loop block is treated differently by the API than a procedure called outside of a loop or in the looped block itself. The `Field` object for the procedure call statement is null in the former case but not in the latter. This prevents us from gathering information about the procedure's parameters as described above. The ideal solution would be for procedures called inside of loops to be handled just like those outside of a loop. It

is hoped that this paper has laid out a sufficient case for the benefits of this functionality to spur interest in resolving these remaining quirks.

There is one implementation issue that deserves special mention with regard to the sort of text substitution program described above. HRS found that it was not practical or efficient to directly use the `RulesNavigator` object to do the tracing described above. In particular, the `RulesNavigator` does not lend itself to free movement within the statement tree, and, at least in many of our early attempts, had a propensity to max out system resources resulting in extremely slow execution or crashes due to the large size of HRS datamodels. Instead, the text substitution program that HRS has built walks through the `RulesNavigator` in a forward-only fashion, but it simultaneously generates a parallel XML document which incorporates all relevant information and is arranged in a similar hierarchical fashion. At each point where a text substitution needs to be resolved, the XML document is used to trace originating variables and assignments exactly as described above. The resolved text substitutions are then incorporated into the XML document as nodes connected to the variable containing the substitution.

Figure 2. Sample XML file output for one statement



XML File Overview

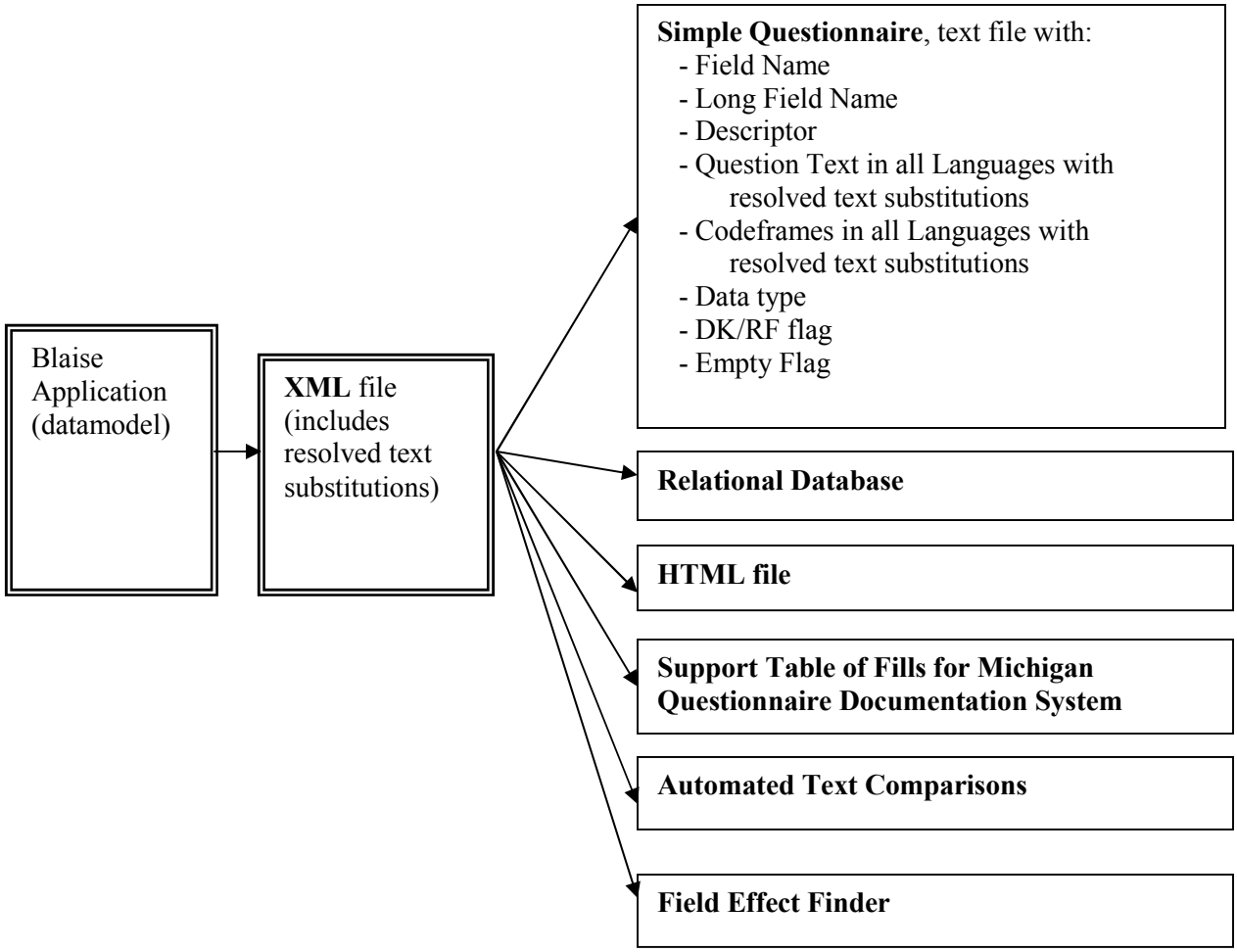
The XML document is handled entirely in memory, despite consuming some 200 megabytes in the case of HRS. The entire process requires between 5 and 10 minutes of execution time on a typical computer. The originally intended output of this process is a delimited text file containing only the substitution information, but a side benefit of generating the XML document is that it can also be saved to disk at the

end of the process and used for other purposes. It turns out to be particularly valuable both because of the resolved text substitutions that are contained within and because the XML seems more accessible than the `RulesNavigator` object to programmers and non-programmers alike. Important uses for this XML output are described below.

Potential Uses of the XML File

One critical function is to accurately resolve text substitutions for input to a metadata documentation system. As a byproduct of the process which handles text substitutions, HRS retrieves a variety of valuable metadata such as field names in ask order as processed by language. The table below describes potential uses of this information.

Figure 3. Potential uses of the XML file with resolved text substitutions for metadata documentation system



Simple Questionnaire

The Simple Questionnaire (SQ) is a formatted output text file designed to document the set of questions in the order and the form they are presented to a respondent, including question wording with accurate text substitutions. The SQ does not contain "ask rules." The procedure to produce an SQ was previously a challenge since many HRS questions use text substitutions. The method described above significantly simplifies the process. In a current version of our SQ generating program, we are using a forward-only

data reader to read the XML file created from the Blaise datamodel. By construction, each question is represented by a separate node in the XML file, with the following attributes attached to the node:

- Field Name
- Long Field Name
- Descriptors
- Question Text with resolved text substitutions
- Codeframes with resolved text substitutions
- Data type
- DK/RF flag
- Empty Flag

As the program reads these attributes for each question node, it produces a formatted output that accurately represents the question texts, codeframes and other prompts that are presented to interviewers during the interview. The output of this process is an easy-to-read text file. The advantage of this approach where we divide the processing from the presentation is that each can be separately modified to accommodate further needs and changes in the final product. This gives us a great deal of flexibility. We can omit, reformat or replace parts of the metadata from the XML file as required for different presentations.

HTML File

The XML file can also be converted to XHTML format by using appropriate XML-to-HTML filters or by adding tags to the nodes. This is a means of altering the presentation in a portable and familiar way. It also allows the metadata to be reorganized in versatile ways.

Portability

Another advantage of a widely-used format like XML is that it can fairly easily be directly imported or programmatically converted to a relational database or other useful storage format. This is helpful for users who want access to this metadata in a way that they are more comfortable with than either the Blaise API or our XML output.

Automated Text Comparisons

This process also makes possible the automation of text comparisons that allow reviewers to easily determine when and how changes were implemented which turns out to be very important for making best use of resources in translation work, among other things. As with the Simple Questionnaire, a small program can be written based on the XML to compare targeted parts of the metadata. There are two work processes in HRS that benefit greatly from the use of such products:

1. Translation comparisons during development. Automated text comparisons between different builds of the datamodel allow translators to easily determine the changes in English that need to be applied to Spanish. Tracking these changes has typically been a very involved, cumbersome, and time-consuming task.

2. Documentation of changes between released datamodel versions. Datamodel changes that affect interviewers need to be summarized and disseminated during data collection periods.

Field Effect Finder

As a side effect of our text substitution process, we get very useful information about how fields and parameters are related in the datamodel. It is possible for a field to influence different parts of the datamodel under different names through parameters. For example, you might have a condition depending on a parameter called *piSex,* which actually represents data in a variable called *Respondent.Gender*. We include nodes in the XML which tell you about these relationships. You can write a simple program that takes a particular variable as an input, reads through the XML, and finds all of the places that this variable of interest affects, even under a different name.

Support Table of Fills for MQDS

This was, in fact the main justification for this programming effort. HRS was concerned that an older mechanism that was being used would not be maintainable in the future. As previously mentioned, extracting text substitutions from the datamodel and using them accurately in documentation has always been a challenge at HRS. This new process to produce accurate text substitutions (fills) has allowed us to replace an older, less accurate, more complex and less flexible process. One of the main output products of this process is a delimited text file containing only the fills. A table called the "support table" is generated from this delimited file and is used as input for the Michigan Questionnaire Documentation System (MQDS), which HRS uses to document the survey. The support table content now has much more accurate metadata than ever before. The resulting output from MQDS now needs less editing, leading to a lesser workload for the editors of this documentation. The documentation's purpose varies, but is mainly used to get study approval from the Institutional Review Board (IRB) at the University of Michigan.

IRB documentation, End-User Documentation and Quality Assurance

While producing documentation for IRB study approval was the original purpose of the new process and products, it will likely become equally important in producing metadata for quality checks and end-user documentation. Some of the other things the new process allows us to do are:

- produce field descriptors and quickly check for duplicates and errors throughout our large instrument
- produce field names in ask order sequence for checking integrity and completeness in documentation
- produce question text and codeframe text and check them for omissions and errors
- produce ask rules for use in easy-to-read formats

Conclusion

In short, HRS was in need of a new product to save time and increase accuracy in metadata processing of text substitutions for IRB documentation to obtain study approval. The application described here took advantage of features currently available in the `RulesNavigator` class of the Blaise API which make it possible to trace the effects of variables throughout the structure of a datamodel. This effort was combined with another longstanding HRS project which converted metadata from the API into an XML document in order to make that information more widely accessible. The intention was to provide a source of metadata with text substitutions in the XML which could be exploited by small, easy-to develop programs for a variety of uses as described here and others that have not yet been imagined.

MetaDEx – From Data Dictionary to Complete Blaise Data Model Code Generation

Farit Vakhaetov, Department of Population Health Research, Alberta Health Services - Cancer Care

Abstract:
Population Health Research is a department focused on population-based epidemiological research in the areas of cancer incidence, causes, survival, and prevention. Blaise has been broadly used in the department to create CAPI data collection instruments from the early 2000s. Based on understanding of the crucial role of metadata definition for effective and robust data collection, management, and analysis, the Meta Data Extended (MetaDEx) Tool was initiated as a utility to create and maintain data dictionaries mainly for surveys, but it can be also used for analytical and other data sets. Its functionality was later extended to generating complete Blaise instrument code. The program was developed in C# and Microsoft Visual Studio 2008. All metadata is stored using eXtended Markup Language (XML).

MetaDEx facilitates a two-step approach in which subject matter specialists prepare specifications, and then IT specialists develop the Blaise CAI instrument. Data specification is structured hierarchically and defines the questionnaire content, text formatting, data flow, constraints, etc. While the system supports the generation of simple Blaise instruments directly from basic data definition, more complex data models require additional programming. The code generation module is built on advanced Blaise specific attributes and parameters which might be specified for any entity in the data dictionary. Additional ability to attach direct Blaise language elements (i.e. LOCALS, PARAMETERS, AUXFIELDS, SETTINGS, etc.) and entire code constructions (i.e. RULES or PROCEDURE definitions) to the section (block) provides virtually unlimited flexibility. Code generation is not limited to initial coding only, but also supports an iterative process of Blaise questionnaire authoring. MetaDEx completely eliminates any manual modifications in the data model script after it is generated, which guarantees congruency between the data dictionary and the actual production instrument in Blaise.

BlaiseIS Sample Management

Hueichun Peng, Lisa Wood and Gina-Qian Cheung - Survey Research Center, University of Michigan

1. Introduction

The University of Michigan's Survey Research Center (SRC/SRO) has been involved in research projects using web surveys as a means of data collection for years. We have used a variety of web survey software packages such as Inquisite and DatStat Illume. Some of the software comes with limited technical systems for sample management, such as tracking survey status, sending out emails and data report download. As these research projects with web surveys are getting dynamic in a variety of manners (e.g., mixed mode data collection and diary types of web surveys), we get ever increasing complex technical specifications and challenges. As a result, we have developed various systems for sample management for web survey projects.

In the last 3-4 years, SRO has initiated exploration of web surveys using BlaiseIS. In August, 2009, we started to do extensive testing and prototyping with BlaiseIS 4.7 for a large scale project, the Army Study to Assess Risk and Resilience among Service Members (STARRS). The original work scope for the Army STARRS project involved using web surveys on a very large scale (400,000+ respondents) and as part of mixed mode data collection in different waves. We made the technical decision to use BlaiseIS as the web survey data collection tool.

In May 2010, we used an internal SRO project to launch key components with BlaiseIS as a pilot project as well as proof of concept for our design and programming work with the Army STARRS Project. Since BlaiseIS does not have a sample management system (SMS), we included programming BlaiseIS sample management functions in our development. The result was very successful.

SRO's implementation of BlaiseIS involves incorporating the web SMS modules within our current Blaise SMS (SurveyTrak) system. We will explain how we designed and managed the process flow of running a BlaiseIS survey, starting from loading sample lines and key preload information, designing and scheduling email jobs, logging survey status, reporting, and closing out the survey.

2. Key Components of BlaiseIS Sample Management

Using the knowledge we have gained from working with web survey projects, we defined five key components we needed for managing sample with BlaiseIS.

1. Link the SMS database with BlaiseIS
2. Create email functions
   - Send email invitations and reminders with customized templates
   - Track bounced-back emails
   - Automatic resend efforts
3. Monitor/control sample
4. Manage/integrate web contact attempts with other contact efforts such as call records
5. Manage survey status and store key BlaiseIS paradata variables in SMS for production monitoring

2.1 Link the SMS Database with BlaiseIS

Linking sample management data with BlaiseIS was first accomplished by parallel preload of sample lines in BlaiseIS and our SMS (SurveyTrak) database. Specifically, the project ID, sampleline ID and login credentials were preloaded into both BlaiseIS and our SMS.

The remainder of this section outlines the process used to allow the respondent to access the BlaiseIS survey and to update our SMS database with key data from BlaiseIS (see figure on next page).

2.1.1 Authentication using our existing SMS/SurveyTrak database

Authentication using our existing SMS/SurveyTrak database involved multiple steps:

1) The Respondent (R) is sent an email directing them to a survey portal-login page. The respondent logs onto a portal page outside of BlaiseIS. This portal-logon page validates credentials in our SMS database.
2) If authentication passes, the R is automatically routed to the BlaiseIS start-up asp page.
3) If the authentication fails (e.g., credentials are not valid, the case has already been completed, or the sample line status is closed out), the R will be routed to customized warning pages or other processing module.

2.1.2 Routing to BlaiseIS start-up page

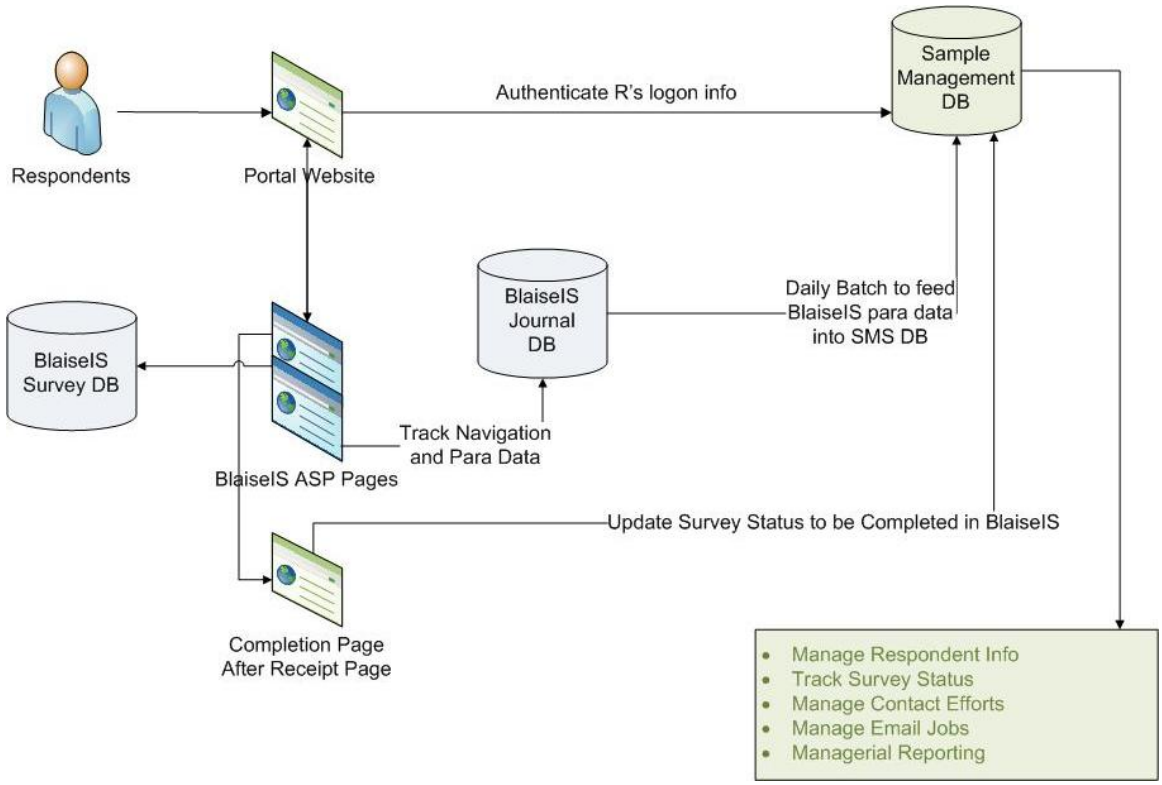
If R passes the authentication at the Portal page, it gets into the BlaiseIS start-up page. For our pilot study, the stored procedure at authentication passes in the logon credentials and the project ID and outputs sample line ID and the location (i.e., URL) of the BlaiseIS start-up page. Then we automatically redirect R into BlaiseIS. The sample line ID is the unique key in the BlaiseIS database.

2.1.3 Determining survey completion

We captured survey completion by adding a second page after the receipt.asp. When R gets to this last page, the page calls a stored procedure in our SMS/SurveyTrak that updates the sample line as the survey is completed. In addition, we update the SMS/SurveyTrak result code to a finalized result code and add a final contact record.

2.1.4 Capturing BlaiseIS paradata

We added modules to capture detailed paradata and navigation data and save these data in a BlaiseIS Journal (see paper by Ostergren and Liu, IBUC 2010). Then we synchronize these paradata (on an aggregate/sample line level) to our SMS as a scheduled job (data was linked using the sample line ID). This information is used for to create managerial reports and for production monitoring.



2.2 Create Email Functionality

We added two modules to (1) automate the process of sending email invitations and reminders, and (2) to capture and log bounced back emails. We developed a suite of C#.Net applications to handle these two processes. For the Send Email module, we used the SMTP Client object of the .Net Mail Namespace; for the Bounce Back Checker module, we used IMAP programming with .NET Sockets Namespace.

Overall, the design of the two Email Modules was to try to minimize human effort in managing the email contacts for web surveys.

2.2.1 Send Email Module

The sending email function is triggered to run at scheduled times that are pre-determined and specified by the project. The email template can be customized with pre-filled information from the sample line. For the pilot project we conducted in May, the sample was divided into two types. For each sample type, there were three templates:

1. Invitation
2. Reminder 1
3. Reminder 2

Criteria for which R should get an email of each type were based on data for the R that was included into our SMS database and the pre-determined dates. For example, if the R has already completed the survey, reminders would not be sent.

When the send email module is triggered, it calls a stored procedure in the SMS database by passing in project ID, type of email, and the time the job is triggered. The stored procedure outputs a record set (i.e., a list of sample lines that should receive the email); our C# .Net program processes the record set and sends out emails one by one. The email has the following information and functions.

- Prefilled information specific to R (such as names)
- Credential data to log onto the BlaiseIS survey
- Portal page URL
- The email content and subject line are customized by pre-defined logic (different email templates for different sample types)
- After the email is sent, the program calls a second stored procedure in our SMS to record a contact attempt record for the R
- Email can be scheduled to run at different times with pre-defined logic
- We use ASCII encoding format for the email body
- Email is sent from a specific email box. The Email Bounced Back Checker (BBC) module checks emails in this email inbox to identify and process bounced back emails.

2.2.2 Bounced Back Checker (BBC) Module

The second part of email program is to track and handle bounced backed emails. The purpose of this module is multi-functional, including:

- Catch the bounced back emails that result from our email sending program
- Record the bounce back in our SMS database
- Inform project and production managers of the bounced-back email
- If possible, launch resend efforts

Bounced back emails will come back to the email inbox used by the Send Email Module. Due to logging by the BBC, this specific email box should only be used by Email Modules and should not to be used for managerial communication purposes with the R. We set up a second study email box, and directed respondents to use this second email box for any replies back to project staff.

The following outlines the process established for the Bounce Back Checker (BBC) module.

- 1st Step: Create a Bounced Back Definition table that specifies the bounced back text to search for and the error codes assigned to each kind of text. This can be configured and customized by project. The table below lists the definitions we used for our Pilot Project.

lvBouncedMessage	vBouncedCode	iErrorCode
The message could not be delivered because the recipient's mailbox is full.	MailBox is full	2006
The destination server for this recipient would not be found in Domain Name Service (DNS). Please verify the email address and retry.	Problem with recipient's server	2006
I'm afraid I had problems forwarding your message.	Could not deliver at time sent	2006
Invalid email address	Insufficient address	2006
MDaemon had identified your message as spam. It will not be delivered.	Return as spam	2006
This Message was undeliverable due to the following reason: The recipient(s) indicated below did not receive this message because their mailbox size limit would have been exceeded.	MailBox is full	2006
Could not deliver the message in the time limit specified. Please retry or contact your administrator.	Could not deliver at time sent	4901
The e-mail system was unable to deliver the message, but did not report a specific reason.	Not deliverable as addressed	2006

- 2nd Step: Logging emails that appear in the study email send inbox. The BBC module opens emails one by one at the root folder of the study's specified email box. The program parses the content (including header) of the email messages in this email box and tries to match the text with bounced back definition texts we defined in step #1.

  For example, we defined that if the program finds "The message could not be delivered because the recipient's mailbox is full", we categorize it as a "Mailbox is full" bounced back and code it with the "2006" bounced back code. Or, if we find the text "Could not deliver the message in the time limit specified. Please retry or contact your administrator", we treat it as a "Could not deliver at time sent" bounced back and code it with the "4901" bounced back code.

  If the parsing finds a match, it moves the email to a processed sub-folder, and logs the email in a Bounced Back Log table with a bounced back code. If the parsing cannot identify the email message, the program either leaves the message in the root folder for managerial checking or moves it to a pending sub-folder. Review of the bounce backs that were not logged could result in new bounced back messages being added to the Bounced Back Definition table.

- 3rd step: The BBC module is triggered to run at scheduled times to process the logged bounced back email messages as a batch. At the end of each batch, the program calls a stored procedure in our SMS, which processes these bounced back logs with pre-defined business rules. For the pilot project, the stored procedure compares the bounced back logs to see if there is any email that matches with the email sent out from our sending email program. If we find one, the procedure inserts a contact record for R and then codes the sample line for R with the specified result code.

- 4th step: For the pilot study, we did not implement re-send efforts if the module finds a bounced back email sent from our program. However, with the Army STARRS Study, we considered very comprehensive business rules regarding whether we should trigger another email from our Send Email module if we have a second email address for the R (and the first email bounced back). Manual review of cases that did bounce back was also an option, which would include allowing the production management staff to update email addresses and flag a case for resend of the invites/reminders.

2.3 Monitor and Control Sample Status

The basic concept is to monitor and track the current status (result code) of the sample at any point in time so that managers can make proper managerial decisions. Since our SMS has full functionalities in this regard, we do not need to make extensive revisions for addition of BlaiseIS sample. What is specific to BlaiseIS is to define another layer of business rules that will drive the BlaiseIS survey, for example, what result codes can be linked with what actions. For instance, if we code out certain sample, the portal will block logins for those sample. Or if we have a diary or journal kind of project, we will track the progress and then route R to correct sequenced journal from the portal.

2.4 Manage and Integrate Contact Efforts on the Sample Line

A lot of web survey projects will still involve phone calls and human tracking efforts like in-person visits or mail communication. Respondents might call in to report problems, opt out of the study, inquire about project details, or even update contact information such as emails, contact address, etc. Our SMS has full functionalities for this part. What we added for BlaiseIS is the logging of the email contact attempts (both outbound attempts such as invites and reminders, and inbound attempts from bounced back emails). These contact attempts can be integrated with contact attempts from the phone or in-person follow-ups. For some projects, the data collected from these contacts might get (pre-) loaded into the web survey itself (such as confirming if the contact information is correct).

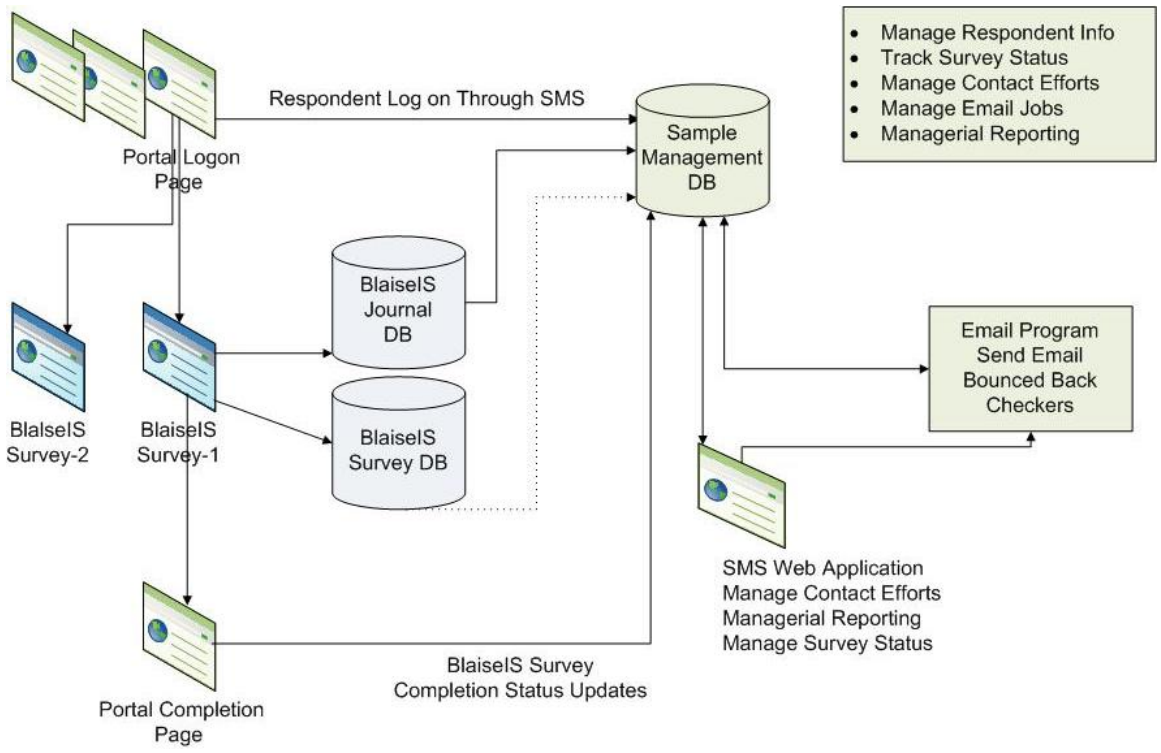
2.5 Reporting

Since our SMS database has integrated the up-to-date tracking of the survey status, contact records, and other BlaiseIS paradata, we are able to set up reporting for the following.

- The cases that have been completed.
- The cases that have been started and aborted.
- Reports showing more detailed information about our sample from the BlaiseIS paradata such as the last question answered, total time in survey, the number of survey logins and break-offs, the browser used, etc.
- The contact efforts for the sample including counts of how many emails were sent out, emails that bounced back, human calling efforts, paper communication, etc.

3. Vision

We propose to use a Portal framework for BlaiseIS surveys. Every R will go into the BlaiseIS world through a common or specific portal. The logon action will interact with the SMS database, route R to correct BlaiseIS Survey Page and track completion status. The integration of BlaiseIS paradata into SMS will facilitate very efficient managerial monitoring and reporting. A lot of managerial functions such as sending emails, checking for bounced back emails and tracking of contact efforts can be integrated into the SMS database.



4. Challenges

Although our pilot using SMS with BlaiseIS was implemented with much success, some challenges were considered as we consider further use.

4.1 Portal concept

The portal concept/framework is not just one or a few web pages; it is a framework that is to wrap the BlaiseIS surveys for sample management purpose. How to design this portal will depend on the complexity of the relationship between the sample and the surveys.

In most cases, we will have one survey for one sample line. We might have multiple surveys activated for the same sample line. Or we could have multiple system entry points that will need access one survey. For example, we invite R to take a web survey, then for follow-up efforts, we might call R and help R to take a survey. This involves mode change on one survey. Ideally this portal concept should be able to handle the variety of operation.

4.2 Cross browser compatibility and consistency

The second challenge is a technical issue. It seems to us that BlaiseIS needs to be tested with a variety of browsers. For instance, we found the behaviors of users pressing browser back/forward buttons are different in different browsers. Since it is extremely hard to control and dictate the browser type when we launch a web survey, to have consistency and compatibility across major browsers will be crucial. As we have learned in a few projects with web surveys, browsers on PDA device have been ever increasingly brought up by R.

4.3 Mixed Mode Sample Management

As we began development of sample management for BlaiseIS, it was with the vision of using as part of mixed mode data collection. As the scope of work changed, our pilot did not include mixed mode sample data collection. Initially we were planning mixed mode data that would be a sequential mode switch from web to phone, not simultaneous. Issues considered included whether we would use a modified version of the BlaiseIS survey or Blaise for decentralized phone follow-up after non-response, how to modify our sample management system to provide feedback to interviewers about the status of the web data collection, and how a mode switch would work when R provided a partial web survey. Developed to be expandable, one of our next steps will be to pilot our BlaiseIS sample management system for a mixed mode data collection project.

Features of Case Management in CAI Systems

Vesa Kuusela, Social Survey Unit, Statistics Finland and
CMS working group set by the Blaise Corporate License Users' Board (BCLUB)

1. Introduction

A Case Management System (CMS) is a central part in all data collection systems which are based on Computer Assisted Interviewing (CAI). In general, case management here means procedures and arrangements for handling sample points[1]. This definition presupposes that a sample with adequate contact information is drawn prior to the actual data collection. This excludes for example RDD sampling methods and quota sampling. This definition also excludes other tasks in survey undertaking, such as the installation of surveys (questionnaires and supporting files) and monitoring of data collection. However, these functions are needed in a CAPI system. Especially, the up-to-date monitoring of fieldwork is an essential task in supervision. Figure 1 shows schematically other relevant parts of a CAPI system and their linking.

Technically data collection modes differ considerably from each other, but even more they differ in how data collection is organised in practice. The fundamental division is between the modes of administration: Face-to-face interviewing (CAPI) and telephone interviewing (CATI) are interviewer-administered modes while web interviewing (CAWI) is a self-administered mode. Especially, in self-administered mode the case management is inherently different from that in interviewer-administered modes. Occasionally, it has been has been desired to establish a CMS that would handle all data collection modes. This paper analyses the typical requirements for a CMS in order to define a universal kernel.

Data collection with CAPI requires a field interviewer organisation which covers sufficiently the region where selected households (sample points) may reside. Interviewers work with laptop computers (in stand alone mode) and they have an arrangement to communicate electronically with the office. The size of a field organisation is determined by the size of country (or the area from where data are collected), time table of data collection, the number of languages, and the actual field organisation. A field organisation may be centralised in the sense that supervising and management is done in one place only, or the organisation may be partly or completely decentralised. A CMS has to comply with the field organisation and with the requirements given for data collection.

In CATI mode, data collection is carried out by a centralised system. Interviewers are working in one or more CATI centres and their work-stations are connected to a network and a file server. The central part of a CATI system is an application called Call Scheduler (CS). The CS feeds cases from the sample file to available interviewers (actually to work-stations) and maintains appointments and management information.

In CAWI, no interviewers are needed. Respondents have to be invited to open the questionnaire and answer the questions. The central component in CAWI survey is the computer system accessible via the Internet, in which the sample is loaded prior to data collection. Case management tasks are inherently different from CATI or CAPI. Within the technical system there may be some tasks resembling case management but they are excluded here.

[1] Case management is called sample management in some occasions.

Applicability of different data collection modes depends essentially on the data provided by the sample file. In many cases, a sample is drawn from an address register or from a database from a previous census. In this case, the sample point is an address but it is not known whether it is inhabited or who is living in it. In some countries, the address may be to a block of flats and one flat may include several households. In this case, the final stage of sampling is carried out by interviewers according to given rules.

Some countries have well updated registers of population and households. Samples from such registers pin point exactly the person or the household to be interviewed. In addition, in some countries there are registers for telephone numbers which can be used for sampling or it is possible to find telephone numbers for persons sampled from the population register. Obviously, email addresses are not available in public databases so that they could not be included in the sample data.

In multi wave surveys, the information about a sample point available before the first wave and later in the subsequent waves may differ considerably. Therefore, also the case managements may be organised differently.

2. Case management in different data collection modes

Because of their technical and organisational dissimilarities, the different modes of CAI set rather different requirements on how CMS should be arranged. Also the possibilities to design the CMS vary considerably between modes.

2.1. CMS in CAPI

The field organisation for CAPI has to be geographically organised because interviewers have to visit the selected households. The average distance between interviewers' homes and potential respondents has to be reasonable in order to keep the travelling costs reasonable. Because of the disperse nature of the organisation, the case management system for CAPI is far more complex than in other modes. In addition, the field organisation partly determines the production process in CAPI surveys.

Characteristic features of the data collection are that all interviewers work with stand-alone laptop computers; they have dedicated samples, i.e. the total sample is exhaustively divided between interviewers and the sample file for each interviewer includes only his or her sample points; interviewers organise their work and make appointments on their own; and they return cases, both completed and non-responses, within the given time limits. This work process introduces three critical tasks for the CMS: case assignment, delivery of sample files to appropriate interviewers, and receipt of completed interviews from interviewers. A special problem in the CAPI case management is how reassignments or case transfers are handled. The reassignment of a case is needed when the interviewer to whom the case was assigned first is not able to carry out the interview. In this case, the case information has to be moved from one interviewer's laptop to the new one.

The interviewer workforce for CAPI can be organised in several different ways and this organisation determines how the case management can be organised. Obviously, the personnel organisation is a more decisive factor than any other part of the production process and therefore, the case management system has to be adapted to it - not vice-versa. Changes in personnel organisation are difficult and slow and they are avoided if other solutions are possible.

From the perspective of CMS, the important question is how supervising is arranged. A major difference between field organisations is whether the supervisors work in one place (centralized organisation) or at regional offices (decentralized organisation). In some cases, the regional organisations are extensive and very strong. The regional offices may even have more influence on survey undertaking than the central office, which may have only a co-ordinating role. On the other hand, in some organisations all

management and supervision is done in one place. Often there are firm historical reasons behind the existing organisation.

Despite the fact that the organisation of the interviewer workforce varies, the technical part of the CAPI system includes always some essential parts as show in the Figure 1. The schematic display refers to an automatic system, but the same tasks appear also in a completely manual system.

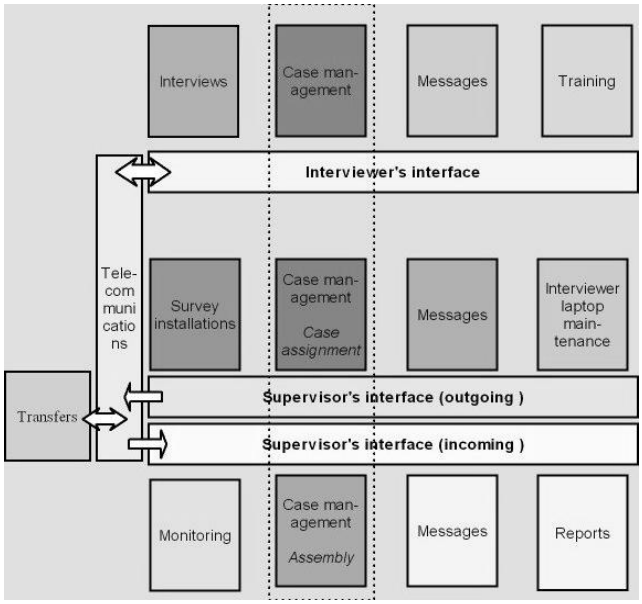


Figure 1: A schematic display of CAPI information system and tasks related to the survey undertaking.

An automated CAPI information system includes three different user interfaces: two for supervisors and one for interviewers. All user interfaces include case management, but its roles differ because the tasks in different phase of a survey are different.

In the interviewers' interface, the case management application passes the sample information to the CAI application (such as Blaise DEP), and transports the interview data and changes in sample information to a place from where it will be transmitted during a telecommunications session (or by other means of transmission). The architecture of the CMS is an important factor in how case management is performed on interviewers' laptops. The solutions can be divided into two major classes: database based systems and object based systems. In a database based system the completed interviews are stored in a database which is transferred to the office system as one file; in object based system each interview is stored in a separate file and the separate files, "objects", are transferred from interviewers' laptops to the office system. The object based system requires a specific application on both ends to handle the objects adequately, but this architecture makes the case management technically fairly straight forward and sturdy.

In the supervisors' interface, the tasks serving outgoing and incoming activities differ essentially also in what comes to case management. In the outgoing part, the most important task is case assignments for interviewers. If it is automated, it requires some sort of a database application which connects respondents to interviewers using geographical information. Therefore, there has to exist a database which includes information on interviewers' whereabouts.

In the interface for handling of the incoming data, the most important part of the case management is the process that assembles data sent by different interviewers to a single data file. Another important part is the application that processes changed contact (sample) data in multi wave surveys, such as the LFS.

A necessary part of a CAPI information system is the communications system which is often a telecommunications system. Typically, CAPI interviewers' communication system is not active all the time. Instead, interviewers contact the central facility at a given frequency .The frequency of communication may vary considerably between organisations. Probably, the most common arrangement is that interviewers open telecommunications system once a day, but the frequency may range from several times a day to once a week or even once in a month. In some countries interviewers do not have communications facilities at their homes. Instead, they have to visit the regional offices to send and receive data.

A communication system is a central part of the CMS and its features partly determine the structure of the CMS. For telecommunications, a CAPI information system requires a telecommunications server that handles the traffic between interviewers and the office. There are several technically different systems in use and obviously they are all built in-house. The most difficult task, however, is to design the communications protocol: how each interviewer receives his or her data once and only once and how interviewers' data are received safely. This protocol is intimately connected to the CMS.

In CAPI surveys, interviewers seek out the responding households using the information they have in the sample file and using their knowledge of the local circumstances. Occasionally the knowledge of local circumstances is the most helpful asset. It is necessary that to some level the case assignment is based on geographical information to keep the distances between interviewers' residences and respondents' homes reasonable.

In some cases, the need for a CAPI arises from the fact that only the whereabouts of the sampling unit (e.g. address of the sampled household) is known, but not who the respondents are or whether the dwelling is inhabited at all. CATI and CAWI are out of the question in such cases. This situation occurs frequently when a sample is drawn from an address register or in connection with area sampling. This situation may also arise if the sampling frame is inadequately updated.

CAPI sets some stringent requirements for the sample information: The locations of sampling units or households' (respondents') addresses are necessary. Names of household members and telephone numbers are not necessary for CMS, but they facilitate interviewers work in setting appointments.

2.1.1. A common CMS kernel for CAPI systems?

The CAI software packages do not include a ready-made CMS. Considering the facilities that have been incorporated for CATI management, it should not be unreasonable to expect some survey or case management facilities to be provided for CAPI. Especially, it would be important for smaller organisations that don't have existing facilities for survey operations. Wensing (2009) presented an example of a CMS for small organisations.

Is it possible to find a common kernel for an automatic CMS? A common kernel requires a definition of a standard CAPI system. The CMS is inherently related to installation of the technical environment for CAPI. Obviously, it has to have separate solutions for the central parts: case assignment, CMS within interviewer's laptop and assembly of a single data file from the received interviewers' files. In addition, an automated CMS cannot be completely detached from the communications systems.

Obviously, the installations of the CMS within interviewer's laptop do not vary too much. For that part a simple solution serving the basic needs is fairly straightforward. It does not solve the total needs but it would help small organisations in the beginning.

Case assignment is at the core of the CMS and it depends essentially on the information available about sample points. In some countries case assignment is partly manual because available information on sample points is not sufficient for an automatic system. For automatic case assignment, it is necessary that there is a sample file that contains respondent information, and a database containing interviewer information. Both databases have to include a field (or fields) that are used to connect a respondent to an interviewer. This field can be called an Enumeration Area Code (EAC). In CAPI it has to be based on geographical information, such as postal area code. The structure of the EAC can be hierarchical and consequently the case assignment can be done in two or more phases. In interviewers database the EA codes have to be unique for each interviewer and the codes have to cover completely and exhaustively the geographical area to be surveyed to end up with proper case assignment.

If no Enumeration Area Code is available, automatic case assignment is not possible and it must be done manually or semi-manually. The Appendix contains a description on how case assignment is done in some organisations.

The end results of case assignment are the sample files dedicated to each interviewer. The files will be delivered to the proper interviewer. Delivery of the files is closely related to the communications arrangements.

2.2. Case management in CATI surveys

A CATI system is typically a centralised facility running in one place (or several places) with interviewers using (desktop) work stations that are connected to a computer network and a file server(s). The sample file is handled by a file server and a program and the cases are not dedicated to specific interviewers. An application called Call Scheduler manages the sample. In contrast to a CAPI system, in CATI there is no case assignment in the same sense and there is no need for sample file delivery. Completed interviews are stored automatically in the database and case status is updated automatically. Because there are no dedicated samples there are no case transfers in the same sense as in CAPI. Similar situations may arise if cases should be transferred from one group of interviewers to another group (e.g. between language groups) but usually CATI systems include tools for supervisors to perform the transfer easily.

A major difference to CAPI is that interviewers working in a CATI centre do not actively organise their own work. Cases are automatically assigned by the Call Scheduler and interviewers are not able to - and supervisors do not need to - influence the process. The Call Scheduler manages the sample by feeding new cases to interviewers according to predefined rules, and maintains the statuses of cases. The Call Scheduler also manages appointments and interviewers have only limited possibilities to influence the process. If manual operations are needed in case management, it is a task solely for supervisors.

In a CATI survey, it is necessary to know respondents' telephone numbers and preferably also their names. Postal addresses (or email addresses) are needed if advance letters will be sent but it is not necessary for CMS.

2.3. Case management in CAWI surveys

Data collection by CAWI does not involve interviewers and consequently there is no case management in the same sense as in CATI and CAPI. Especially, there is no need for case assignment, which is the most demanding task in CAPI.

In a CAWI survey, case management includes installation of the survey on a web server including the arrangements for the authentication of respondents; publication of the survey, i.e. the announcement of the address of the server and login information for respondents, accompanied with the presentation of the survey; follow-up of accumulation of responses; and sending reminders to non-respondents.

In a CAWI survey it is difficult to know the status of cases during the survey because respondents answer at different rates. Even though the answer has not been stored in the server by the given date it does not mean that the respondent is not going to answer later. It is a characteristic feature of a self-administered survey that the collection of data has to be stopped at some stage although answers are still arriving. In interviewer-administered surveys the schedule of data collection is set before the survey begins and the status of each sample point is known.

In a CAWI survey it is necessary to know respondents' names and postal addresses or their email addresses.

3. General case management system for mixed mode surveys

In a mixed mode survey, at least two different data collection modes are used simultaneously or in succession for data collection. For example, data collection starts as a web survey (CAWI) and in the next phase respondents who have not answered are called (CATI) or interviewers visit those respondents whose telephone number is not known (CAPI). Another example may be a CAPI survey in which a refusing respondent is offered a possibility to answer via the web. In multi-wave surveys, different modes are frequently used in different waves, but strictly speaking, they are not mixed mode surveys.

The possibility to carry out a mixed-mode survey depends on the information that is available on the sampling units. More information is needed if all modes should be possible because all modes require different contact information. In cross-sectional surveys this may pose a problem. In a multi-wave survey it is possible to add contact information after the first contact and maybe also to settle the mode of the next interview.

Installation of automated case management for a mixed mode survey is not straight forward because the tasks of CMS are inherently different in CAPI, CATI and CAWI. It is difficult to find common factors in them. Even the interviewer-administered modes, CAPI and CATI, differ essentially from each other in this respect. In addition, the information available on the sample points determines what is possible. For example, if a telephone number is not available, CATI is out of the question; or if there is no adequate address available for respondents, CAWI is not possible.

Obviously the automated case management for mixed mode surveys would require a management server in which a daemon process handles contact information and passes cases from one mode to another. Each mode has a separate CMS incorporating its typical features. A case can be active only in one CMS at a time. Each separate CMS has to know the status of each case. Status includes also information about in which system the case is active. When the status of a case changes, the new status should be made known in each CMS. A technical difficulty is posed by the fact that unlike in CATI and CAWI, in CAPI the connection to interviewers' applications is not continuous. Moving cases from or to CAPI interviewers' workstations is delayed and the length of the delay is not known. In addition, moving cases in or out of interviewer's laptop requires consent from the interviewer.

4. Conclusions

Each data collection mode has its characteristics which set different requirements for case management. In addition, case management may be an important factor behind the efficiency of the data collection system. It is a danger that if case management systems of different data collection modes are forced in a single system the efficiency of data collection in all modes will be compromised.

In a CATI system, the well functioning call scheduler is an important factor behind the efficient data collection. Technically, it would be possible to install a CAPI system with direct access to case management at the office using new wireless telecommunications facilities - even a system similar to a typical call scheduler. However, it is difficult to see any benefits in a system like that. Cases to be interviewed face-to-face have to be directed to interviewers who are living close to the respondents. This, in turn, leads to dedicated samples and the call scheduler does not add efficiency. Probably it would make data collection less efficient.

The design of a common CMS for mixed mode surveys is not straightforward. It is technically challenging, but not too difficult. It is another question whether it is possible to find a common kernel, which all organisations could apply. Probably all organisation who have collected data by CATI and CAPI for many years have a working case management system of their own, which they presumably will use and develop in the future. A common kernel for a CAPI CMS would be most useful for new (and probably small) organisations that are starting computer assisted data collection.

References

Wensing, F. (2009). Development of Survey and Case Management facilities for organisations with minimal survey infrastructure. In *Essays on Blaise 2009*. Proceedings of the 12th Users' conference, Riga, Latvia. (Also at http://ibuc2009.blaiseusers.org/papers/2c.pdf)

Appendix:

CAPI case assignment in some countries

INSEE (France)

In France, case management is done in two phases:
- Phase one: an automatic pre-assignment is done centrally, based on a database containing the area for each interviewer that he/she covers, and the survey that he/she is interested in (this database is updated once a year). This is done by the method unit.
- Phase two: regional offices check the case assignments and can change what they want. The tool for manual re-assignment is unfit for large assignment. This work is done by the interviewers' supervisors.

ONS (United Kingdom)

At ONS, the sample is drawn using the sampling system and stored on a pre-defined table, which contains all sorts of geographical information about the location of each case.

Cases are aggregated into quotas of work for interviewers. This is usually done as part of the sampling so either we select clustered samples from postcode sectors (with a fixed number of addresses selected per postcode sector), or, for unclustered designs, we pre-define a geography and see how selected cases fall within it (producing a variable number of addresses per quota of work).

There is a separate allocation system for assigning quotas to field interviewers. This is de-centralised, the allocation to interviewers is performed manually by field managers, based on the information they have about availability.

Once the allocation is finalised, IT packages the individual cases into case objects, and scatters them to interviewers along with the relevant questionnaire object. We have systems for creating the case and questionnaire objects, and a separate system which places the objects into the correct mailbox, based on the allocations. When the interviewer connects to the Office, the objects are automatically downloaded onto the laptops ready for manual installation by the case management software.

Interviewers work off-line, and need to connect to the Office to transmit completed cases. Casebook zips up the completed case objects and places them in a specific location on the interviewer’s laptops. When the interviewer connects, all objects in that location are picked up by the Gather systems.

Statistics Canada

SC has an in-house interviewer messaging system that allows messages to be passed up and down the reporting hierarchy. Interviewers can only send/receive these messages when they dial in. Interviewers work off-line and transmissions are usually once per day.

Case assignment is manual. For ongoing surveys, where the sample units come back periodically, the previous cycle's assignment information is used as the starting point, and is manually reviewed and revised.

For new samples, the data collection managers in the regions look at the geography information and manually assign the sample to the appropriate interviewer. Generally, the basic information about where the sample is being selected is provided in advance so that hiring decisions can be made

The actual assignment planning activity (assigning cases to interviewers) does not occur until the sample is loaded into the case management database and made available to the managers in the regions.

Statistics Netherlands (SN)

Interviewers work off-line. They have a communications-system on the laptop for connection with the office. When an interviewer connects to SN, the cases (addresses) are automatically downloaded onto his/her laptop. Interviewers connect to SN on a daily basis. At the end of a survey, interviewers return interviews (questionnaires) to SN together with logistic information.

The sample is drawn by a sampling system. The sample includes information about some (background) aspects of the respondents. The sample is then loaded into the system. If respondents have an actual phone number, they will be interviewed by phone (CATI), otherwise (for respondents without actual phone numbers) a face-to-face interview (CAPI) will be held.

Respondents are first allocated to field interviewers automatically by the system. Region Data Collection Managers (all 13 regions have own data collection manager) can make some changes manually in the allocation. They can assign manually the sample to the appropriate interviewer. When the allocation is finalised, addresses are dedicated centrally to each interviewer.

Statistics Norway

The interviewers work off-line from their homes but at least once a day they connect to the office.

A sample is drawn by a separate sampling system and imported into the CMS. The sample is drawn according to our geographic cluster plan (interviewers are located in pre-sampled clusters). The case assignment is done automatically according to the geographic clusters and postcodes. If some cluster is without interviewer (due to illness or vacancy), the field managers do the re-assignment partly manually.

After the case assignment process, cases are prepared for transfer by zipping each in a file. When an interviewer connects to central office he/she reaches his/her own homepage which shows workload and progress, distributed on the different surveys. From the homepage the interviewer starts data transfer. During this process questionnaires and cases are copied out to the interviewer's laptops and completed cases are sent back to the office. The zipped files are unzipped automatically and interviewers find the new cases in the case list. By Statistic Norway the assignments the manual reassignment work, and re-assignments (e.g. in case of illness, slow progress etc.) are time consuming.

Statistics Finland

Case assignment is done centrally and automatically. Each CAPI interviewer collects data on a specified geographical area. The areas are defined using zip codes (postal codes). Technically, the case assignment is done simply by assigning the interviewer code on each record in sample file by a merge procedure based on the zip code. Supervisors do some refinement on the assignments based on interviewer’s workload or temporary leaves.

In the earlier version of the CMS, the sample file was divided in parts so that at the end there was a dedicated sample file for each interviewer. In the current system the whole sample file is sent to each interviewer and the interviewers’ interfaces produces a view into the sample file with only the specific interviewers’ cases.

Case assignment is still based on postal area codes but now the cases for an interviewer are given in a small file that includes only the interviewer code and sampling unit ID (separately for each survey). This file is sent to each interviewer with the total sample file and it is used in the CSM in interviewer's laptop to create the view into the sample. Cases are transferred from one interviewer to another simply by changing the information in this case assignment file. All installations in interviewer laptop take place automatically in the laptop.

Completed cases are returned to the central office automatically when interviewer opens the telecommunications system which takes place usually once a day.

Some other countries

In previous Eastern Block countries the arrangement in survey work is close to the organisation in France. Most of the practical work is done in the regional offices. Automation of survey data collection is still under way but in most cases the plan is to use regional offices. In few countries (e.g. Latvia), the system is centralized. In those countries where CAPI is in use, case assignment is semi-automatic.

Management of CAPI and CATI at the Labor Force Survey

Zipora Radian and Michal Nir, Israel Central Bureau of Statistics

1. Introduction

The Labor Force Survey (LFS) in Israel is a quarterly survey that tracks fluctuations in the Israeli labor force, its size, and its various characteristics.

The LFS is a panel survey comprised of 12,000 dwellings, which are surveyed four times over a period of one and a half years.

To this end, households are interviewed twice through CAPI and twice through CATI.

Most of the cases in the first and forth waves are interviewed through CAPI, and most of the cases in the second and third waves are interviewed through CATI.

There are special cases in which there is a need to change the data collection mode during the investigation period of the same wave.

In the past, in cases where CAPI was problematic, cases were converted to a phone interview according to CAPI procedure. The field interviewers were allowed to conduct these interviews using their home phone. After reexamination of the process, a decision was made to introduce a new practice, where such cases are reassigned to the Data Collection Center using the CATI management system.

This paper presents the overall case management which made it feasible to change the mode of investigation in a short period of time. The focus will be on the process employed during the first wave of the survey.

We will discuss the considerations for using CAPI or CATI during the same wave. We will also describe the different steps and activities that were taken by the interviewer, the CAPI manager, and the system as a whole, in order to change the mode of investigation from CAPI to CATI.

2. The computerized data collection system in LFS

Computerizing the data collection system of LFS began in 1999. First, telephone interviews which were conducted during the second and third wave were moved to the Blaise CATI management system, which was conducted from the Data Collection Center located in Jerusalem. The transition to CAPI mode for the first and the fourth waves, as well as the overall management system for both modes, was completed in 2008.

2.1 Data collection facilities

The system is able to combine several tools, and by doing so to enhance their efficiency of use. This is done as follows:

The questionnaire is written using Blaise 4.6.

The interviewer's management on the laptop computer is written in Maniplus.

The CAPI and CATI supervisor management system is written using .Net.

Management of CATI interviewing is done by Blaise CATI Management.

The questionnaire data is stored in the Blaise database, and the survey management data is stored in an SQL2005 server.

The overall management system is written in .Net, combining Blaise and .Net in two ways: from data entry program (Dep) to .Net and from .Net to Dep. (For more details see: Har & Luskin, 2009).

2.2 Main functions of the Survey Management System

The survey management system allows:

- Assignment of cases for mode of interviewing in accordance with predetermined criteria, such as the wave number.
- Assignment of cases to interviewers by CAPI supervisor.
- Transmission of data.
- Keeping track of non-response cases and contact trials.
- Moving cases from CAPI to CATI and vise versa.
- Managing the response status for each case.
- Producing follow-up reports on response rates and outputs.

3. Field work process in LFS

3.1 General

The sample drawn each year is divided into four panels, each allocated to the field work, one after the other.

Each panel is investigated four times according to the following pattern: The first quarter of the investigation (wave no 1), the following quarter (wave no 2) a break of two quarters and two more investigations in the following two quarters (wave 3 and 4 respectively) that are parallel to those of the first two investigations.

The survey is carried out continuously each and every week for the entire three months of the investigation period of each wave. Each week about 1/13 of the households included in the survey are interviewed.

The entire period devoted to the investigation of each case is one pre-determined week, plus three additional weeks which are allocated for unresolved cases (a total of four weeks).

3.2 Data collection methods

The investigations are conducted in two major methods: face to face interviewing using CAPI is conducted in the first and the fourth waves, telephone interviewing using CATI is conducted in the second and the third waves.

Although face to face interviewing is the most expensive method for data collection, it is essential to use it in the first wave for the following reasons:

- Correct identification of the sample unit (the dwelling).
- Exclusion of ineligible cases, such as unoccupied or non-residential dwellings.
- Finding solutions to tracing problems.
- The need to verify personal demographic data through personal documents such as identification cards.
- Reducing refusals to cooperate by using personal contact and reassuring respondents as to the importance of the survey.
- Developing rapport with respondents improves cooperation in future contacts during the next wave.

The fourth wave is conducted by CAPI because it includes an income survey and there is a need to present supporting documentation, like salary slips, and financial reports.

The second and the third waves are conducted by CATI in order to save costs. The cost of one case conducted through CATI is about 13 percent of that of CAPI.

While each wave has a preplanned mode, there are special cases in which an additional mode is required in order to maximize response rates. Following are some of the reasons for using the different investigation modes:

In waves conducted by face to face interviewing: limited number of interviewers, lack of time, remoteness and security problems makes it impossible to send interviewers in some cases. These cases are transferred to phone interviewing.

In waves conducted by telephone interviewing: At times, personal visits are needed because there's a lack of correct phone number.

4. Data collection procedure before transition

4.1 The collection process

Steps taken at the first wave start at the overall management system, continue to the management system of the CAPI supervisor, and than to the management system of the field interviewer using his/her laptop.

This is done as follows:

- Assignment of all cases in the first wave to three different regional centers;
- CAPI supervisor assigns a workload of about 12 cases to each interviewer;
- CAPI interviewer makes personal visits to his assigned cases;
- Data of each visit is recorded in the Blaise questionnaire on the laptop;

- The interviewer indicates all the non-response cases that require supervisor intervention;
- CAPI interviewer transmits data each night through his home telephone line;
- According to supervisor's guidance and authorization, the field interviewer makes more contact attempts to selected non-response cases, using his personal home phone;
- Completed cases are transferred to the main office for processing;
- At the end of the four weeks (section 3.1), non-response cases attain final status according to the information provided by CAPI interviewers.

4.2 Disadvantages and advantages of phone attempts using the CAPI system

While most of the cases in the first wave are conducted face to face, the need to meet a tight schedule sometimes requires the use of the telephone method. It was found that 18% of all cases in the first wave were transferred to telephone interviewing.

When CAPI was first introduced to the LFS in 2008, when there was a need, a case was converted to a phone interview according to CAPI procedure. As mentioned earlier, field interviewers were allowed to conduct these interviews using their home phone.

4.2.1 Disadvantages

- The interviewer's main assignment is the personal visit; therefore s/he has only a limited amount of time in which to make contact attempts by phone.
- Administrative details such as time invested and number of phone attempts are provided by the interviewer. As such, they are less reliable than if they were automatically recorded.
- It is not possible to monitor the interviews in real time in order to assess their quality.

4.2.2 Advantages

- The CAPI interviewer is fully in charge of all efforts made to interview all assigned cases within the allocated four weeks.
- Since the CAPI interviewer visits the dwelling in person, s/he has detailed information on the dwelling and on the respondents. This information is very important for future further contacts.

5. Transition to the new procedure

The transition to the new process was encouraged mainly because of the disadvantages of the previous procedure. Moreover, the use of an overall management system allowing a combination of CATI and CAPI procedures has driven us to issue a new approach.

In April 2009 we introduced a new procedure, where non-response cases of the first wave were reassigned to the Data Collection Center using the CATI management system.

Interviewers working in the Data Collection Center were responsible for conducting telephone interviews in the 2nd and 3rd waves of the LFS. The same interviewers are now also responsible for conducting telephone interviews in the 1st wave.

5.1 Advantages of the new procedure

The CATI management system has several advantages which improve the efficiency of data collection and the reliability of the data:

- The investigations are conducted from one Central Collection Center by a group of CATI interviewers.
- The interviewers use an automatic call scheduler with an option for making appointments and prioritizing urgent cases.
- Assigning special cases to appropriate interviewers, such as bilingual interviewers.
- Monitoring of the interviews by supervisors and giving immediate feedback in real time.
- The CATI management system tracks administrative information, including dates and times of interviews, as well as the duration of each interview.

5.2 Disadvantages of the new procedure

- CAPI interviewers were previously used to have full responsibility for the process of making contacts attempts and interviewing during the whole investigation period. In the new procedure, responsibility for some of the cases is divided between CAPI interviewers and CATI interviewers.
- A second problem could arise concerning the information CAPI interviewers supply as to the contact attempts.

5.3 Steps of the new procedure

Similarly to the previous procedure, all cases of the first wave are assigned to the regional centers, and then assigned by the supervisor to each CAPI interviewer. Unlike the previous procedure, unresolved cases in which it was decided not to make any more face to face contact attempts are transferred to the CATI management system. This involves the following steps:

- Before transferring a case to the CATI management system, the CAPI supervisor verifies that there is a valid telephone number in the questionnaire.
- In his management system the CAPI supervisor indicates the case as suitable to move to CATI.
- The overall management system creates new questionnaires for the above cases.
- The new questionnaires are included in the "day batch" created by the CATI supervisor and delivered to the CATI interviewers.

5.4 Special considerations

When we redesigned the process special attention had been paid to certain factors:

5.4.1 Keeping detailed information

The CAPI interviewer can record detailed information obtained during up to 8 visits.

For each visit s/he can record the date, the time of visit, the response status, the non-response reason and remarks.

The remarks are very important for further attempts, especially when another interviewer will handle the case, for example:

> "I met only young children, the parents usually return after 8 pm."
>
> "I met an old man who speaks only Russian."

All this information has to be fully recorded and available in the overall management system.

5.4.2 Using CAPI recorded information in CATI investigation:

The first screen of the CATI questionnaire was designed to include the latest remarks recorded by the CAPI Interviewer. Also presented are the last reason for non-response recorded in the CAPI questionnaire and the CAPI supervisor's remarks.

5.4.3 Time table consideration

Special attention is given to managing the case by CAPI interviewers within the first 2 weeks of the wave. This is done so that, if necessary, enough time will be left in order to make the transfer to CATI for phone attempts.

5.4.4 Final status

In non-response cases, the final status is determined by combining information from both CAPI and CATI recorded details. For example:

If the CAPI interviewer recorded "language problem" on his last visit, and the result of all the calls in CATI were "no answer", the final non-response reason will be "language problem" ("no answer" is secondary to "language problem").

5.4.5 Monitoring tools

The monitoring tools after transition have to allow control on both CAPI and CATI systems. These include:

- Lists containing all cases that have been relocated to CATI, the date of relocation and the current status of the case.
- Tables containing the response rate of each mode.
- CATI history files which permit viewing the number of call attempts and the results.

These monitoring tools allow the Center office supervisors to check the case status and to avoid duplication and missing cases.

6. Summary of operating the new first wave procedure

Looking back, it seems that, altogether, the transition to the new procedure was successful.

Cases were transferred to CATI rapidly and efficiently, and CATI interviewers handled the cases successfully and achieved a good rate of response. Most importantly, there were no "missing" cases or duplicates.

However, some issues were problematic: adjusting to the new procedure was difficult on CAPI and CATI interviewers and on their supervisors. CAPI interviewers found it difficult not to be allowed to complete handling a case in which they have invested a lot of efforts. CATI interviewers complained that the information transferred from CAPI was incomplete, and not presented in a user friendly way. Moreover, they complained that they now had to deal with extra problematic cases.

7. Conclusions

Utilizing the new procedure successfully convinced us that our overall managing system is flexible enough to adjust to the use of the new procedure in the LFS.

New means of training are to be designed to improve the cooperation between the CAPI and CATI personnel. Also we are looking for better ways to present relevant information to the CATI interviewers.

8. Future plans

At the beginning of 2012 the LFS will change to be a monthly survey. The investigation period will be shorter (two weeks instead of four), and there will be eight waves instead of four.

We intend to employ the new procedure of the first wave in the monthly survey. This will be done within a new management system planed using Blaise 4.8, which will replace our current Blaise 4.6.

Reference

Shifra Har and Evgenia Luskin, *Different Methods of Working with Blaise in the Israel Central Bureau of Statistics*, Proceeding of the 12 international Blaise Users Conference 2009, Riga, Latvia.

BlaiseIS Paradata

Jason Ostergren and Youhong Liu, The University of Michigan

Paradata are interview data that are collected from supporting systems rather than the interview script itself. They are used heavily by the Institute for Social Research at the University of Michigan for processes like quality assurance, detection and fixing of problems, testing, instrument design and more. This paper will examine our efforts to prepare paradata collection mechanisms for Blaise IS that will be adequate to our needs at ISR. It will first cover page-level paradata, which are mostly handled server-side. Then it will discuss sub-page-level paradata that are collected on the client side using javascript.

Blaise includes a feature called a journal to record events and user actions that occur during a web interview. At the University of Michigan, we began to explore BlaiseIS earlier this year; as part of that exploration, we needed to make sure we could capture the paradata we need. This paper will discuss how the web journal can be implemented for paradata capture. We will discuss how to define a useful paradata data structure. For some paradata capture, modifications can be made to the default style sheet and the page handler ASP. For some other data, large text fields in the survey data model are used, so information from the survey session data can be extracted. We will also talk about how different types of fields are being processed and how to utilize the BOI database type and the type of partition used. Finally, we will discuss the data, procedures and process required to output the journal data. What follows is a step by step guide that illustrates how ISR has handled page-level paradata.

Capturing Page-Level Paradata

The purpose of paradata is for assessing data quality and for verification. Paradata are critical to project staff for checking the survey contents and are also important for verification of programming accuracy. For instance, the timing difference from one page to another indicates the time a respondent spent on a page. If the average time on one page is too long or too short, then there is a potential problem with the question text or answer categories on that page. Another example is the "Action" variable in the paradata that keeps track of how respondents use the survey navigation buttons: Next, Previous and Submit. These are very important user behaviors that can be used to evaluate a survey design. In the paradata, the CurrentPageQuestions and CurrentPageAnswer are useful in two ways: first, they can be used to recover the survey data in case the survey data is corrupt; second, they can be used to trace a user's answers from the beginning to the end. In the next section of the paper, all fields in the paradata are discussed in detail.

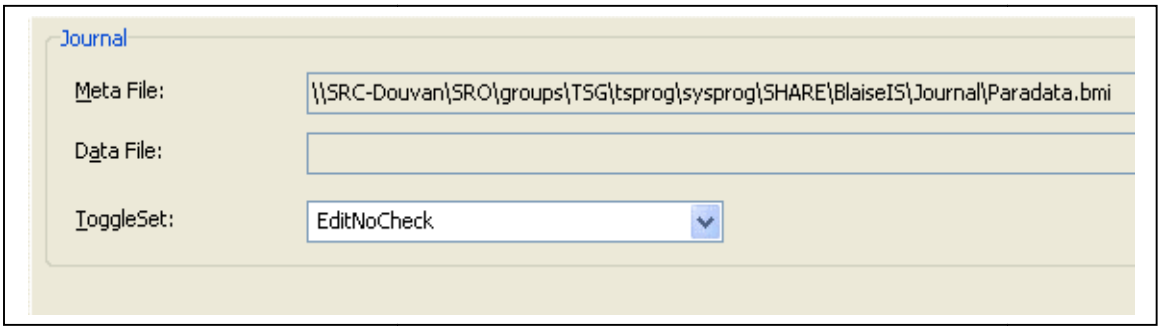
Step 1: Define a Journal Datamodel

The structure listed below shows the Journal datamodel for capturing page-level paradata that ISR has adopted. Each field will be described in the following steps. Please note that in order to use the Journal, the survey instrument must have a primary key.

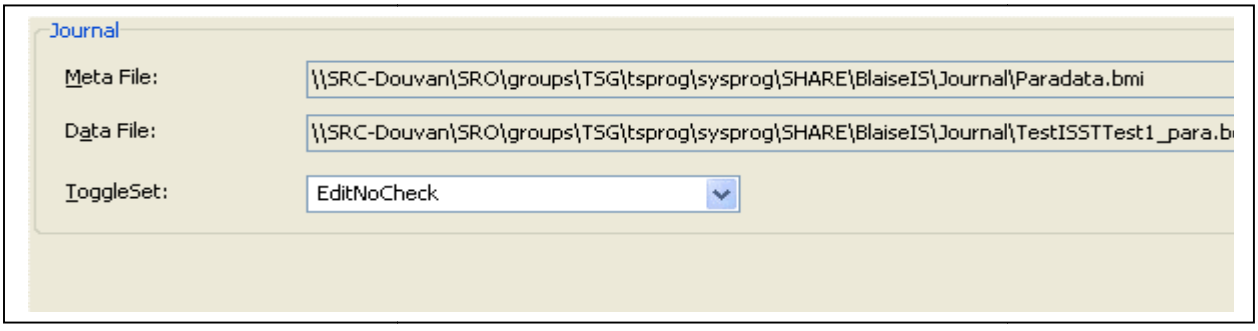
```
Datamodel Paradata

SECONDARY
  keyTimeStamp = PageStartTimeStamp
  keyPrimaryKeyValue = PrimaryKeyValue
  keySession = SessionID

FIELDS
  PageStartTimeStamp : string[30]
  PrevPageTimeStamp : STRING[30]
  PrevPageLength : real
  SessionID      : STRING[20]
  PrimaryKeyValue: STRING[200]
  vProjectID  : string [40]
  Mode : STRING [10]
  ScreenSize : STRING [50]
  BrowserSize : STRING [50]
  JavascriptEnable: STRING [10]
  ConnectionSpeed : STRING [20]
  SurveyName : STRING[50]
  Browser : string[100]
  CurrentPageNo : INTEGER
  PrevPageNo : INTEGER
  VersionDate : string[20]
  VersionTime: string[20]
  Action : string [20]
  PrevPageQuestions : string[2048]
  PrevPageAnswers : OPen       {H}
  CurrentPageQuestions : string[2048]
  CurrentPageAnswers : OPen
  SubmitStatus : String [20]
ENDMODEL
```

Step 2: Specify the Journal BMI file and data file

The second step is to specify the Journal description in the Interview Specification. Select the Server Settings in the Specification and push the button in the Journal panel. Fill in the name of your prepared datamodel for the Meta File. If you want to give a special name to the Journal database, specify this in the Data File textbox. If no data file is specified, the system will create a Blaise database that has the same file name as the datamodel name.



You can also use a BOI file for the data file. In this case, you will need to specify the BOI before using it. Journal BOI files will be discussed later.



Step 3: Update survey ASP pages

In the survey ASP pages, the Blaise Journal component is used to insert records to the Journal database. Most Journal database updates are within the Page Handler ASP page. Here is an explanation about how the fields are updated:

a. SessionID – It is straightforward to get the sessionID:
```
Journal.Fields.Item("SessionID").Text = Session.SessionID
```

b. CurrentPageNo - Current Page number can be obtained from the BlaiseIS BlaiseISSessionState of the Blaise Internet HTML/XML Generator Component:
   - `Set SessionState = Server.CreateObject("BlHXG3A.BlaiseISSessionState")`
   - `CurrentPageNo = SessionState.StoredPageIndex`
   - `Journal.Fields.Item("CurrentPageNo").Text = CurrentPageNo`

c. PrevPageNo – Previous Page Number can be obtained from a hidden variable in the form:
   - `PrevPageNo = cstr(Request.Form("spi"))`
   - `Journal.Fields.Item("PrevPageNo").Text = PrevPageNo`

d. Action – page actions are derived from the CurrentPageNo and PrevPageNo:

```
Dim Action
IF CurrentPageNo<>"" AND PrevPageNo<>""  THEN
  IF Cint(CurrentPageNo)>Cint(PrevPageNo) THEN
     Action = "Next"
  ELSEIF CurrentPageNo<PrevPageNo THEN
     Action = "Previous"
  END IF
ELSE
  IF PrevPageNo="" THEN
      Action = "Start"
  ELSEIF CurrentPageNo="" THEN
      Action = "Submit"
  END IF
END IF
Journal.Fields.Item("Action").Text = Action
```

e. SubmitStatus – When the survey page is redirected to the Receipt page, the Submit Status is assigned to “Completed”. For other actions, the status is empty.

```
Journal.Fields.Item("SubmitStatus").Text = "Completed"
```

f. PrevPageQuestions and PrevPageAnswers – Obtaining the previous page questions and Previous page answers requires changes to the Blaise default stylesheet.

1) In the XSL style sheet, first create the field name list. The format of the list is Form name and Blaise field name: e.g, qu1:BLAge qu2:BLName. After the field name list is created, it is then saved in a form hidden variable named Para.pagefieldnamelist.

```
<xsl:variable name ="fieldnameList" >
  <xsl:for-each  select ="/BlaisePage/descendant::Question">
    <xsl:value-of select ='concat(@ID, ":", @Name, " ")'/>
  </xsl:for-each>
</xsl:variable>
<input type="hidden" name="para.pagefieldnamelist" value ="{$fieldnameList}" />
```

2) In the Page Handler ASP, all Request.Form variables are examined. If a variable is matched up with a form-hidden variable in Para.pagefieldnamelist, then the corresponding Blaise Field name is extracted. The Request.Form variable's value is also saved. It is possible that a survey page contains more than one field, so the questions and answers are actually caret ("^") delimited strings: e.g. PrevPageQuestions = BLAge^BLName^ and PrevPageAnswers =46^John Smith. The answers and questions are in the strings correspondingly, so it is very easy to write a procedure to construct Name:Value pairs.

```
For each Field in Request.Form
  If Left(Field, 2) = "qu" And IsNumeric(Right(Field,Len(Field)-2)) then
    Dim Pos1, Pos2,FName
    Pos1 = instr (1, cstr(Request.Form("para.pagefieldnamelist")), Field + ":")
    Pos1 = Pos1 + len(Field + ":")
    if pos1 >=1 THEN
       Pos2 = instr (pos1, cstr(Request.Form("para.pagefieldnamelist")), " ")
       Fname = mid(cstr(Request.Form("para.pagefieldnamelist")), pos1, pos2-pos1)
       PrevPageQuestions = PrevPageQuestions + Fname + "^"
       PrevPageAnswers = cstr(PrevPageAnswers) + cstr(Request.Form(Field)) + "^"
    END IF
  End if
```

g. CurrentPageQuestions and CurrentPageAnswers: these two fields can be obtained from the HandleRequest method of Blaise Page handler object:

```
Dim PageHandler
Set PageHandler = Server.CreateObject("B1PgHn3A.BlaiseISPageHandler")
Set XMLDoc = PageHandler.HandleRequest
Dim Question, Questions
Set Questions = XMLDoc.documentElement.SelectNodes("descendant::Question")
Dim CurrentPageQuestions
Dim CurrentPageAnswers
CurrentPageQuestions=""
CurrentPageAnswers=""
For each Question in Questions
  IF GetAttribute(Question,"Visibility") = "AskField" THEN
      CurrentPageQuestions = CurrentPageQuestions + GetAttribute(Question,"Name") + "^"
      CurrentPageAnswers = CurrentPageAnswers + GetAttribute(Question,"Value") + "^"
  END IF
Next
```

h. ProjectID, VersionDate,VersionTime, Mode, SurveyName – these are the special fields defined in the survey data model. They are assigned fields, e.g. VersionDate := VersionDate := TODATE (2010, 07, 13). They can be extracted from GetReceiptInfo of the Blaise Page handler object:

```
set xmlReceiptInfo = PageHandler.GetReceiptInfo() 'yliu 20100103 fields info
ProjectID = GetAttributeFromField(xmlReceiptInfo.documentElement, "ProjectID", "Text")
VersionDate = GetAttributeFromField(xmlReceiptInfo.documentElement, "VersionDate", "Text
VersionTime = GetAttributeFromField(xmlReceiptInfo.documentElement, "VersionTime", "Text
Mode = GetAttributeFromField(xmlReceiptInfo.documentElement, "Mode", "Text")'
SurveyName =GetAttribute(xmlReceiptInfo.documentElement, "DictionaryText")
```

i. ScreenSize, JavascriptEnable, ConnectionSpeed, BrowserSize – these fields require adding Javascript in the XSL page. Then the values can retrieved from Request.Form variables:

```
cstr(Request.Form("paraScreenSize"))
cstr(Request.Form("paraJavaScriptEnabled"))
cstr(Request.Form("paraConnectionSpeed"))
cstr(Request.Form("paraBrowserSize"))
```

```
<xsl:if test="$paraSessionPageCount='0'">
  if (navigator.appName.indexOf("Microsoft")!=-1)
  {
    document.f.paraBrowserSize.value =
    document.body.clientWidth + 'x' + document.body.clientHeight;
  }
  else
  {
    document.f.paraBrowserSize.value = window.innerWidth + 'x' + window.innerHeight;
  }
  document.f.paraScreenSize.value = screen.width + 'x' + screen.height;
  document.f.paraJavaScriptEnabled.value = 'true';
  var testImage = new Image();
  testImage.onload = function(){
  document.f.paraConnectionSpeed.value = (new Date()).getTime() - time1;
  };
  var time1 = (new Date()).getTime();
  testImage.src = 'images/BlaiseISLogo.gif?' + Math.round(Math.random() * 10000);
</xsl:if >
```

j. PageStartTimeStamp, PrevPageTimeStamp and PrevPageLength are obtained through time calcuations in combination with a session variable:

```
timestamp = year(now) & right("0" & month(now),2) &
          right("0" & day(now),2) & right("0" & hour(now),2)
          & right("0" & minute(now),2) & right("0" & second(now),2)
          //format: YYYYMMDDHHMMSS
Journal.Fields.Item("PageStartTimeStamp").Text = timestamp
Journal.Fields.Item("PrevPageTimeStamp").Text = Session("prevPageTimeStamp")
//retrive from the session varaible
Journal.Fields.Item("PrevPageLength").Text =
Cstr (getPrevPageLength (timestamp, Session("prevPageTimeStamp")))
//getPrevPageLength function returns time differece in seconds
Session("prevPageTimeStamp") = timestamp
//save it back to the session variable
```

Step 4: Use BOI Database for Paradata:

BOI file usage

a. As with survey data, we choose to use the BOI data format to store paradata. In comparison to standard Blaise data files, it has the advantage that data can be stored on a more secure location than an internet server.

a. Data partition type – Flat no block has advantages for the Journal:
   - It's good for small datamodels like the Journal
   - It allows stored procedures and other relational database tools to be used to manipulate paradata directly

c. The code below is a sample query to extract data form the OLE DB database. Note that for string lengths greater than 255 and open type fields, the data are located in the para_Open table, so it is necessary to use sub queries to get those data.

```
SELECT  [PARADATA].*,
(select OPENTEXT  from PARADATA_OPEN where FieldID =
 (select ID from PARADATA_ID where Name =  'PrevPageQuestions')
 and JOINKEY = PARADATA.JOINKEY ) as PrevPageQuestions,
(select OPENTEXT  from PARADATA_OPEN where FieldID =
 (select ID from PARADATA_ID where Name =  'PrevPageAnswers')
 and JOINKEY = PARADATA.JOINKEY ) as PrevPageAnswers,
(select OPENTEXT  from PARADATA_OPEN where FieldID =
 (select ID from PARADATA_ID where Name =  'CurrentPageQuestions')
 and JOINKEY = PARADATA.JOINKEY ) as CurrentPageQuestions,
(select OPENTEXT  from PARADATA_OPEN where FieldID =
 (select ID from PARADATA_ID where Name =  'CurrentPageAnswers')
 and JOINKEY = PARADATA.JOINKEY ) as CurrentPageAnswers
 FROM [PARADATA]
```

Capturing Client-Side Paradata

The above discussion focuses on page-level paradata that are captured when a button is clicked to leave a page. While that can provide useful information about time spent on an entire page or backtracking across pages during an interview, it does not approach the level of granularity of a CATI audit trail which captures keystrokes and mouse clicks. These sorts of details ("client-side paradata") can be captured with javascript while the respondent is inside the page and then saved by the same mechanism as the page-level paradata when the interview advances. We will now discuss a scheme for modifying the default Blaise IS javascript and xsl files to capture client-side paradata such as changing answers within a page, keystrokes within an input box and scrolling.

When deciding how to implement client-side paradata, we paid particular attention to the footprint. We wanted to minimize risk by changing as little as possible, especially since we had decided to try to piggyback on existing Blaise IS xsl and javascript files. We also thought it was necessary to try to minimize the size of the paradata collected because of bandwidth considerations. Although the paradata were unlikely to be large enough to ever affect performance, the amount one could collect from each page is almost unlimited in theory (scrolling, all keys…) and the quality of respondents' connections are not uniform.

The paradata of most interest for quality assurance and testing purposes are those that allow the reconstruction of user input after the fact (for reproducing problems, mainly). What we want to see here is the order in which user input occurred and focus changed. This information can help to reveal bugs of various types or user friendliness problems. Because it is common that routing occurs between pages rather than between questions on a page, the page-level paradata collected on the server side are usually sufficient for this purpose. However, when there are scripts running on a page which may be affected by the sequence of events, having access to the granularity afforded by client-side paradata could be very important. Client-side paradata could also provide helpful clues about confusion or interactivity problems on a page by recording repeated switching of answers or attempts to do something not handled in the original design. Finally, client-side paradata could help reconstruct why big failures like missing data occur. For example, in a web survey fielded in 2009 by the Health and Retirement Study (HRS) it was found that on a page with a table where respondents were expected to select a radio button and then put an answer in a corresponding text box, an unexpectedly high number of the boxes in the first row were blank. If HRS had been collecting CSP at the time (which it was not) it would have made it easier to determine the source of the problem and reconstruct lost data, if any. Strategies for collecting a reasonable amount of paradata about which controls had click events and for collecting relevant keystrokes will be discussed in detail below.

This same information could also prove useful for methodological purposes, particularly by pointing out bad design decisions revealed in some of the ways described above. In addition, data about time between various events, clicks on hyperlinks, or scrolling activity within the page is all potentially valuable. Strategies for handling each of these types of paradata will also be discussed below.

The basic format we have chosen for this client-side paradata project is not dissimilar to audit trail data that can be produced for Blaise surveys that use the DEP program. Each line indicates some sort of user-initiated event and begins with some sort of timestamp. One key difference is that the times recorded are elapsed times relative to previous events rather than absolute times. The main reason for this is that it results in less data, but it also helps to highlight the fact that any absolute time recorded on a client machine is less meaningful perhaps than one recorded in a call center in a CATI interview because the settings on the respondents machine and the connection quality are outside of our control. This notation owes its inception to Dirk Heerwegh's CSP project explained at this URL: https://perswww.kuleuven.be/~u0034437/public/csp.htm. The remainder of the data on each line either

provide identifying information about the field involved and its value after the event, or about something like scrolling activity that occurred since the last event.

The client-side paradata mentioned above can be captured by adding a few functions and some function calls to the following files used by Blaise IS: biEvtHan.js, biHTMLWebPage.xsl, biPagEvt.js, and biStatus.js. A hidden input box must be added to the FormPane in the biHTMLWebPage.xsl, which can collect the CSP for a page until it can be saved by the same means as the rest of the page-level paradata when a page navigation button is pressed. The CSP is collected using four new functions we added to biEvtHan.js (code provided at the end of the paper) that handles recording the paradata, keystrokes, scrolling, and hyperlinks.

The recording function subtracts the time of the previous event from the current time to get the elapsed time value and concatenates it into a line with the fieldname and value. If a scrolling event occurred since the last event, it first writes the elapsed time at the last scroll event and whether the scrolling was horizontal, vertical, or both. The recording function takes as arguments references to the HiddenStatus control, the visible control, and also the value of the visible control (in the special case of a checkbox, the value is constructed as follows: *'#' + CheckBox.value + '=' + CheckBox.checked*). The function gets the Blaise fieldname from the *.parentNode.id* property of the HiddenStatus control. The value argument is used directly, except in the case of keystrokes as described below. This function is called from the following already-existing functions which are otherwise unchanged: OnClickRadioButton in biEvtHan.js, and FillAnswer, OnChangeDropDown, and OnClickCheckBoxStatus in biStatus.js.

A function which acts as an intermediary when keystrokes are involved takes an event argument instead of a value and converts *.charCode* or *.keyCode* using *String.fromCharCode()* for alphanumeric keys and a switch statement to produce readable output for others, such as converting arrow keystroke codes to "[LEFT]", "[RIGHT]", etc. The function also queues up keystrokes in the same control to be recorded as paradata only when the focus changes. This is done by comparing both the *.parentNode.id* property of the HiddenStatus control and the *.name* property of the visible control to determine whether focus has shifted. The result is that most text answers will appear as a single line in the paradata rather than as multiple lines with single keystrokes. This function is called with onkeyup events added to the xsl for input and textarea controls in the following manner: *<xsl:attribute name="onkeyup">processKey(this, document.f.<xsl:value-of select="$Question/@ID"/>st, event);</xsl:attribute>*. It then calls the recording function when a set of keystrokes is complete.

The function that handles scroll events is responsible for setting two boolean values that indicate whether horizontal or vertical scrolling occurred since the last recorded event. These values are used in the recording function as described above. The function is simply called from an onscroll event attached to the formpane div as follows: *<xsl:attribute name="onscroll">scrollEvent(this)</xsl:attribute>*. The function just compares *.scrollLeft* and *.scrollTop* to their values at the previous onscroll event and sets the two boolean values accordingly. It is possible to gather much more information than a pair of Boolean values out of these functions. For example, it is possible to record the maximum extents of scrolling on each axis between events, or even to attempt to distinguish separate scroll events. However, at this point, our methodologists are mainly interested in whether scrolling occurred at all and on which axis.

A final function records hyperlink clicks on the page. The function itself is straightforward – it simply records elapsed time and the url on a new line. At the moment, we are only collecting hyperlink clicks associated with HelpImage, using a function call at the onclick event like this: *<xsl:attribute name="onclick">recordHyperlinkClick('<xsl:value-of select="$URL"/>');</xsl:attribute>*.

Conclusions

Overall, while taking into account the inherent difficulties presented by web surveys, the methods described here would appear to allow us roughly parallel capabilities between CATI/CAPI and CAWI. It may be some time before ISR begins to field enough Blaise IS surveys to start to refine these server- and client-side paradata collection methods, but we anticipate that they will provide a strong foundation for us going forward.

Appendix: functions added to biEvtHan.js

```
var beginTime = new Date();
var lastField = '';
var lastName = '';
var scrollx = 0;
var scrolly = 0;
var endScroll = 0;
var scrollEventHoriz = false;
var scrollEventVert = false;

function processKey(AnswerField,HiddenStatus,e){
        var evt=window.event? event : e;
        var unicode=evt.charCode? evt.charCode : evt.keyCode;
        var key;
        switch(unicode)
        {
        case 8:
          key='[BACK]';
          break;
        case 9:
          key='[TAB]';
          break;
        case 13:
          key='[ENTR]';
          break;
        case 16:
          key='[SHFT]';
          break;
        case 17:
          key='[CTRL]';
          break;
        case 18:
          key='[ALT]';
          break;
        case 19:
          key='[PAUSE]';
          break;
        case 20:
          key='[CAPS]';
          break;
        case 27:
          key='[ESC]';
          break;
        case 33:
          key='[PGUP]';
          break;
        case 34:
          key='[PGDN]';
          break;
        case 35:
          key='[END]';
```

```
            break;
          case 36:
            key='[HOME]';
            break;
          case 37:
            key='[LEFT]';
            break;
          case 38:
            key='[UP]';
            break;
          case 39:
            key='[RIGHT]';
            break;
          case 40:
            key='[DOWN]';
            break;
          case 45:
            key='[INS]';
            break;
          case 46:
            key='[DEL]';
            break;
          default:
            key=String.fromCharCode(unicode);
          }
          if(lastField==HiddenStatus.parentNode.id && lastName==AnswerField.name){
                  document.f.paraKeyStrokeStr.value += key;
          }else{
                  recordClientSideParadata(HiddenStatus,AnswerField,key);
          }

}

function recordClientSideParadata(hiddenRef,displayRef,responseValue){
          //roughly follows the format of Dirk Heerwegh's CSP JavaScript Version 2.0 for comparability
- see https://perswww.kuleuven.be/~u0034437/public/csp.htm
          var timeEvent = new Date();
          var timeLapse = timeEvent - beginTime;
          beginTime = timeEvent;
          if(scrollEventHoriz == true || scrollEventVert == true){
                  document.f.paraKeyStrokeStr.value += '^t=' + endScroll + ':endScroll(' + "x=" +
scrollEventHoriz + ",y=" + scrollEventVert + ')';
                  scrollEventHoriz = false;
                  scrollEventVert = false;
                  timeLapse -= endScroll;
          }
          document.f.paraKeyStrokeStr.value += '^t=' + timeLapse + ':' + hiddenRef.parentNode.id + '='
+ responseValue;
          lastField=hiddenRef.parentNode.id;
          lastName=displayRef.name;
}
```

```
function scrollEvent(ref) {
	var timeEvent = new Date();
	var timeLapse = timeEvent - beginTime;
	endScroll = timeLapse;
	if(scrollx != ref.scrollLeft){
		scrollEventHoriz = true;
		scrollx=ref.scrollLeft;
	}
	if(scrolly != ref.scrollTop){
		scrollEventVert = true;
		scrolly=ref.scrollTop;
	}
}

function recordHyperlinkClick(url) {
	var timeEvent = new Date();
	var timeLapse = timeEvent - beginTime;
	beginTime = timeEvent;
	document.f.paraKeyStrokeStr.value += '^t=' + timeLapse + ':target=' + url;
	lastField = '';
	lastName = '';
}
```

C3B: Exploiting The Numerous Possibilities Web Technology Offers To Elevate Questions

Arnaud Wijnant/Maurice Martens - CentERdata, Tilburg University, the Netherlands

Abstract

BlaiseIS provides the possibility to administer questionnaires over the Internet. A next step would be to exploit the numerous possibilities web technology offers to elevate questions, share data sources or create extensive statistics like paradata. This can be achieved by creating special web applications that collect data and return a set of answers to the Blaise engine based on the collected data. These web applications are called custom questions.

A problem, however, is that the development of these custom questions requires different skills than programming a questionnaire in Blaise. For this reason the Custom Questionnaire Control for Blaise (C3B) was created. C3B is a mechanism which has two sides: one side which can be used by a questionnaire designer to define custom questions in a BlaiseIS questionnaire, the other side can be used by web developers to create a context-aware web application which can easily communicate with the Blaise IS engine. C3B takes care of the mappings between these two sides.

The web developer is free to choose between different development platforms (like, for example, JAVA, .NET, ASP or PHP), different techniques (like, for example, JavaScript, AJAX) and even different servers where the custom question can run on.

The questionnaire designer can easily parameterize custom questions and share them within different questionnaires, projects or even organizations.

At CentERdata, several C3B-compatible custom questions have been created. Examples of these custom questions include question-to-speech, clicking and dragging images in a certain order and auto-completion of open answers. The number of available custom questions can increase quickly since the development of a custom question is not complicated and the web offers many possibilities.

1. Introduction

CentERdata is a research institute, specializing in online survey research. CentERdata is affiliated with Tilburg University, the Netherlands. One of CentERdata's most important activities is to carry out panel surveys through online panels: the CentERpanel and the LISS panel. Both panels consist of several thousand households in the Netherlands. The members of those households complete questionnaires on a regular basis, weekly or monthly.

CentERdata started using Blaise in 2000. We intended to do so for our online surveys as well, but had to develop a software package to mold it to our needs: C2B (Weerman, 2001). With the recent improvements and needs in web technology we decided that C2B does not meet our requirements any more, and rather than redesigning C2B we decided to switch to BlaiseIS.

At CentERdata, we appreciate the advances BlaiseIS has made over the years. However, for our purposes one key feature is still missing. Sometimes, an online experiment has to be carried out that involves some advanced functionality, be it some javascript drag and drop experiment, a widget that tracks mouse movements, a complex layout scheme or it should use extended media like movies or sound. We refer to these enriched advanced applications as web technologies.

Compared to Standard Blaise with C2B, BlaiseIS offers a more structured way of separating the questionnaire definition and presentation of the questionnaire. However, only some basic Blaise question types can be defined, which makes it impossible to have "advanced" questions without altering the program code again. The decision was made to create a more generic way to include advanced questions in BlaiseIS questionnaires. This was done by creating the Custom Questionnaire Control for Blaise (C3B). The requirements for C3B are the following:

- Questionnaire designers should be able to include custom questions in a questionnaire without the help of a programmer
- Programmers should be able to design web applications independent of BlaiseIS
- Custom questionnaires should be reusable
- No adjustment of the BlaiseIS code should be necessary to include a custom question

This paper describes how we managed to create C3B, the mechanism that managed to fulfill the requirements. Also, some examples are given that show how C3B can be used to enrich BlaiseIS questionnaires.

1.1 BlaiseIS Custom Questions

In Blaise it is possible to define question types like open questions, multiple choice questions and set-of questions. BlaiseIS transforms these different question types to a suitable HTML form which enables respondents to see a web page which looks similar to the DEP representation of a questionnaire.

On the Internet there are many more possibilities to create non-standard (custom) ways of asking questions or gather information. Some examples are:

Interactive questions
Questions that give feedback to the respondent while the respondent answers a question. A good example is auto completion. Auto completion tries to complete the text in an open answer box while the respondent is typing.

Alternative user-input questions
Respondents can use other types of input mechanisms. For example, clicking on areas in an image to answer a specific location on a map, dragging objects to locations on the screen or spoken question texts.

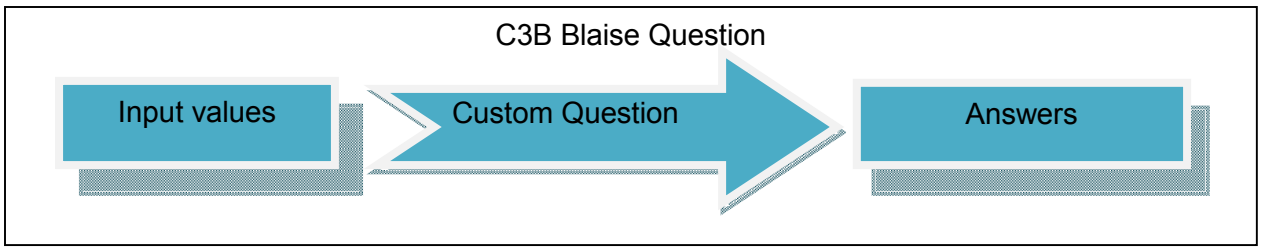
On other data source relying questions
Another type of custom question is a question where the user does not need to answer a question directly, but the answer is derived from an external data source on the web. Examples are Social Media, Google maps.

In this paper, we refer to all these types of questions as custom questions.

1.2 Integrating custom questions in Blaise

Questionnaire designers should consider a C3B question as a box which can be plugged into a questionnaire. The questionnaire designer should not need to know how a custom question is programmed. He should only look at what input values are needed and what type of answer can be expected from the C3B-question. In general, a C3B question can be modeled as in Figure 1.

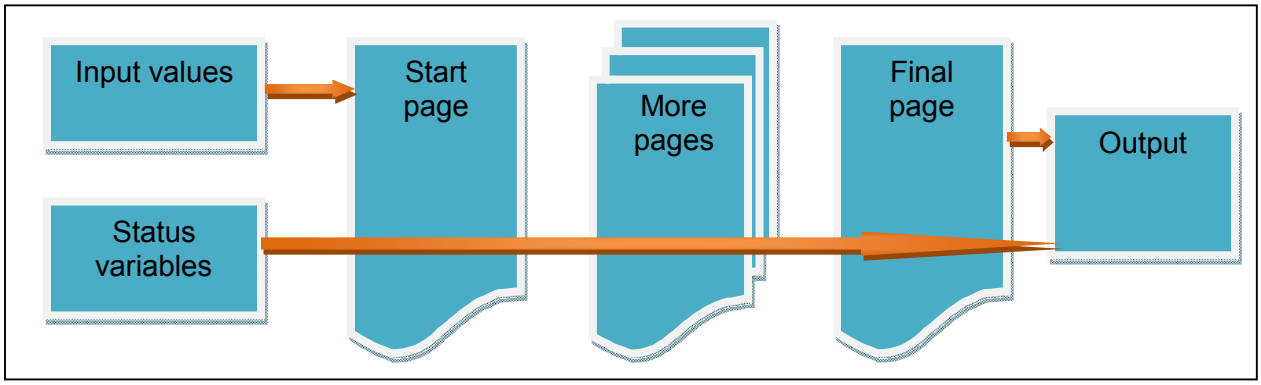
Figure 1. High level model of a C3B question



This model enables the questionnaire designer to adapt C3B custom questions easily by changing the input values. In addition, it enables the reuse of the same C3B custom question for several questionnaires or several questions within a questionnaire.

For the web programmer a C3B question looks like Figure 2.

Figure 2. C3B from a web programmers' perspective.



The input values in this model are the input values as they are defined in the questionnaire design. Together with the output, the input values take care of the communication between the questionnaire and the web application. The status variables are BlaiseIS specific status variables that will be created by the C3B software (in association with BlaiseIS). The C3B component should return the values of these variables at the end of the custom question. What happens between the start page and the final page is completely up to the programmer (even if there is a page in between, it could also be possible that the start page and final page are the same web page).

2. The process

C3B is a component between BlaiseIS and web applications. The C3B component collects data from two different sources and sends it to the web application:

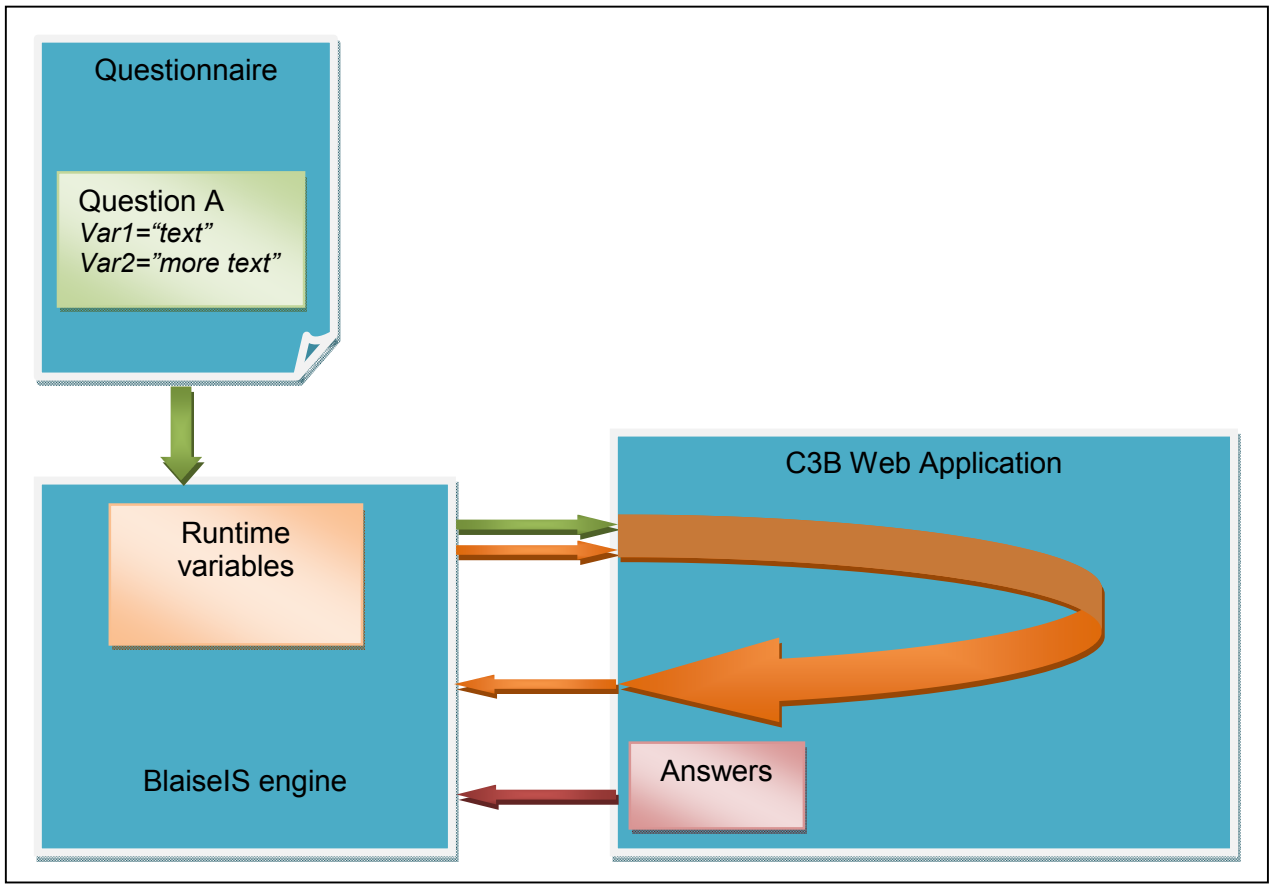
1. Variables and values as defined in the questionnaire
2. Variables and values generated by the BlaiseIS engine

An example of the first type of variable is a question text. The questionnaire designer can define a question text C3B variable and parameterize the custom question with a variable question text.

The second category contains interesting variables too. For example, the return page URL for the last page in the custom question sequence. Or the variable names of the Blaise questions and what name should be given to these variables upon the last postback action in the custom question.

An overview of how the different variables are combined with each other is shown in Figure 3.

Figure 3. The communication between the different C3B components



The C3B web application is the part of the custom question that is programmed by a computer or web programmer. There are some guidelines for the web application to become C3B compatible. These guidelines mainly consist of variables that need to be passed on or changed during the custom question web application. In Table 1, a list of these variables and what action should be done with them is displayed.

Table 1. List of the C3B web application input variables

Input variable	Action
sh	Status variable needed for the output variable "sh"
lsi	Status variable needed for the output variable "lsi"
pli	Status variable needed for the output variable "pli"
spi	Status variable needed for the output variable "spi"
aqi	Status variable needed for the output variable "aqi"
cqi	Status variable needed for the output variable "cqi"
KeyValue	Status variable needed for the output variable "KeyValue"
InterviewID	Status variable needed for the output variable "InterviewID"
Lmr	Status variable needed for the output variable "Lmr"
statusvarnameX	For x=1 to N do create an output variable that is named "*statusvarnameX*" and has the value "*statusvarvalueX*"
statusvarvalueX	See statusvarnameX
varnameX	For x=1 to N do return value for Blaise variable "*varblaiseVariableX*" as output variable named "*varnameX*" the previous value for this variable is called "*varvalueX*"
varvalueX	See varnameX
varblaiseVariableX	See varnameX
nextvarname	This is the name of the "next button" to go to the next question in the questionnaire, for example: <input type="submit" name="*nextvarname*">
previousvarname	This is the name of the "previous button" to go to the previous question in the questionnaire, for example: <input type="submit" name="*previousvarname*">
returnpage	The URL of the page where the final page should submit the answers/variables to

These variables with their values will be passed to the C3B web application by a HTTP post-action. This post-action is initialized by the browser of the respondent. Because the HTTP post-action is a very general web mechanism, some flexibility is created for the programmer. Not only is the programmer free to create the web application in the preferred web programming language (like, for example PHP, ASP,

JAVA, .NET), but also the location of the web application is not restricted with C3B. Especially this last advantage is useful to link a questionnaire to other data sources on the web.

One should however note that some browsers have their security options configured to display warnings when a page posts to a different server.

A similar variable specification exists for the questionnaire designer. These variables are listed in Table 2.

Table 2. List of the C3B questionnaire output variables

Output variable	Description
url	The URL of the C3B web application start page
serveraddress	The address of the server where this questionnaire will be loaded (NOTE: only necessary for javascript disabled browsers)
[any other variable name]	The questionnaire designer is free to define new variables, these will be posted as input variables for the C3B web application

A special C3B construction can be used to specify these variables. This C3B construction looks like a Blaise question of the type "string , empty". But there are two differences:

1. The question text is preceded by "(customapplication)"
2. The question text contains the variables and values in the following form:
    - *variablename1*;value1;*variablename2*;value2;*variablenameX*;valueX;

In Figure 4 an example is given of how to specify these variables in a Blaise questionnaire. In this example question v1 has the output variables and values specified as mentioned in Table 3.

Table 3. The C3B output variables and values used in the first question of the Figure 4 example

Output variable	Value
url	http://.../question.php
text	Deliver meals in a nursing home
audionumber	161

Figure 4. An example of a Blaise Questionnaire that uses C3B

```
DATAMODEL siwittest
LANGUAGES = NED "Nederlands"
PRIMARY
nohouse,
nomem
FIELDS
      nohouse: 0..100000
      nomem: 0..500000
v1  (customapplication)
"url;http://cdata4.uvt.nl/blaiseistest/question.php;text;Deliver meals
in a nursing home;audionumber;161;":  string, empty
c1 (hidden) "Deliver meals in a nursing home" : string
v2  (customapplication)
"url;http://cdata4.uvt.nl/blaiseistest/question.php;text;Decorate
tables in a building;audionumber;178;":    string, empty
c2 (hidden) "Decorate tables in a building" : string
v3  (customapplication)
"url;http://cdata4.uvt.nl/blaiseistest/question.php;text;Fix
computers;audionumber;189;":    string, empty
c3 (hidden) "Fix computers" : string
LAYOUT
      AFTER c1 NEWPAGE
      AFTER c2 NEWPAGE
      AFTER c3 NEWPAGE
RULES
      nohouse.keep
      nomem.keep
      v1
      c1
      v2
      c2
      v3
      c3
ENDMODEL
```

Please note that several variables can be part of one C3B custom question. Blaise needs to know that these variables belong to each other. This can be done by ensuring that all variables belonging to one C3B custom question are on one page (by using the NEWPAGE command in the layout section of the questionnaire).

3. Results

At CentERdata some C3B custom questions are already in use. It turned out that C3B is not only helpful to create custom questions. We also found out that complex layouts of tables in questions are sometimes easier to be generated by a C3B component than with the standard BlaiseIS functionality.

The number of variables within one C3B question varied from 1 to 16 (for example, the complex table layout contained up to 16 questions to be filled in by the respondent).

Respondents did not encounter any problems when using questionnaires with C3B.

3.1 Example

A good example of what is possible in C3B is the job interests test. This is a questionnaire that is used to advise students with some kind of a disability about what type of job would suit them best.

There are special requirements for the questionnaire since the group of respondents is not able to complete a normal questionnaire. These requirements are:

- Since not all students can read, the question should be read out by the system and the answers "like" and "don't like" should be simple and clear images
- Since not all students understand what is meant by a profession, a question must be accompanied by a picture that shows what is done in this profession

Together with a design team the interface as shown in Figure 5 was created and programmed in PHP and Flash. Next to the visual interface the question text and answer categories are read out as soon as the question is shown on the screen.

Figure 5. A screenshot of a C3B custom question

To create a questionnaire that uses several questions of this type, a questionnaire similar to the example mentioned in Chapter 2 of this paper is used. The web programmers created a web application which used two additional parameters:

- text: the text that must be shown for this question
- audio number: an identifier for both an image and audio file per question

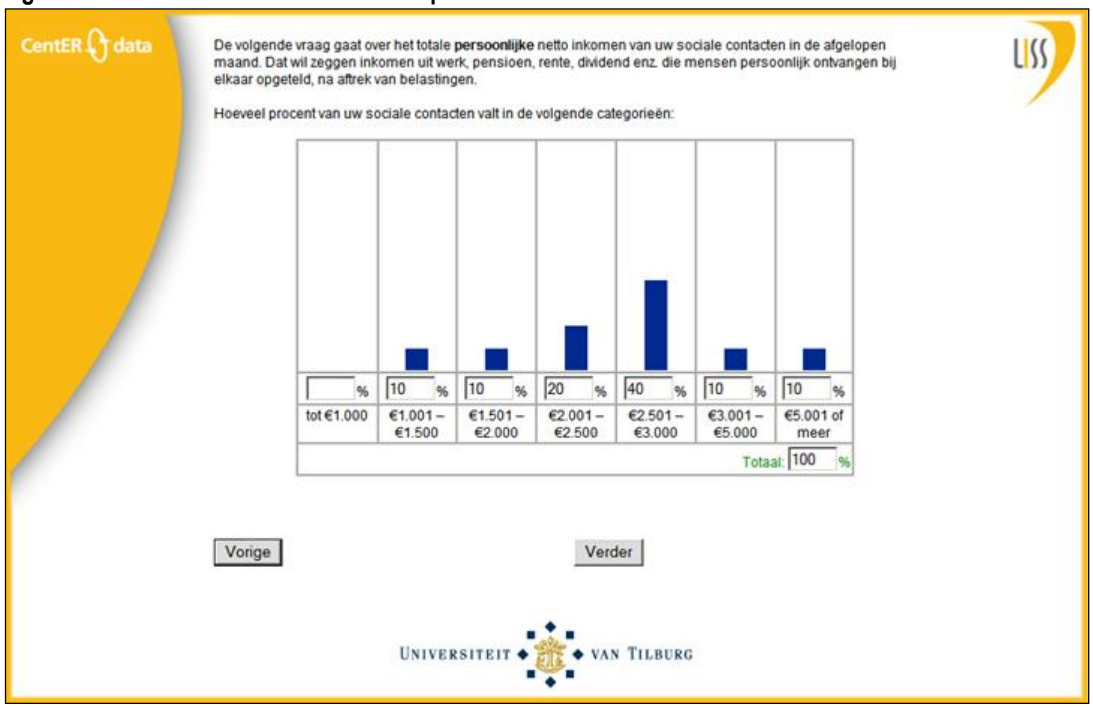
If both parameters are defined for a question, C3B and the web application take care of everything. For the questionnaire designer the questionnaire is not different from a normal BlaiseIS questionnaire with the exception that there are a few additional variables that are used to define which C3B application should be used.

3.2 Javascript examples

The C3B method has already been used in various advanced online questionnaire applications where it helped visualising percentages. When asked about mathematical or economic concepts like chances or percentages, respondents might not completely comprehend. To help them, several instruments were developed to provide a better intuitive presentation of these concepts.

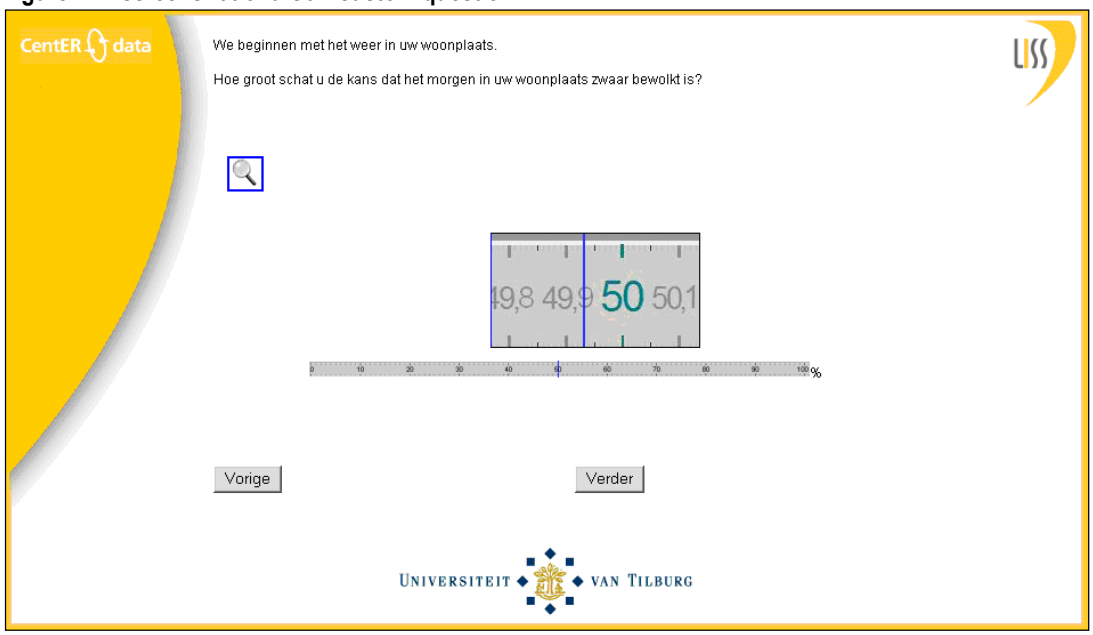
For example Figure 6 shows a screen that asks respondents to divide the persons in their social network into income groups. A javascript application was developed that visualized percentages. The bars change height when a respondent inserts integer values for each group.

Figure 6. A screenshot of a C3B custom question



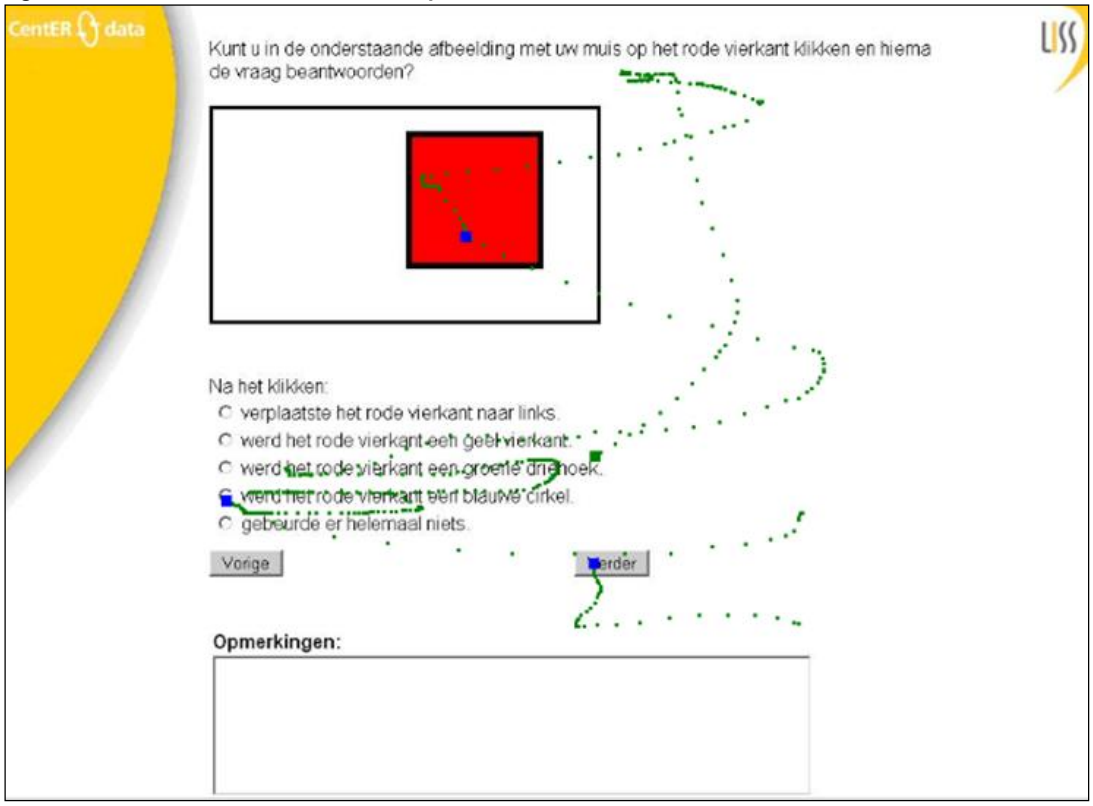
Another example is a project where respondents were asked about chances of events happening in their life, for example the chance of falling ill in the near future. To allow respondents to insert answers more significantly a scale was developed. Respondents could drag on the complete scale and specify their answers on the zoomed in window.

Figure 7. A screenshot of a C3B custom question



Javascript can also be used to check respondents' behavior. Respondents were asked to click on the red square, once clicked, this square automatically changed into a blue circle. The respondent was asked what happened. In the meantime, all mouse movements and clicks were recorded. This could be used to monitor respondent behavior, but also to test interface design. In Figure 8 a respondent's tracked mouse locations (green) and clicks (blue) are displayed over the original screen.

Figure 8. A screenshot of a C3B custom question



Other javascript examples that were developed include changing interface styles for certain groups of respondents, use of calendars, visual manipulation of graphs and puzzles.

3.3 Future development

There are several other new possibilities to create C3B custom questions. For example, social networks contain lots of data about respondents. C3B could help web programmers to link Blaise questionnaires to these data sources. Interface specifications are easier to be created, which makes it possible to create questions that measure actions of a respondent. We could for example track how a puzzle or other complex procedure was solved by a respondent. We can actually measure the respondent's actions.

Sharing components is another advantage of C3B. Reuse of C3B questions is easy and does not need involvement of a web programmer. If more projects use C3B in combination with BlaiseIS, a library of several innovative ways of asking questions in a web questionnaire could be created.

We believe improvements are still possible on the C3B interface side. At this moment we still learn from practice. For example the input and output variables can change or be expanded if questions need more

context information. Currently we think about error states and error messages and what would be the best way to provide them to the web application.

C3B is a new way of coping with web questionnaires which has lots of potential on both the improvement of the mechanism and on the usage. Its mechanism of stepping out of Blaise and returning back to it gives us the freedom to develop online questionnaires with the latest web based technologies without being hampered by third party development.

4 References

Weerman, Bas, Internet interviewing using Blaise API, IBUC 2001

Moving to Blaise IS

Peter Sparks and Gina-Qian Cheung, Survey Research Center, University of Michigan

1. Introduction

As the internet becomes ever more prominent the need to conduct web surveys increases. Blaise has had internet support since 4.6, but Michigan has only recently started using Blaise IS in Blaise 4.8. This paper will look at Michigan's experiences in implementing Blaise IS in their production environment, the difficulties encountered and the solutions made. Topics covered include lessons learned in following areas: written specifications, programming, server set up, data storage choices, case management, paradata capture, data export, and remaining challenges.

2. Initial Project Requirements

The initial scope of a large multi-mode survey would require use of CATI, CAPI, and self-administered web. The plan was for the survey content to change frequently as data collected and analyzed would help refine the survey. Because of the rapid and constant change aspect of the study it was decided to use one source code and prepare directives to produce the needed different datamodels needed for the different modes. The goal was to minimize the logic flow and question content discrepancies that happen when working across different instruments.

Another requirement was that data needed to be secure, and so the choice was to use Blaise's relative BOI files via Datalink and store the actual data on a secure data server for the web mode. SQL Server 2008 was chosen as the database platform. It allowed for login management as well as ease of queries against the tables, robustness of the data, and so forth.

Without a built-in case management system, our use of Blaise IS meant integration with an existing system. We chose to integrate it with SurveyTrak (distributed phone interviewing) because of the built-in reporting features. The ad hoc nature of web data collection lent itself better to CAPI than CATI.

Given the short time frame to actually implement this large and complex survey we wanted to use existing utilities and reports whenever possible. To this end we made sure the data that was captured could be exported to formats that our utilities already used.

The final requirement was to produce printed documentation that reflected the web survey itself, along with similar documentation for the other modes.

3. Challenge - Written Specifications

Needless to say, even though we had a plan for starting such a project, there were pre-requirements that we hadn't anticipated. Because of the years Michigan has spent developing and programming CATI/CAPI surveys, we have solid guidelines for how to write specifications for a survey, and how to program the survey in Blaise. However, we didn't have either for the web mode.

Hence, at the start of this process there were no written specifications, and no standards for programming for Blaise IS. The areas to be developed for these specifications included:

- Allowable controls on the screen
- Placement of the controls
- Size, color, font of the controls and text

- Background color
- Stem content and color
- Navigation behavior
- Menu items
- Buttons (placement, size, color, ...)
- Other actions (DK/RF, external lookup, consistency checks)

4. Challenge – Programming

Programming for web is very different than the effort involved for CATI/CAPI. The basic structure to the datamodel remains the same, but all the look and feel changes because of the different mode. Decisions were made tailored for the Michigan's data collection environment.

4.1 Global settings

We followed the method set by another survey package, Illume, to determine DK/RF actions. If a question is skipped then that question is assumed to have an RF response. The DK option is only allowed on particular questions and is shown explicitly. All questions are allowed to be EMPTY, whereas CATI/CAPI most questions are the opposite.

4.2 Rules changes

The programming had to be tailored to include the additional auxfields used as stem text for pages with grids. More of the items are presented in tables instead of just a single question at a time.

4.3 Question Text

The wording had to reflect a self-administered interview: layout, color, font, size, readability.

4.4 Multimedia

Multimedia could be an option (videos, sound), but the pilot survey done by Michigan, Sunflower, did not use additional multimedia.

4.5 Modelib layouts

There is a complex interaction among the information contained in the question text, what layout is going to be presented via the LAYOUT section, the actual layout itself in the modelib, how the screen in general is going to look within the menu file, and the grouping mechanism. The modelib plays a critical role in the appearance of the survey and takes more effort to use effectively than a comparable CATI/CAPI survey. The challenge was to learn how to use all these pieces efficiently.

4.6 Menu files (panels, buttons, actions)

The menu has a complexity to its arrangement, and there was a learning curve to understand how the pieces related. For example, the different panes on a page, and especially how to center text, place the navigation buttons, color the different background areas, incorporate hyperlinks and text, and call different functions from the menu.

4.7 ASP files

We needed deeper journal information (paradata), so that meant modification of the ASP files. We had to modify pages to log into the survey, log additional information to the server, and other tasks.

4.8 Stylesheet customizations

We found that we needed to modify the default stylesheets in order to gain more control over the HTML generated to make them more generic for different browsers. We found that by modifying some attributes

within the stylesheet we could solve some scrolling problems, although others, such as table/cell border outlines and centering, were still a problem.

4.9 Grouping (layout, field & label selection)

The grouping editor in the Blaise Internet Specifications file proved to be a challenge because of the editor. We found that small changes in the question (code frames, renaming questions) could invalidate a group and cause the grouping work to be redone.

4.10 Critical questions

Learning about client/server communication and when a question was required took additional time during testing. On-screen fills, hiding/showing questions, and so forth require setting this for a question. This is an additional step in the programming not present in CATI/CAPI.

5. Challenge – Servers

In order to deploy a Blaise IS survey, servers are involved. Blaise IS uses a minimum of three server roles (rules, data, and web), and allows multiple rules and web servers but only one data server. The arrangements can be one server or many physical servers. The entire set of server/service roles was a challenge to set up.

5.1. Blaise IS Ports

Working with the computing support staff at Michigan, ports on different servers were opened until the right combination worked with the firewalls we had in place. The typical symptom of a port not opened is the utter lack of response when using the Internet Server Manager, or some other error when connecting. This step took much more time than expected.

In addition, these ports need to be open for communication with the web server:

- **Firewall (optional)** should have the following ports open:
    - 80 for http
    - 443 for https (Secure Socket Layer) if desired

5.2. Blaise IS Services

The challenge was to set up these servers so they could communicate with each other through the Blaise services (API, CATI, Survey Manager, Portal, Registry, Server Park, and Service Monitor). A good reference of what services are needed for each server role is found in the Blaise Help: Control Centre - Tools - Blaise Services - Type of Blaise services. The roles of Blaise services and ports are found in "Requirements for conducting surveys."

We estimated that we eventually would need two data servers for the large project, but just used one separate data server and one web/rules server for the Sunflower project.

Installation of the Blaise software was required in different locations: the developer's machine (internet installation, full), the web server (API, Survey Manager, Portal, Server Park), the data server (API, Survey Manager), and the rules server (API, Survey Manager). We did not install CATI on any of the Blaise IS servers.

Users needed enough rights to get to the web server, but once there, the services on the server handled the other communication details & access rights. Hence, the user could never have direct access to the data because it was stored on the data server, and the data was transferred via the Blaise services.

We had assumed that we would need to create our own relative BOI file -- but Blaise (4.8.1) automatically creates the directory structure and copies the appropriate files to the data server, in addition to the rules server. This took the guesswork out of the installation.

6. Challenge – Data Storage

The most obvious choice for a data storage format for Blaise IS would simply be to use a BDB format. It would allow instant access to the data through use of existing utilities, and with use of a data server and the implied relative BOI would remain secure behind a firewall. However, there are additional features of SQL that were attractive that influence the choice.

6.1 Data Server

A separate data server was used to store the Blaise data apart from the web/rules server, and required different user logins & ports opened.

6.2 Data Roots

Blaise allows definitions of additional directories to be stored in a list of data roots to be used with BOI files. At the beginning of the process the need of data roots were unknown and somewhat confusing.

6.3 SQL backend

In preparation for the large study and knowing there was going to be heavy use of the database from both users and analysts, Datalink with SQL was a logical choice over a BDB. Blaise inherently uses datalink, and so setting up a BOI file for data use means the most flexibility in working with the data.

Using a SQL database also allowed us to use three different "databases" within the database: main interview data, working data, and paradata. This kept management of the files tidy and secure.

6.4 BOI data structure (Stream, Flat No Blocks) - Datalink

We had to examine the different layouts available and choose a BOI format that worked for our needs. Two different formats to the SQL database were chosen for this study: stream and "flat, no blocks." The former was used for the main and working databases, while the latter was used for the paradata. The stream format was chosen for its compactness and speed. It cannot be queried directly via SQL, but through BOI files it can be easily exported to another format. The flat format was used for the paradata to allow it to be queried as needed, and in fact the tables created were used to store timing information back to the sample management system, SurveyTrak.

6.5 Minimize number of tables (Main, Working, Paradata)

To minimize the amount of overhead, each study published in Blaise IS was assigned a database within SQL. This allowed all the associated tables (main, working, and paradata) to be placed in one area. Although generic tables could have been chosen, we chose for this test to let the default configuration remain.

6.6 Username/password login to tables

A study-specific login (username, password) to the SQL database was created. This provided an extra level of security to the survey data. The original BOI file is password protected (via Blaise) to prevent revealing the database login.

6.7 Datamodel changes/migration

We made plans for migrating the data during production if needed. From our testing it seemed like the best plan would involve the following:

1. Update the datamodels
2. Create new BOI files that match the updated datamodels
3. Suspend the survey
4. Run an OLEDB (old datamodel) to Blaise (old datamodel) to save the data
5. Delete the survey
6. Install the new survey
7. Run a Blaise (old datamodel) to OLEDB (new datamodel)
8. Start the new survey

6.8 Database "locks" – hangs

During development we encountered locks on the database. Although we never tracked down exactly the cause, this seemed to be caused when we deleted/reinstalled a survey through the Survey Manager.

We found out that most often the process responsible for causing the lock was the Blaise API service found on the data server. We tried a variety of ways to release the deadlock that stopped the survey from responding, including rebooting the machine that was hosting the data server and reinstalling Blaise. We found that by stopping and restarting (and sometimes killing and starting) the service we could regain control. A simple solution of just waiting sometimes worked. We did find that once production began and we stopped uninstalling the survey we experienced no other locks.

7. Challenge – Case Management

Blaise IS provides a great base for creating web surveys, but does not provide the management piece. This includes the tracking and management of sample, status reports, and holding/releasing/preloading cases into the system, as well as sending emails to respondents.

7.1 Existing system (CAPI) integration

SurveyTrak is the sample management system used for our decentralized interviewing system. It has, throughout the years, had many management tools built into the system. We decided to use existing features in SurveyTrak and add a few more functions outside of SurveyTrak.

7.2 Email jobs

Email is an important way to contact the respondents, but unfortunately Blaise IS does not provide a system to do this. However, a new module was written to handle email jobs (see the conference paper, "Blaise IS Sample Management."[1]).

7.3 Suspend and resume interviews

The respondent was required to log into the survey (after being provided the required information in an email). Upon resuming an interview the respondent was taken back to the place they last suspended.

7.4 Reports

Blaise IS's Internet Manager provides very basic reports on the number of completes and the number of times the survey has been accessed, but other reports needed to be customized. These were created in part from the email utility, and others from integrating with SurveyTrak.

7.5 Daily updates to SurveyTrak

Since SurveyTrak was used to supply sample to the study and run reports, daily updates of paradata information from the paradata table to SurveyTrak was needed. Information stored to SurveyTrak included the time spent by the respondent in the interview, the last question answered, and the number of resumes.

8. Challenge – Paradata

We needed detailed journal information -- much more than the default Blaise IS journal:

Field Name	Type	Description
Date	DATETYPE	Date of the event
Time	TIMETYPE	Time of the event
PrimaryKeyValue	STRING	Primary key value of the interview

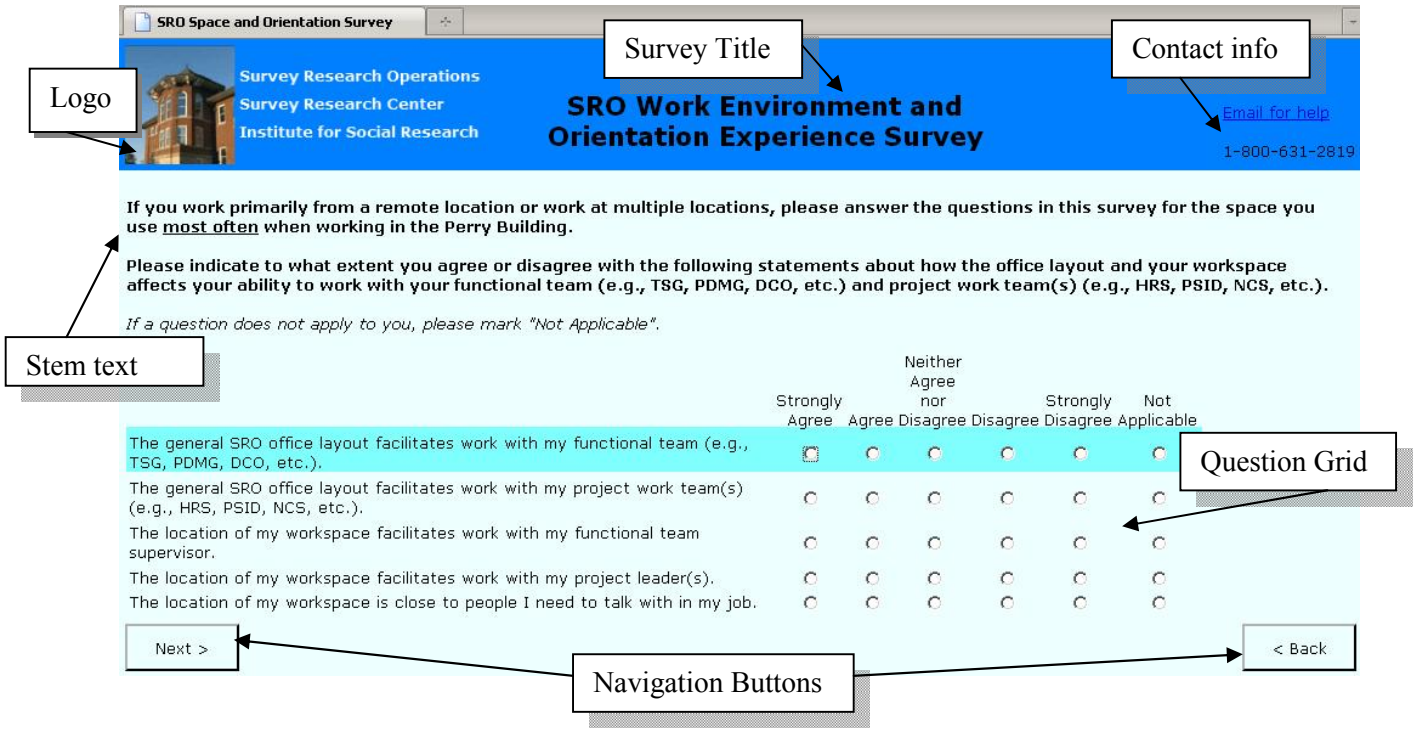
-- Blaise Help - Monitoring - Web journal for programmers

The datamodel that describes the journal can be extended within the datamodel, but we needed to store information outside the interview. These fields were populated using the menu file via javascript and ASP code. This is detailed in the conference paper, "Blaise IS Paradata."[2] The data defined in the paradata (journal) datamodel are stored in an SQL table using the Datalink "flat, no blocks" format.

9. Solution - Written Specifications

Although it would have been nice to have a lengthy time to research all the best web designs and implement them fully within Blaise IS, time was always of the essence. That is, the "specs" were tailored to the current need and limitations within Blaise IS as the instrument was being developed. There was difficulty in determining the best method of describing the layout of a screen, and eventually a simple set of markers in the spec document was used.

The markers described the start of a screen, and then the start of a grid or other structure. Any text that appeared before the start of the grid would become "stem" text that required additional programming.



General layout elements, such as the logo, survey title, help reference, button navigation, background color, highlight color, font size and font type, and so forth were initially chosen, but revised as the project went forwards.

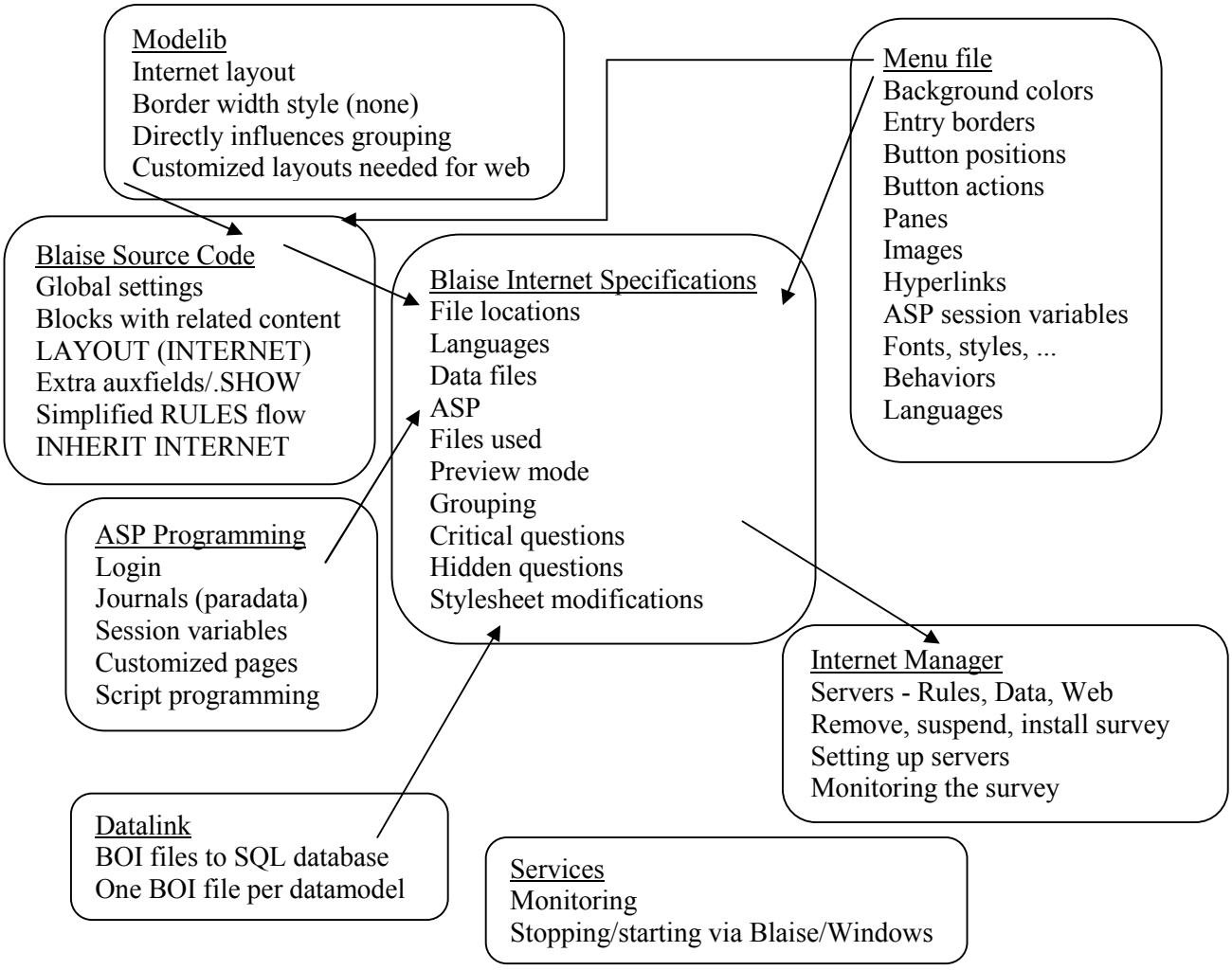
Because web surveys are self-administered, close attention was paid to making the question text distinct from the answer categories. In the above screenshot, the question text is in bold, instructions specific to this page to the respondent are in italics, and the question category topics are in a normal font.

There are areas that have not been decided. These include the use of tables, lookup lists, and other user controls (drop down lists, date and time pickers, custom controls, if ever to show DK/RF values, and so forth). Therefore we expect the specifications to change as we become more versed in web studies.

10. Solution – Programming

Necessity is the mother of invention -- and that's no tall tale for entry into web survey programming. The complexity of going into this area is about the same level of complexity as learning the CATI call scheduler and setting up related systems.

A big necessity was simply learning about the different areas and how they related to one another. In CATI this process seems fairly straight-forward, with the complexity coming from the INHERIT CATI command and the call scheduler parameters (ignoring customizations to the system via manipula). However, in Blaise IS, the diagram becomes more complex because there are so many interrelated parts.



10.1 Appearance Specs Required

Programming in the Blaise instrument is influenced by the layout of the web page, so realistically the general layout of the survey needs to be specified first (and programmed in the menu file). The specific design of each web page also needs to be determined early to help determine any "stem" text for the page, as well as any needs for grouping questions (grid, table, amount/per, or other format).

10.2 "Stem" text

In order for question text to appear before a grid, an auxfield is created. It is placed in the rules with a .SHOW attribute, and a layout that displays the question text only (via the modelib). In this example, the questions in the grid are grouped together via the Blaise IS grouping editor and via the modelib, and the NEWPAGE command is used start a new web page.

```
FIELDS
   xA1 "@/@T@BIf you work primarily from a remote location or work at multiple locations,
       please answer the questions in this survey for the space you use @Umost often@U
       when working in the Perry Building.@B@T

       @/@/@T@BPlease indicate to what extent you agree or disagree with the following
       statements about how the office layout and your workspace affects your
       ability to work with your functional team (e.g., TSG, PDMG, DCO, etc.)
       and project work team(s) (e.g., HRS, PSID, NCS, etc.).@B@T

       @/@/@IIf a question does not apply to you, please mark ""Not Applicable"".@I" :

      TContinue

RULES
   xA1.SHOW
   A1a
   A1b
   A1c
   A1d
   A1e

LAYOUT (INTERNET)
   BEFORE xA1 NEWPAGE
   AT xA1 FIELDPANE QuestionTextOnly
   FROM A1a TO A1e FIELDPANE GroupTableQuestionTextLeft
```

10.3 Menu modifications

One frustrating thing about working with the panel and placement of controls on the menus is that the dialogs work with pixels, but realistically it's a proportion of available space (height and width) on a web page. Placement of an item is relative to the widths of all the items.

We found that the background color of an object will carry through to the controls below it. So if a panel is colored green, then all the things on the panel also get green unless it's reset.

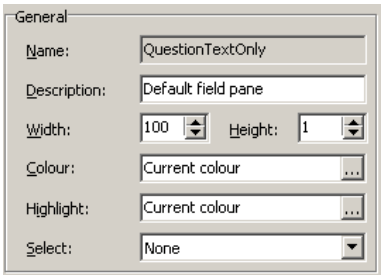
We also recently added a variable to the menu for our internal CAI Testing Tool utility to have it work with Blaise IS surveys. The use of this variable may be discussed in the paper, "BlaiseIS Paradata."[2] An earlier paper on the CTT was given in "CTT: CAI Testing Tool."[3]

10.4 Variables

Programming for a Blaise IS web page is more intensive than for CATI, and will carry more auxfields. It's recommended that programmers use a standard naming convention for the additional auxfields, such as xFieldname when its purpose is introduction text to a page.

10.5 Layouts

Blaise Internet requires a modelib specific for that mode. Within that mode there will be a number of default field panes for internet. We found it useful to customize a few additional fieldpanes.



One setting we universally did was to remove the dashed highlight around the question text. This was accomplished by choosing a fieldpane, then selecting "None" for the question select in the General group.

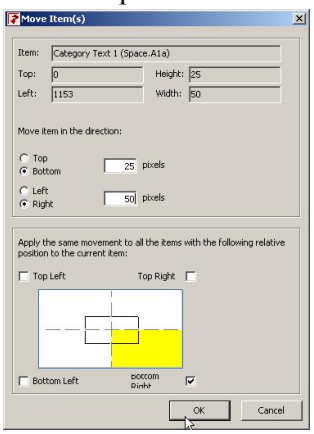
10.6 ASP - login & paradata

We modified the active server pages to provide a different login, and also to store survey data from the interview back to the SurveyTrak database.

10.7 Grouping

Grouping in Blaise IS is a huge challenge, and becoming familiar with the editor does speed up the process a little. Given the fragility of grouping (a change in the datamodel could make the programmer redo the grouping) this should be one of the very last things done to a web survey. Make sure the survey is solid, the logic and fills work, and the pages are correct, and then finally do the grouping.

We learned that what appears under the grouping editor is influenced by what is shown in the modelib and what fields are in the datamodel. To work with a group in the editor, follow these steps:



1. Create the appropriate layouts in the modelib.
2. Make sure all the elements you want in the grouping editor are visible in the modelib (or not)
3. Make sure the layout in the datamodel is correct
4. Prepare the datamodel
5. Define the group
6. Edit the group

We found saving our grouping often was beneficial.

Like the panel and positioning of controls in the menu editor, the placement of items within a web page in the grouping editor is really based upon percents. So if the page width is 2000 pixel, and a control is 500 wide (and your theoretical web page width is 3000), the control will fill 1/4 of the web page, not 1/6. The elements in the group will also expand to fill the data up to a point. So for editing all the groups eventually we came up with the following method:

- Set the global height and width to 1.
- Set all border margins to 2.
- Set all category items to 50 wide.

- Set all question text to 1150 wide.
- Space rows of items 25 apart (each is 25 high)
- Use the "move items" (right-click, choose "move items") to move a cluster of items at the same time. In this picture, everything to the right and below will be moved to the right 50 pixels and down 25 pixels.
- Be patient when moving items & undoing the moving of items (it can take time).
- Do use the multi-select with the control key. Change the attributes of multiple items at the same time after using the multi select.

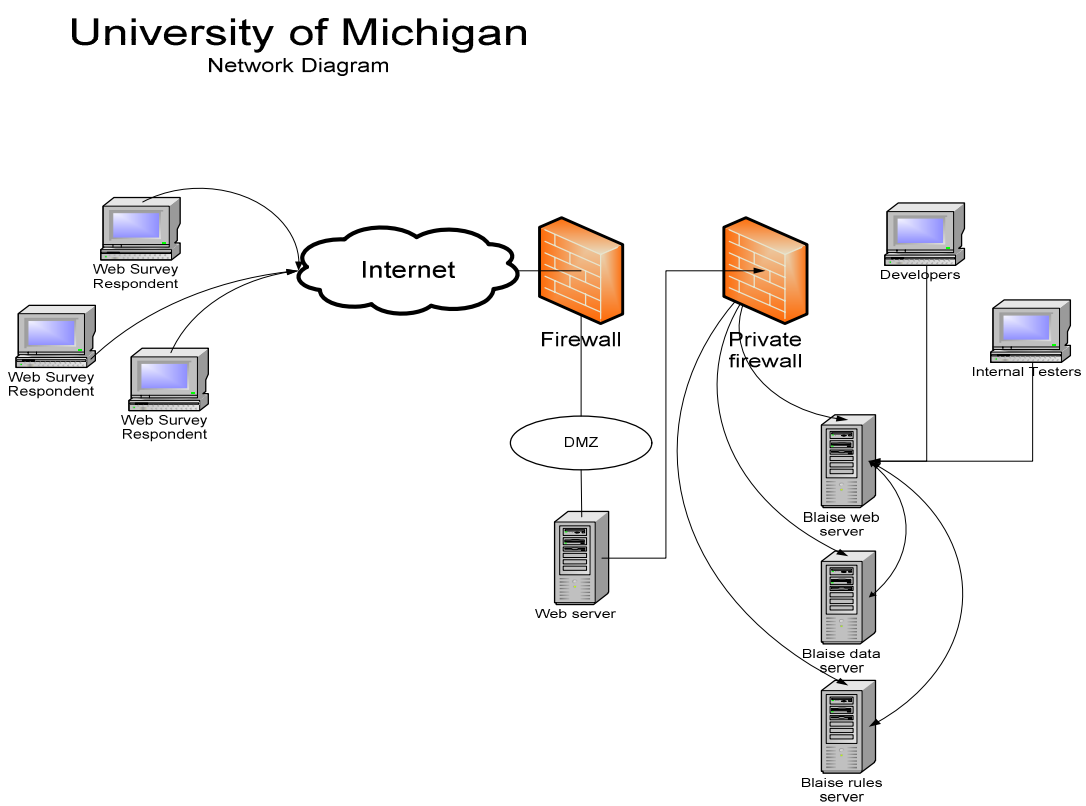
10.8 Stylesheets & HTML

We found that the stylesheets included in Blaise IS work well for Microsoft's Internet Explorer, but generated html code that gave odd results in other browsers. Notably, in Firefox we found that not all the gridlines would appear, in Chrome all the cells within the tables were centered, and similar problems in other browsers. We did not fix this problem due to time constraints, but conjecture it is due to the rendering of the html page from the stylesheet.

We also discovered that horizontal and vertical scroll bars would sometimes appear even though there were no content to be shown. After investigating, we found that the generated html page consists of a number of tables within each other, and the problem was caused by a calculated border width defined for some of those tables. Replacing the calculation with a constant "1" solved the problem.

11. Solution – Servers

The intent was to have three physical servers, and add in additional rules servers as needed. However, we ended up with three virtual servers that acted the same. Since our first project had such a small sample we never really tested the load on the server itself, but we also did not notice any problems for running the survey.



In preparation for the large survey most personnel were being moved behind an additional firewall within the Survey Research Operations. This caused a problem with just being able to access the web server: it was behind the additional firewall, and users who wanted to test from outside the building could not. The resolution was to have the users log in to the servers within SRO via VPN, and then they could access the web site.

The computing support staff made sure that the communications that Blaise IS needed to talk among the servers via the ports were done correctly, and the appropriate ports were opened.

It was confusing at first to learn where the files were placed when the Internet Manager deployed a survey. We used Blaise 4.8.1 for our first survey data collection, and hadn't known that the when a survey is deployed it gets copied to multiple servers, and found it best to let Blaise handle all the files via the package (BIS) definition. We also found that Blaise 4.81 made relative BOI files automatically for use with the data server we had set up.

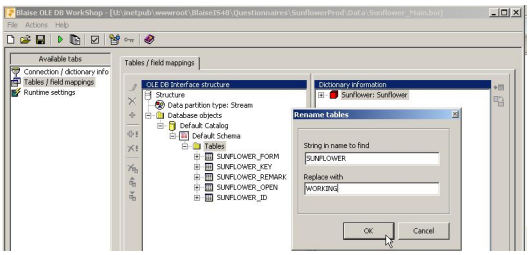
12. Outcome – Data Storage

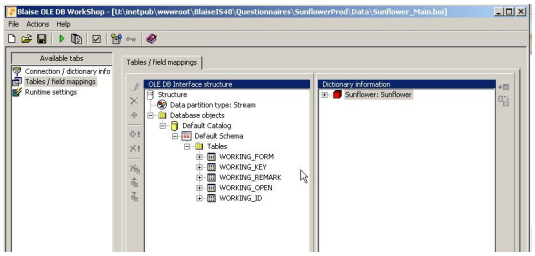
We had to find out which was better: Blaise BDB, SQL storing Blaise data, or both?

Not surprisingly, Blaise BDBs are very handy when using existing utilities. The processes are known for retrieving data, running reports, and maintaining the databases. The downsides are typical -- multiple files per datamodel, file locking problems and speed of access in shared mode, and the possibility of an "oops" moment by deleting a file/moving a file using the file explorer. Placing the data in SQL tables did protect against the downsides (not as likely to accidently delete data/corrupt the data, and won't restrict access speed to the data), as well as provide more security to the data.

The paradata (journal) information was stored in a "flat, no blocks" format that made it very easy to run SQL queries and work with the data using stored procedures and C# utilities. The questionnaire data, stored in the stream format, was used to minimize storage and increase access speed for the respondent. So we had to still export the interview data into a BDB for analysis, but were still able to run basic queries (i.e., number of completes/partials) against the SQL data.

We did find it easy to store all the related survey databases (main, working, and paradata) in the same SQL database without the generic format. We created the tables by making the BOI files in the OLEDB Workshop. After creating the main BOI, we edited a copy of the BOI to point to the Working database by renaming the tables in the main. This created a new set of tables in the SQL database that referred to the same main datamodel.





We never did fully resolve the problem of having the data server hang, but by the same token we did not post a new survey during the data collection period. Therefore, the "survey locks" is still an outstanding issue.

13. Solution – Case Management

A basic case management system was developed for the web mode that extended the functionality of SurveyTrak. A list of email addresses and status were stored in a database, and an email module managed the sending of invitations and reminders. Once the respondent began or finished an interview, that information, along with a summary of the paradata, was stored in the SurveyTrak database. Managers could run existing reports from SurveyTrak, or additional queries against the Blaise IS tables stored in the SQL database. The email module was a major addition to the system, and the discussion of that system can be found in reference (1).

There were relatively few problems in developing the stored procedures to store the Blaise IS paradata and case status to the SurveyTrak tables. Those problems were more of the technical communication methods needed rather than any system design issues.

14. Solution – Paradata

The details of the paradata (journal) information can be found in reference (2). The benefit of the paradata was that it enabled managers and researchers to analyze respondent behavior down to the field level on a page, and included their navigation actions. This is more detailed than the default journal file.

To achieve this required new standard, this included disabling the default journal, modifications to the ASP code, a new standard datamodel for storing the data, and new SQL tables and stored procedures to summarize and transfer the data.

This doesn't have any visible affect on the respondent (i.e., no slowing down of the web page), and the setup overhead is minimal now that the standards have been defined. The development did take some effort.

15. Lessons learned

We learned the following.

- *Standards before implementation*
  This is well known, but bears repeating. Without clear standards before implementation starts, there will be much redoing and discarding of work, and ultimately will take more time and cause more problems than tackling the standards up front.

  When time is an issue, it's better to still agree to something and have it written down even though that standard may not be perfect. The point is that there is a common document that everyone uses, and the tendency is that without that document there are no standards.

- *New systems require lead time*
  One reason we used virtual servers is because they were fast to implement using existing hardware and we didn't have to order new servers. The downfall is that there are multiple processes running on the same physical server, thus, overall, the virtual server cannot handle the same load as a physical server.

  Even so, setting up the servers required time to install software, assign users, roles, and groups, and to make sure everything works. Technical glitches can easily delay the start of server use until issues are resolved.

  The case management system was actually designed and implemented while the survey programming was being done. The last parts implemented were the data export and management reports for the new system.

- *Learning curve*
  Stepping into the Blaise IS world took a great deal of time because of the many components, and required a broad set of skills to master. It was a task best handled a little at a time, but with a big picture overview to help guide the process. A training class that gets down to the littlest details would be helpful.

- *Each area has its own challenges*
  This includes the datamodel source code, modelib, menu, internet specifications, BOI files, ASP, stylesheets, SQL, SurveyTrak, services, servers, file locations, ports, and user rights. Each of these had its own challenges, and it was best to concentrate on learning one part at a time.

16. Summary

The move into Blaise IS has not been without its challenges, but the journey has been worthwhile. Blaise IS offers a rich environment to develop and deploy web surveys, but needs additional systems to supplement the features that are in the internet package. A great deal of customization can be accomplished within the system.

Michigan has greatly expanded their capabilities in the area of web surveys in Blaise, and is ready to conduct additional surveys more efficiently. Future directions include resolving some outstanding issues

that were discovered during this first foray: defining archiving procedures, correcting browser incompatibilities, providing more SQL stored procedures for accessing the data, and updating testing tool and the case management system and reports.

References

[1]Hueichun Peng and Lisa Wood, BlaiseIS Sample Management, IBUC 2010
[2]Jason Ostergren and Youhong Liu, BlaiseIS Paradata, IBUC 2010
[3]Mary Dascola, Genise Pattulo and Jeff Smith, CTT: CAI Testing Tool, IBUC 2007

Challenges of Developing and Supporting Multimode Survey Instruments

Leonard Hart, Scott Reid, and Erin Slyne, Mathematica Policy Research

Over the past decade, Statistics Netherlands' Blaise survey software has evolved into the primary data collection system used at Mathematica Policy Research because of its adaptability in meeting our multimode survey instrument needs. Although computer-assisted telephone interviewing (CATI) and computer-assisted data entry (CADE), also known as high-speed data entry, were the predominant modes of computerized data collection early in the past decade, the introduction of computer-assisted web interviewing (CAWI) and computer-assisted personal interviewing (CAPI) increased our need to find ways to best integrate and leverage the advantages provided by using these other very distinct data collection modes.

This paper reviews our experiences with multimode survey data collection and the challenges inherent in working with instruments in single and multiple data collection software environments. We will discuss supporting multiple web-based data collection systems in an ever-changing internet environment and the effects they have on data collection. We will also describe the obstacles and solutions involved in coding and maintaining a single Blaise instrument that handles multiple modes of data collection in one central real-time Blaise database.

1. Background

1.1 Experience with multimode surveys

Over the years, Mathematica has conducted a growing number of multimode surveys. Currently, we believe we are one of the few in our industry successfully deploying surveys that use one Blaise instrument accessing in real-time one centralized Blaise database accepting data across distinct data collection modes. Most of these surveys combine CATI and CAWI in real time; on a couple of occasions, we have included CADE into the mix. We have also incorporated CAPI data collection into a shared centralized database, but not on a real-time basis, although that is a future goal of ours. Getting to the point of implementing single-instrument/multimode surveys did not happen overnight; rather, it developed over a few years.

Paper-and-pencil methods of data collection have been around since before Mathematica's inception and are still used today to fulfill the needs of simple self-reporting surveys or quick in-person observations. In spite of all the technological advances we have seen since our founding—namely, the proliferation of personal computing; the ability to easily share data via internal and, later, external networks; and the explosive growth of the internet—the paper survey still has its place as a valuable data collection tool.

All the advantages of the computer-assisted interviewing (CAI) systems and software developed over the years, which enable our industry to conduct computer-assisted telephone, in-person, and internet-based surveys efficiently, would go beyond the scope of this paper. From the outset, however, many organizations, including Mathematica, recognized the importance of incorporating them into a data collection arsenal. When properly applied, CAI data collection systems that implement CATI, CAPI, or CAWI instruments allow for greater accuracy of survey data, help speed up the entire data collection process, reduce the burden on our respondents, and can lower overall data collection costs. As these CAI systems and software became more robust, our capabilities to apply them across multiple modes also increased. No longer were we tied to just one method of data

collection. Having the capability to reach respondents in multiple ways enabled us to achieve high response rates for our surveys, even as it has become increasingly difficult to contact respondents in recent years using traditional methods.

CATI was used initially in a stand-alone mode for conducting instruments where utilizing the capabilities of a CATI system to handle complex questionnaire logic and systematically handling and delivering cases to respondents via a call scheduler were advantageous over the paper-and-pencil methods of having an interviewer try to calculate complex logic on the fly or handling thousands of paper contact sheets. CATI later became a tool used to supplement mail or in-person paper surveys, by enabling us to follow up quickly with respondents who failed to self-report within the time allotted for data collection.

CAPI's introduction was very similar to that of CATI's; it was initially used in a stand-alone mode because of the computer assisted advantages it provided in handling complex logic and case delivery over manual paper processes. Later CAPI was used to follow up on self-reported paper surveys or CATI. A knock on the door and the ability of a field interviewer to collect the survey data easily via a CAPI instrument and then be able to transmit that data back to a central office works extremely well when dealing with a universe of respondents who could be difficult to reach by telephone.

As the internet became a more widely accepted communication tool in society, CAWI began to grow as a valuable data collection method. CAWI was first used at Mathematica in a stand-alone mode on surveys in which the entire sampling universe had the capability to participate via the internet. It then grew into a multimode option used to supplement other data collection efforts. In multimode data collection, CAWI was first used as an alternative method of gaining respondent cooperation. For example, as a final follow-up to a CATI data collection, respondents were told they could self-report their data via an internet-based instrument and would not be burdened with a potentially lengthy telephone interview or the inconvenience of scheduling that interview. Over the years, this trend has reversed, because nearly all of our multimode surveys that use CAWI start data collection in this mode. CAWI is often the first option available to a respondent, although CATI is used only as a follow-up. As we have seen on several of our annual or longitudinal surveys, these methods have grown increasingly popular with our respondents; for these studies, CAWI is becoming a dominant mode of data collection. Another potential benefit of offering respondents a CAWI option is in the cost savings that projects see when you decrease the number of mailings, in-person visits, or telephone contacts you have to make to gain the cooperation of the respondent.

Over the years, we have conducted many multimode surveys. Surveys of which we are most proud include the 2003 National Survey of Recent College Graduates (NSRCG), which was the first real-time, one Blaise instrument with a centralized Blaise database, CATI/CAWI/CADE survey we conducted; the 2008 NSRCG, which was the first survey in which we collected CATI/CAWI and CADE data via one Blaise instrument and a central database using Blaise IS; the National Survey of Substance Abuse Treatment Services (N-SSATS), in which we have collected data using CATI, CADE, and CAWI instruments for the past ten data collection cycles and have seen the internet-based mode grow to be the predominant mode chosen by our respondents; and the Ewing Marion Kauffman Foundation's Kauffman Firm Survey, which is now entering its fifth follow-up round and has been able to track a cohort of new businesses successfully via both CATI and CAWI data collection.

1.2 CAI software used

Mathematica has used several CAI systems over the years to support our data collection efforts. We currently use Blaise for our CATI and CAPI studies, Viking Software Solutions Viking Data Entry system for CADE, and different products for web surveys.

We have tried using Blaise for CADE on a several surveys and had some success. However, due to the additional labor involved in programming for the specific requirements of a data entry instrument, while also making it work with the needs of a CATI, CAPI, or CAWI instrument, as well as our client's needs to have data double key verified, we have found it is easier and more cost-efficient to use the Viking system to meet our high-speed data entry needs.

For CAWI studies, we use three different systems, based on the capabilities of each package and how it best addresses particular survey requirements.

Blaise IS is the CAWI system we rely on for the studies in which we determine using one instrument, with one real-time centralized database supporting multiple modes (usually CAWI and CATI) with extremely similar instrument specifications, works best for our data collection needs.

ObjectPlanet, Inc.'s Opinio system is used for internet-based instruments that are short (in number of questions presented and the overall time to complete the survey) and simple in content (for example, simple skip patterns, simple question designs, and no grids or complex tables). A nonprogrammer can easily set up the screens and basic skips available in this system, which makes it cost-effective for very straightforward web surveys.

Mathematica's homegrown WebSurv system, which is an ASP.NET front-end/SQL back-end application, is used for CAWI studies in which we have an extremely specialized data collection need that Opinio or Blaise IS cannot easily meet. Using WebSurv can occasionally give us increased web page design flexibility over our other systems. Another advantage is that most of the instrument set up can be done by junior-level staff from outside the Information Systems department.

2. Support Issues

2.1 Web dominates

Based on our extensive experience supporting different modes of data collection, including CATI, CAPI, CADE, and CAWI, we have found that CAWI requires the most user and programming support. Much of this has to do with CAWI being the "new kid on the block" and with the learning curve for both programmers and respondents in dealing with a new and ever changing technology.

For this reason, we are concentrating on CAWI, but we will also discuss how other modes might be affected. CAWI support is different from the other modes of data collection because we cannot control the respondent environment as we can for an internal user's environment. As with other modes, CAWI surveys will have traditional support issues (such as logic errors or internal system issues). However, CAWI surveys open up a whole new area that we cannot control: the user desktop. We discuss some of these CAWI-specific issues next.

2.2 Authentication processes

Making sure the correct respondent accesses your CAWI instrument properly is as important as the data collection efforts of the instrument itself. Respondents must feel confident of the security of their data being transmitted, especially when it is sensitive or personally identifiable. How a respondent views the capabilities of your authentication process is the equivalent of making a guest feel welcome when entering your home. Because we use three CAWI systems, getting them all to meet our security needs, as well as representing Mathematica web systems as a "welcome place" to participate in a survey, can be a challenge.

Although Blaise out of the box offers some basic authentication solutions, we felt we needed increased security, the ability to integrate the authentication process into other systems, and greater customization than it offered. Therefore, we decided to utilize Full Content Protection (FCP) using ASP.NET. FCP forces a user to be validated when the user requests any content from the web application. Because the FCP authentication process we use for Blaise instruments closely replicates the process used by our WebSurv system and is also written as an ASP.NET application, we have a greater pool of programmers available who are familiar with ASP.NET applications. Because of the similarities of the two systems, we were also able to standardize the feel or the authentication process across both and build common standards for our login and password combinations.

Opinio's authentication process is proprietary and less customizable than FCP for Blaise or WebSurv. User authentication is optional with Opinio, as you can create surveys available to the general public; however, this is not an option we would choose to implement. You can assign user names and passwords to particular respondents. However, because of the proprietary nature of Opinio, it requires staff to learn a second set of requirements for creating and maintaining a study's list of web respondents. We investigated using the FCP authentication piece we use for Blaise CAWI instruments, but found it to be much more cost-effective to use the authentication piece provided by Opinio itself. Ideally, we would find it most beneficial if we could come up with one authentication system that we could use across all three of our products.

2.3 Multiple types of web browsers

According to the Net Applications Browser Market Share report for the period ending July 2010, the top five types of browsers in use worldwide are Microsoft's Internet Explorer (60.74 percent), Mozilla's Firefox (22.91 percent), Google's Chrome (7.16 percent), Apple's Safari (5.09 percent), and Opera Software's Opera Browser (2.45 percent) (http://marketshare.hitslink.com/browser-market-share.aspx?qprid=0). This list covers 98.35 percent of all the browsers in use, but that still leaves 1.65 percent of the internet population using alternate, unsupported, or even obsolete browsers. Factor into that the number of versions of the top five browsers available and you can see that it would be nearly impossible to provide a comprehensive list of browsers potential respondents could use on one of our CAWI surveys.

For the most part, our three CAWI tools have been able to handle the preponderance of the browsers in use; however, as they say about the stock market, "Past performance is no indication of future returns." When we first started supporting CAWI instruments, we did come across situations in which a question would display one way in one browser and a different way in the others. However, as CAWI use grew and the tools available became more robust, these differences have become less problematic.

We are seeing an increase in the number of respondents using mobile or handheld devices, and there is a growing concern that our current web design standards might have to be adjusted further as this

segment of the browser marketplace increases. The Net Applications Browser Market Share report for July 2010 shows that handheld or mobile browsers make up 2.2 percent of the market, up from 0.98 percent in October 2009 (http://www.netmarketshare.com/report.aspx?qprid=61).

The internet is an ever-changing environment, and our mantra for the past decade when working with CAWI instruments has been "Keep it simple." Although it is possible to do many exciting and flashy things within an internet instrument, you have to make sure that almost any respondent can properly view the page you are presenting and that you are not introducing any bias into your questionnaire design because of browser behaviors.

2.4 Synchronization and merging of data between disparate systems

Getting disparate data collection tools to work together smoothly for a single questionnaire can be a challenge. The data is more than likely stored in either a proprietary format or database, and the data variables and values probably will be completely different for each because of the unavoidable idiosyncrasies of each system.

The first challenge we encountered when using disparate systems for the same instrument was trying to prevent duplication of data between the different modes of data collection. Although you would hope that the data between the systems would be identical, we found that is not always the case, especially on institutional or establishment surveys in which two people could be reporting for the same entity. For example, in the early rounds of data collection for a facilities study, a particular facility might have completed a mail survey that was marked as received in our sample management system (SMS) before being passed to data entry staff to input into the Viking data entry system the same day a telephone interviewer contacted the facility to complete the survey via the Blaise CATI instrument. Meanwhile, someone else at the same facility might have logged into the WebSurv CAWI instrument the same day and completed the survey there. Because there were differences in the data collected in each instrument, additional calls to the facility would have been needed to determine which mode of data collected was the one we wanted to keep.

One way we cut down on the discrepancies between the systems was to increase the number of real-time updates between the systems being implemented. For example, as soon as a mail survey was marked as received in our SMS database, the login for the web survey was disabled in the WebSurv system and the case status there was adjusted accordingly. For CATI, if the case was delivered by the Blaise call scheduler, or if selected for manual dialing by a CATI interviewer, a call was made to the SMS database which pulled the status into Blaise and let the interviewer know that a call was not necessary and upon closing the case it would be then statused as mail received in Blaise and no further dialing attempts would be necessary.

Because WebSurv and the SMS data are both SQL databases, the triggers and stored procedures used to make the updates between those systems were relatively straightforward; however, we had to develop a process to share data between the Blaise and SMS database, which in turn would update the WebSurv database. We developed a Dynamic-link library to use as an interface. The Blaise instrument (or the SMS database) can use this interface to pass data between these systems.

The other challenge we encountered involved merging and processing the survey data between these systems. Because each system collects and outputs (or allows you to access) its data in its own format, there needed to be a way to "crosswalk" the data into one common format. The biggest problem is that any mistake in the crosswalking leads to errors in your final data and the eventual analysis. When dealing with disparate systems, we have tried as much as possible to name the variables the same in each system, make sure we ask the questions in the instruments in the same

format, and make the answer categories or data types the same. By keeping the modes as common as possible, there is less chance of the data not lining up when combined via SAS processes.

Because of these challenges, we are strongly considering implementing Datalink on a future project, so all the survey-related data will be stored in a common SQL format, and the communication and updates between these systems should be handled easily. More important, this is why we try to use one Blaise instrument with one central real-time database when we have a multimode need.

2.5 Around-the-clock support

Multimode surveys have support needs on varying levels and timelines. For the most part, pencil and paper and the software to enter the data recorded on hard-copy instruments are the easiest to support technically and have least urgency when requests come to programmers, as there is no respondent waiting for a solution. Supporting CATI and CAPI is a little more challenging than CADE, because you have the interaction with your respondents to contend with; however, having the interviewer as part of the process can reduce the urgency when problems do arise. The CATI or CAPI field interviewer can easily explain to the respondent the issue we might be encountering and negotiate a time when we can continue the survey.

When it comes to CAWI, the level of support needed and the urgency factor increase. The CAWI instrument itself is the main point of communication between you as an organization and the respondent. Because the internet never sleeps, keeping CAWI respondents happy becomes a 24 hour a day, seven day a week, 365 (or 366) day a year proposition. Because problems with CAWI surveys can pop up anywhere, they can be the hardest to diagnose and fix. For example, if a respondent cannot access your survey, the problem could be anything from an Internet Service Provider (ISP) issue, the connection to our web servers having gone down, the web server itself having failed, an internal network issue, or a programming error putting the survey into an unstable state.

When CAWI was a data collection tool newly available to Mathematica, we encountered support issues relating to browsers, internet connections, software, and user errors; compounding these issues was that this was a new technology for end users and for us. Over time, survey support staff have gained more experience supporting CAWI instruments, and they now triage calls and emails. Before, programmers had to become involved in every CAWI issue. Now, survey staff can check that a respondent's browser is functioning properly or that the respondent can reach other sites besides our survey's site, which helps determine if the problem is on the respondent's end or ours. In addition, respondents are becoming more internet savvy and can address items such as connectivity. Also, web survey packages have gotten better at addressing some issues, and we have built new features into our supporting tools to address certain inefficiencies.

2.6 CAWI instrument troubleshooting

No matter how much instrument testing you do, occasionally a programming bug will get into the real world and create problems for your survey. One of the biggest support challenges is trying to determine what exactly a CAWI respondent is doing when that problem occurred.

Unlike a pencil-and-paper survey, in which the respondents will occasionally leave comments in the margins if a question is not clear, or a CATI or CAPI survey, in which a programmer can extract an accurate description of the problem from the interviewer, communicating with CAWI respondents is much more difficult. They may choose to tell us nothing and just complain about not being able to complete the survey, or, if you are lucky enough to secure their cooperation in triaging the issue, they might not remember or have recorded exactly what they were doing when the instrument

problem occurred. However, there is hope in trying to determine what CAWI respondents are doing when problems arise and information is limited. Occasionally, you can glean information from some of the web server logs, but one of the best tools available to the CAWI instrument programmer is a web audit trail.

With Opinio surveys, tracking respondent movements and issues that arise from those movements is less of a problem, as the logic in these surveys is less complex than what is found in our homegrown WebSurv system or in Blaise Internet instruments. In WebSurv, it is possible to trace the path a respondent takes through the survey. The system tracks the web page from which the respondent submitted data and exactly when the respondent submitted that data. The system also determines if the respondent changed the data initially submitted by backing it up on his or her own or if the instrument itself put the respondent at an incorrect point in the survey.

Trying to implement CAWI Blaise audit trails has been a hit-or-miss proposition. For the initial Blaise surveys we hosted, we used a third-party piece of software called C2B ("CentERdata to Blaise"), developed by CentERdata at the University of Tilburg. C2B provided the interface between PHP (an open-source server-side scripting language) and our Blaise instruments. As part of the C2B system, we could capture detailed audit trails that showed us the movement and data changes respondents made. This proved to be a valuable tool in finding the programming equivalent of a needle in a hay stack when trying to resolve programming bugs. When we switched from C2B to Blaise Internet several years ago, we hoped the journaling feature built into Blaise would provide us with the same level of detail we had with C2B. However, we have been unable to replicate that success using the Blaise IS journal without customizing it greatly for each project. We hope this is something that Statistics Netherlands can improve upon in future versions of Blaise, as it is an important tool to have at our disposal.

3. Blaise Single Instrument for Multiple Modes

Using a single Blaise instrument for surveys conducted for multiple modes has advantages and disadvantages. Perhaps the biggest advantage is the ability to maintain the data in one centralized database. Because there is no need for special programming to synchronize different databases, data management is easier to handle.

The major drawback of this type of single-instrument design comes from the complexity of our surveys, which have occasionally challenged the capabilities of Blaise IS. Complexities have included handling multiple arrays of survey data, addressing complicated edit checks that involve calculations using fields spanning the entire survey, sporadic issues with question display in various browsers, CATI management not recognizing web activity, incorrect delivery of cases in the day batch, and problems displaying Blaise tables.

Statistics Netherlands works with us continuously to improve Blaise IS and has provided us with periodic updates or workarounds to resolve most of these deficiencies. Updates have included solutions for browser display issues and complex table displays; in some instances, however, we still have to write additional code to overcome issues. In spite of these issues, using a single instrument for conducting a multiple-mode survey is the method we have chosen more than any of the others, as the advantages have easily outnumbered the disadvantages.

With all modes needed for a survey residing in the same instrument, multimode surveys require less redundant questionnaire programming. For most of the questions we encounter, one field can be

declared for all modes. For those questions that need to be presented or asked differently in each mode, we used Blaise's multilanguage capabilities and declared each mode as a separate language (see code example 5.1 for a CATI/CAWI example).

Because the logic path (the Blaise rules) and text fills are usually identical for multiple modes, redundant code is eliminated. However, if the logic or text displayed is specified differently for each mode, then separate *IF statements* and question variables are needed in the *RULES section* to handle the differences for each mode (see code example 5.2).

Fields in Blaise group tables, which use only enumerated types and have question header text in CAWI, although needing introductory questions in CATI, can be shared by using two separate LAYOUT sections (see code example 5.3). These separate layout sections are declared in the mode library individually for each mode. Although the planning and actual programming in doing this can be more involved and time-consuming for the instrument programmer, the benefits of sharing fields and coding just one instrument outweigh the cost of the complexity.

Writing the instrument code for a single Blaise instrument to be conducted in multiple modes can complicate your programming efforts, particularly when the survey is complex and includes tables, which we use often in collecting fields with different data types. Because the programming requirements often differ between modes, challenges arise when coding to accommodate all the modes involved. Sharing frequently means compromising; although we were able to avoid trading off capabilities most of the time, there were instances in which we had to write superfluous code or forgo a request due to increased programming costs—for example, a multimode survey instrument that requires CATI to allow for nonresponse options, but not in CAWI. Because one field cannot have different field attributes, we had to write redundant edit checks in CATI for every field shared with CAWI (see code example 5.4). This inability to declare different attributes for each mode within a field particularly causes problems when programming tables. The *don't know* and *refusal* options in CATI interfere with the table display in CAWI. The solution involved creating two separate tables, one for each mode, so they could display properly in a web browser and be asked in the manner needed by a CATI interviewer. Because we had to allow for a respondent possibly to transition between modes, as the instrument could be started in one mode and then be prompted to finish in the other (usually starting in CAWI and finishing in CATI), we occasionally had to assign the collected value from one mode into the other mode's field so it would then be on path and available as part of that mode's rules. Creating two separate instances of the same question can compromise data integrity if this is not carefully programmed and then tested (see code example 5.5).

Although the flexibility to switch easily between modes is powerful, it is not always reliable. We have experienced difficulties, including delayed day batch updates and case management issues. For example, we sometimes have inadvertently called a respondent to interview just after that respondent had completed the survey in CAWI because the CATI day batch had not yet refreshed to reflect that the case had been completed and should not be delivered to a CATI interviewer. To solve this problem, we developed a Maniplus program, scheduled to run every 15 minutes during interviewing hours, that updates the day batch if necessary. However, when accessing a case via CAWI, the CATI management block is not updated; as a result, the call scheduler is still not completely up to date.

An additional advantage to using a single Blaise instrument with one centralized database for surveys conducted in CATI, CAWI, and CADE modes is in its case-locking capabilities. CAWI respondents, CATI interviewers and CADE staff rarely encounter this feature when accessing data in their respective data collection modes. For example, when a CATI interviewer talks to a respondent

on the telephone, the respondent is unlikely to log into the web survey at the same time. This feature is important, however, on surveys in which different respondents might have to complete different sections of the same survey. An example would be a survey of businesses in which we ask questions about financial records and how the business is managed. In this situation, different staff from within the business might have to address certain questionnaire sections based on their knowledge and expertise. However, we would not want more than one respondent to access the same instrument in CATI or CAWI mode at the same time, because one could potentially change the other's data. One minor issue we have noticed with Blaise in this multimode, one database concept is when a CATI interviewer, CAWI respondent, or programmer has to access the case again immediately after exiting. Occasionally they might be presented with an error message that the case is not available, when it actually should be. The message is presented because it can take from a few seconds to several minutes to clear the lock. The delay does not cause issues often, so this is not a deal breaker for us using the single instrument in this way.

4. Conclusion

4.1 Mathematica successes with multimode

Mathematica has conducted multimode surveys successfully for many years. It wasn't until the past six years, however, that we have been able to use a single real-time database containing both multimode CATI and CAWI surveys. Previously, like any other survey organization, Mathematica had one database for each mode, then either used a complex procedure to combine the data at the end of the data collection or tried to move data from one database into another while interviewing was active. Either process was time-consuming, expensive, and introduced the possibility of errors in the final data because of all the procedures moving survey information around.

By moving to a single real-time database multimode CATI and CAWI survey, Mathematica has successfully reduced the total development cost for these types of multimode surveys. By eliminating the steps moving data from one database to another, we have reduced costs and eliminated possible errors in the data movement between systems.

Moving in this direction did introduce new survey management issues that had to be addressed, however. For example, suppose you start data collection by calling a respondent and getting partway through the interview, then set up an appointment to complete the interview later. Before the appointment is delivered in the call scheduler for telephone interviewing, the respondent responds to the survey by CAWI but still does not complete it. Do you keep the original appointment that was set, do you delete the appointment, do you have the case go back into the normal pool for calling, or do you adjust the appointment to a later date hoping the respondent will complete the survey by CAWI so a call back won't be necessary? Another example: you want your respondents to respond by CAWI so you can lower your interviewing labor costs by not having to pay for telephone interviewer time. If the respondent doesn't finish the case by CAWI, how do you then manage calling the person for a follow-up?

Another beneficial aspect of using one database and one instrument for both CATI and CAWI is that it decreases the instrument development time. In theory, creating one instrument should save development time and labor costs, because you do not have to write two different instruments. One might think this would cut your development cost in half because you are writing just one instrument. In reality, however, you probably will save only about 30 percent in development costs. This is primarily because of the complexity introduced due to mode differences, which take longer to program. For example, it is not always possible to design a question that can be displayed to a CAWI

respondent or asked by a CATI interviewer in a similar manner. On the other hand, if you do have to make a change to the code, you might see additional savings because you are programming just one instrument for both CATI and CAWI. The changes would be in one instrument, whereas previously, if you made a change in one instrument for a particular mode, you would have to make that change in the other.

From our survey division's perspective, having one database that is maintained in a real-time environment offers better reporting of the data they collect, which translates into better management of the survey and decreases the overall costs. Reports available to the project are up to date and available anytime they have to be reviewed. In contrast, in the past, reports were normally created after some data combination process occurred and were prone to logic or crosswalking errors. Due to the cost and possible programming logic in combining of data, reports were basically a snapshot picture from a particular moment, whereas with one database it is much easier to create an up-to-the-minute report. Another benefit of having all the data in one location is that it enables us to write a reporting process only once, instead of possibly writing a report for one mode's database and then needing a secondary report process for the other's database.

We have conducted a few surveys in which we have done CATI/CAWI/CADE in one instrument, one database, all in real time. However, due to limitations of using Blaise as a high-speed data entry program, we have not had as much success as we would have liked. Although Blaise can do data entry, it is not very cost-efficient, cannot do double key entry verification without a lot of tricks and/or additional programming code, and cannot keep statistics on the error rates of the data entry staff. In addition, the extra coding increases the complexity of the instrument and adds greatly to the programming cost. We have found it easier and cheaper to use specialized software designed for high-speed data entry than trying to incorporate this mode into our Blaise real-time, one database concept. We will continue to look at Blaise as a high-speed data entry application if some fundamental changes happen (for example, native double key verification). As a side note, we have discussed moving to a relational database for back-end data storage, and this might allow our high-speed data entry package to write directly to the same relational database.

We have had some success with Blaise writing to a relational database, but not using the Blaise Datalink application. The product we developed uses the Blaise API and allows a Blaise instrument to write to its native database while also writing to a relational database. We have used this product, called SQL2Blaise, on multimode surveys in which Blaise is used for CATI and we use our in-house web product called WebSurv. We have thought about expanding the use of SQL2Blaise, but we will wait for additional testing of Datalink in Blaise 4.8.2 before making a final decision.

4.2 Future of Blaise and multimode surveys at Mathematica

Building upon the success we have had with multimode using one instrument for CATI and CAWI, and one database accessed in real time, we would like to begin to include CAPI in this design. Our ultimate goal is one instrument, one database accessed in real time for CATI, CAWI, and CAPI. We have tested this concept in a nonproduction environment with great success, but in the United States wireless broadband coverage is still a strong limiting factor. Coverage continues to increase, but there are still vast sections within the United States where wireless broadband doesn't work efficiently or you have to use multiple carriers to cover the areas in which you are interviewing, which increases the costs for your project. We envision this being an invaluable solution to meeting multimode data collection needs efficiently in the future, but it is still a few years away. However, when we do reach this point, it will raise some interesting questions. Should the CAPI instrument be another version of the CATI instrument or should it be a CAWI instrument shared by all modes? Testing and client preferences will probably determine the final resolution on

that issue, and there is no definite opinion in either direction. Also, if you are in the field doing real-time CAPI and writing data directly to a centralized database at the home office, what happens if the broadband connection goes down? The latest version of the .NET framework might offer some possible solutions if the connection back to the office cannot be made to store the data locally and when the connection returns a process in the background could upload the data from a laptop back to your central database. We feel we are getting closer to moving in this direction for real-time multimode CAPI surveys, but we think we are still a few years from this becoming a reality.

Because of the cost of producing a Blaise IS instrument compared with our other CAWI systems, we hope the next generation of Blaise (Blaise NG) will be a lot easier and more flexible to use for developing and deploying instruments. For this to happen, the designer portion of the system must be very easy to use (possibly to the point where a nonprogrammer could design the basic survey) and must easily address possible mode differences without creating additional fields or tricky coding to overcome these difficulties.

There has been a push in recent years toward unimode instruments, in which all modes are designed with commonality in mind. If unimode instruments do move into the forefront, that could help address the problems of differently designed questions for different modes. We are still left with the problem that field attributes could be different for certain modes, but, hopefully, Blaise NG will also address that.

Finally, we believe multimode with one programmed instrument and one real-time centralized database is here to stay. There are several systems already on the market that attempt to do this, but typically they cannot handle the complex surveys we conduct at Mathematica. We hope Blaise NG will continue to build upon this concept, while making it easier to create instruments without the additional code needed to handle mode differences and improving its ability to create CAWI screen layouts. Having seen some early beta versions of Blaise NG, it looks as if Statistics Netherlands is addressing the cumbersome process of making changes in various locations to get a screen to display across multiple modes and in various internet browsers.

5. Code Examples

5.1 Example of using Blaise's language utility to declare each mode as a separate language

```
A2
     ENG "What is the correct business name?"
       WEB "@BV2. Please enter the business name.@B@/" : STRING[50]
```

5.2 Example of using IF statements to specify different paths between modes

```
IF (A5a=NonOwner) THEN
    IF (piModeOfProcessing = CATI_) THEN
        A5a_NonOwner
    ELSEIF (piModeOfProcessing = WEB_) THEN
        A5New
    ENDIF
  ENDIF
```

5.3 Example of using two separate LAYOUT sections, one for each mode

```
AUXFIELDS
       Label
      WEB "@BF3. Equity investment is money received @U in return for
       some portion of ownership@U, and it is another way to fund
       business expenses.
       During calendar year 2009, did the business obtain @Bequity
       financing@B from any of the following sources?@B
       @/@/@I@OPlease indicate Yes or No for each item.@I@O"
      ENG "Equity investment"
  : STRING[30], EMPTY

RULES
...

LAYOUT
       BEFORE Label NEWPAGE
LAYOUT (Internet)
       BEFORE Label NEWPAGE
       AT Label FIELDPANE BISQuextextOnly
```

5.4 Example of redundant edit checks due to the inability to declare different attributes for each mode within a field

```
F3f
IF piModeOfProcessing = CATI_ THEN
    F3f <> EMPTY "You must answer."
ENDIF
F3g
IF piModeOfProcessing = CATI_ THEN
    F3g <> EMPTY "You must answer."
ENDIF
```

5.5 Example of storing the collected value into the other mode's field

```
FOR I := 1 TO 9 DO
    IF piModeOfProcessing = CATI_ THEN
      C1CATI[I](aItem[I], aLetter)
      C1[I].C1 := C1CATI[I].C1
      C1[I].C2 := C1CATI[I].C2
      C1[I].C3 := C1CATI[I].C3
    ELSE
      C1[I](aItem[I], aLetter)
      C1CATI[I].C1 := C1[I].C1
      C1CATI[I].C2 := C1[I].C2
      C1CATI[I].C3 := C1[I].C3
    ENDIF
  ENDIF
ENDDO
```

Impressions of Basil

Kathleen O'Reagan, Richard Frey and Karen Robbins, Westat

1. Introduction

Basil is an alternative data entry program for Blaise questionnaires, specifically designed to handle self interviewing situations. Since Basil combines the strength of the Blaise rules engine with the flexibility of Maniplus, which allows programmers of Basil to create tailor-made applications, we have been testing the use of these capabilities for process and workflow management. This paper describes the overall functional requirements, the design and subsequent coding techniques used, including the integration of Maniplus, Basil and SQL Server, and lessons learned when incorporating Basil in a tracing system project.

2. Tracing System Requirements

2.1 General Tracing Systems

In most long term surveys, losing track of some participants is inevitable. Tracing is the process of locating participants who can no longer be contacted using any of the location information on record. A variety of tracing methods and sources of location information are used in an attempt to find a participant. Information gathering sources and methods include phone numbers, emails, letters, relatives, friends, coworkers, neighbors, schools, community organizations, places of worship, directory assistance, and government offices such as voter registration, taxation, department of motor vehicles, and social services. Most of these sources can be researched simultaneously to locate the participant in a quick and effective manner. Timely tracing is a key component in continued data collection. When the participant has been located, all the data on record will be confirmed and updated in the study's survey management system. The SMS will also contain the historical records of the participant since joining the study.

Tracing systems can be can be made more intricate with the addition of a case management component to include supervisor review, roles and permissions to access functions and data, status codes milestones and reporting. For the purposes of this paper, we are addressing the data collection and verification activities of a tracing system to show how Basil is used.

2.2 Westat Tracing System

The functional requirement of a tracing system is to obtain location information about the participant. The Westat Survey Management System (SMS) contains the participant's name, physical and mailing addresses, email address, home, work, and cell phone numbers, and the name, address, phone and email of contacts who would know how to reach the participant in case their phone number or address changed during the course of the study. Tracers have a wide variety of sources to use to produce leads to the missing participant. When the participant is successfully located, they will be asked to review and update their information on record.

In the Westat Tracing Logical Flow (Figure 1), the Westat SMS Workflow issues assignments to the interviewer to conduct a survey at the participant's residence, to the Telephone Research Center (TRC) to make a followup call to the participant's home, work, or cell phone, and to send letters to the participant's physical , mailing, and email addresses .

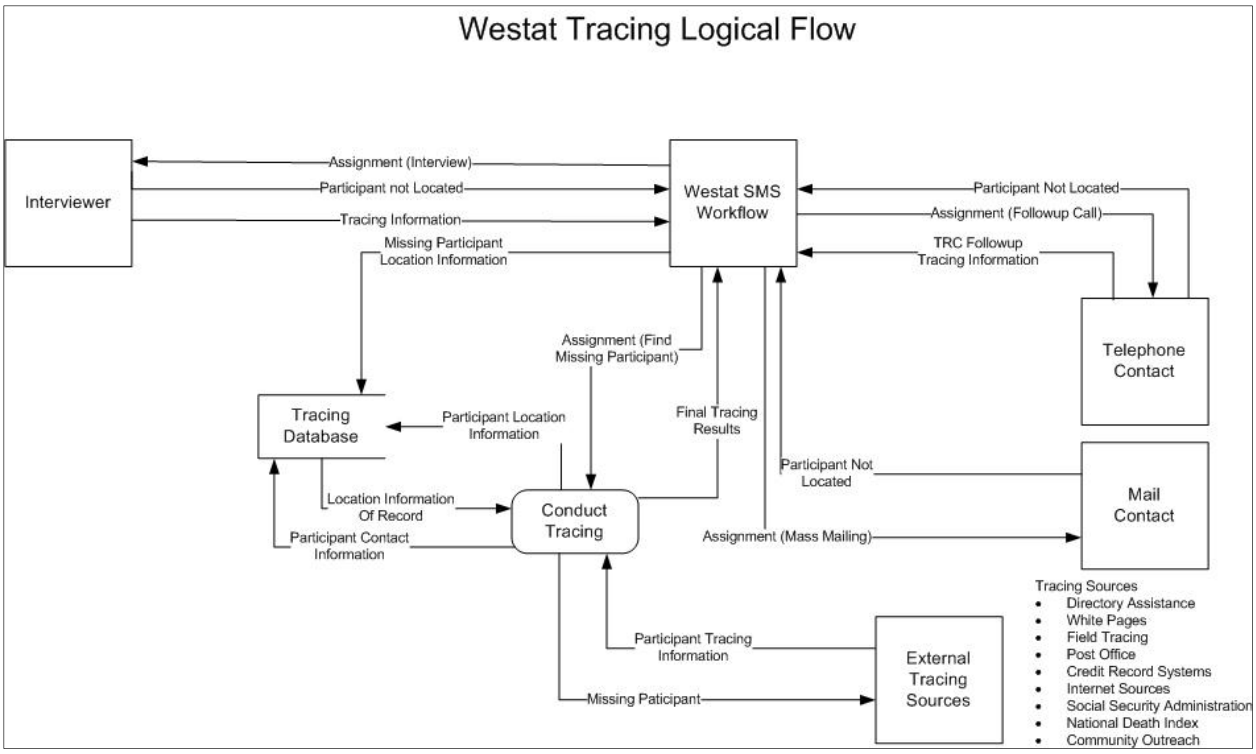


Figure 1. Westat Tracing Logical Flow

The tracing task of the Westat Tracing Logical Flow begins when the participant is not located. Responses from the workflow assignments signal that part of the information on record may no longer be correct. The interviewer might have discovered the participant's address on file is vacant. The TRC could have received a 'disconnected' message when they call one of the provided phone numbers. From the mail contact source, the letters or email could have been returned as not deliverable. With this recognition of having incorrect data for a participant, the complete record of that missing participant, their personal information and their contacts' information, is passed from the SMS to the Tracing Database.

The tracer conducts the search for the participant in stages. First, the tracer tries to locate the participant using the various phone numbers, addresses and email that they provided. If the participant was not found, attempts to reach each contact reference provided by the participant will be made by phone, mail and email.

When the tracer has exhausted all the location means provided by the participant and still has not located the participant, tracing focus shifts to investigating external sources. These sources are less cost efficient and are therefore used last due to their history of yielding lower results. Examples of external sources include directory assistance, post office, field tracing, internet searching, community outreach and government offices such as the Social Security Administration, Department of Motor Vehicles, and social services, and local businesses for instance credit card companies.

Whenever the tracer is successful in reaching the participant, the tracer reviews and updates all of the participant's new location information and the participant's contact references information. At study specified intervals, the SMS will retrieve this verified update information and add it to the SMS database.

3. Using Basil for the Westat Tracing System

Blaise components, Basil, Manipula, and the relational database storage capabilities offered by Blaise Datalink, were incorporated into the latest prototype of Westat's Tracing System. This system allows for the tracing of missing participants using a variety of location sources, reporting and processing options. It includes a set of core capabilities and can be customized to meet the needs of specific project application requirements.

3.1 Tracing System Capabilities

At a minimum, the tracing system needed to provide core project requirements allowing for the inclusion of missing participant location information, the management of tracers, the tracking of location changes, and the ability to work external sources within the system.

The tracing functional requirements were matched against the Blaise components and implemented based on the amount of time the user was using the interface. For instance, adding or assigning a tracer is usually done once, therefore, using the DEP to add a tracer and give permissions did not require the sophisticated look and feel Basil would provide to the user interface.

Figure 2 shows the functional tracing requirements and the Blaise component used to implement the requirement.

Functional Requirement	Blaise Component
Controlling system	Manipula
Assign a tracer to a missing participant	Manipula and DEP
Add a tracer	Manipula and DEP
Delete a tracer	Manipula and DEP
Add a participant to the tracing system	Manipula and Datalink
Add a contact for the participant	Basil application and Datalink
Participant Tracing Information Sheet	Basil application and Datalink
Contact Tracing Information Sheet	Basil application and Datalink
External Source Work Sheet	Basil application and Datalink
Radio buttons, check boxes, and buttons	Basil input controls which trigger execution of a Maniplus procedure

Figure 2. Tracing Requirements

For the remainder of this paper, we will focus on the actions that use Basil as the basis for collecting and verifying location information. These include the verification of participant (P-TIS) and contact (C-TIS) information, and the searching of external sources.

The first screen viewed in the Westat Tracing System is shown in Figure 3. This is the main menu screen controlled by the Manipula setup, TracingStratup2. From this Manipula setup, the supervisor manages the tasks of adding a tracer to or deleting a tracer from the system, assigning a tracer to find a particular participant, and adding a missing participant to the system.

Figure 3. Manipula Setup for Tracing Participants

Tracing and location verification of the missing participant using the information of record utilizes the buttons P-TIS (Participant Tracing Information Sheet), C-TIS (Contact Tracing Information Sheet) and Add Contact.

If the participant and contact reference information on file hasn't led to locating the participant, then the the Manipula procedure launched by the External Source button will provide other means to locate the participant.

3.2 Basil, Manipula and Manipula Extensions

As mentioned in the paper's introduction Basil is an alternative data entry program for Blaise questionnaires, specifically designed to handle self-interviewing situations. Basil combines the strength of the Blaise rules engine with the flexibility of Maniplus. It allows for a layout with a diverse variety of field types. It can readily provide on a single screen one of a large variety of different types of standard data collection input forms. Within the context provided by the screen, users can enter the requested information in whatever order works for them. Moreover, users can take advantage of available buttons to launch an event associated with one or more inputted values.

The Manipula setting that allows for the communication between Basil and Manipula is Interchange. Interchange allows work directly on the record from the calling process, in this instance Basil. This is only possible when the data model of the calling process is the same as the data model used in the Manipula setup. Generally, this is done by using the reserved word VAR in the USES section of the Manipula setup. Each Basil application identifies the Manipula setups, the datamodel and the update file in the Application section in the Basil datamodel.

Basil relies heavily on the usage of Manipula functions, methods and instruction extensions. The extensions are used in building procedures to enhance Basil controls. In the Westat Tracing application there are two Manipula setups. The one named Generic is found in the Blaise Samples from the Blaise installation. This setup contains generalized procedures for writing to an audit trail, error handling, printing, and exiting the questionnaire. The other setup is TracingProcedures in which procedures and dialogues specific to the tracing application are located. Specific procedures include those to handle button controls for dialing telephone numbers, collecting new address information and launching Basil pages. The Application section code in each Basil datamodel identifies each setup as follows:

```
BASIL "<application title='Westat Participant Tracing System'
            width=950 minimumwidth=800 Height=850 clientheight=825
            resource='Generic.dll'
            icon='BLAISE' sizeable=true fontface='Microsoft Sans Serif' fontsize=8
            fontcolor=#663300 color=#FFFFEF
            setups='TracingProcedures.msu /KdmGeneric=TracingTIS
                    /NufGeneric=TracingTIS;
                    Generic.msu /KdmGeneric=TracingTIS /NufGeneric=TracingTIS'
            oncreate='Generic.Initialisation'
            onclosequery='Generic.StopQuestionnaire'
            onhelp=blaise:exec('Generic.chm')
            onrangecheckerror='Generic.RangeCheckError'
            ongeneralerror='Generic.GeneralError'
            singleinstance=true>
          <panel left=0 align=client>
            <panel top=0 align=client color=#FFFFEF>
              <content-area horizontalscrollbar=false verticalscrollbar=true>
            </panel>
          </panel>
       </application>"
```

3.2.1 How Basil Was Used

On the Participant Tracing Information Sheet (Figure 4), you see a number of fields defined as a Blaise or user defined type according to Blaise syntax. They are each displayed as a label, input box or button using Basil layout controls.

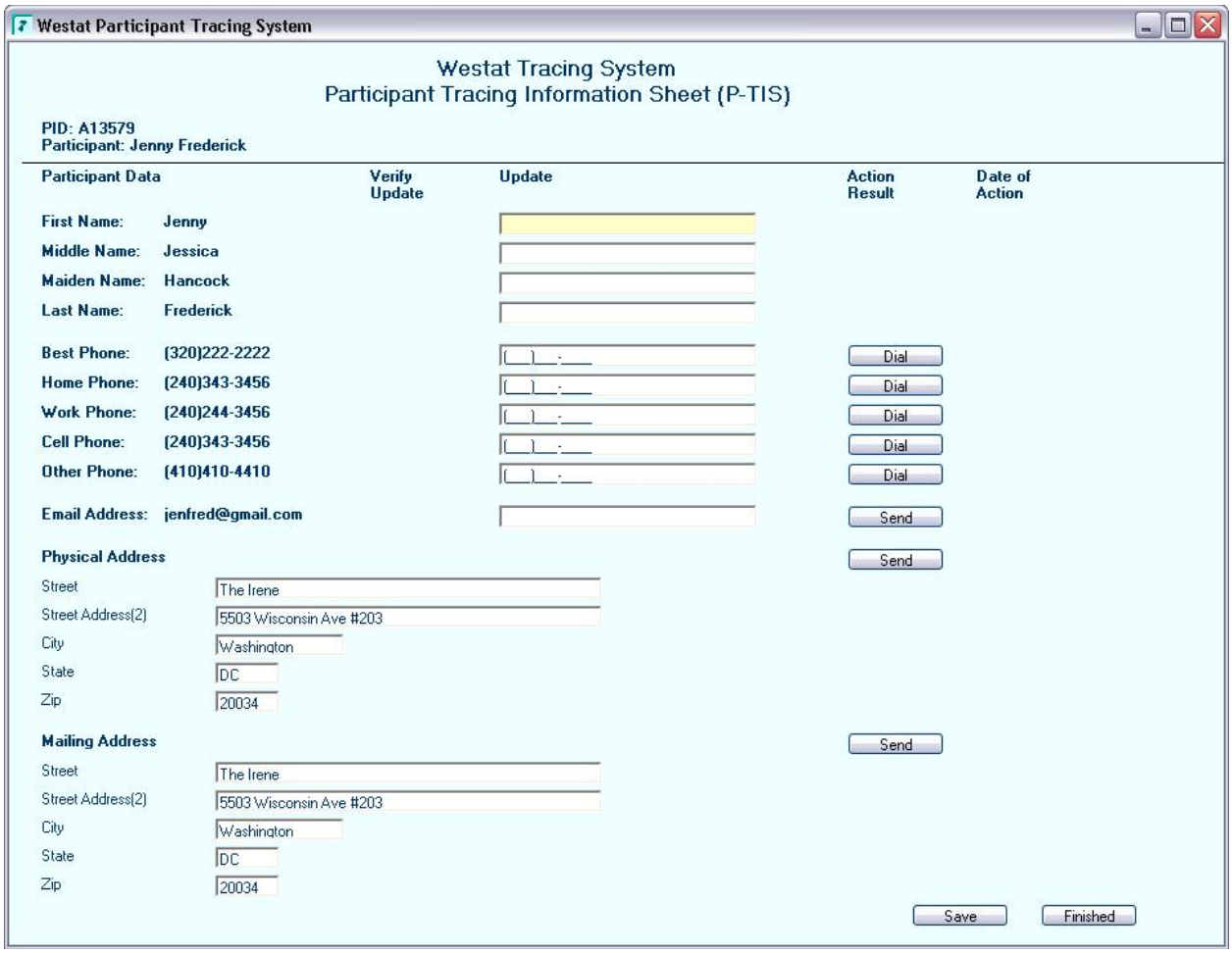


Figure 4. Participant Tracing Information Sheet

These controls are defined in the question text of the field similar to the way multimedia instructions are used in a Blaise multimedia instrument. The following code snippet shows the definition of the First Name field:

```
FirstName
      BASIL "<question span=5>
               <label left=25 width=100 topmargin=5 text='<b>First Name: </b>'>
               <label left=120 width=100 topmargin=5 text='<b>%TIS.FirstName </b>'>
             </question>"
      : STRING[20]
```

The FirstName field is defined as a string. Basil code uses two label controls. The first is used to display the label text ‘First Name:’. The second label control is used to display the current value of FirstName as indicated by the Basil fill string % symbol followed by the fully qualified field name. In the Rules, FirstName uses the .SHOW method.

The question attribute, span=5, indicates, the next five questions, including FirstName are to be grouped horizontally on one line.

The input box is used to hold any update made by the tracer to the FirstName field. The code for this field is below:

```
auxFirstName
      BASIL "<question>
              <input $auxAuxFieldPos>
             </question>"
      : STRING[20]
```

You'll notice the input control is followed by $auxAuxFieldPos. This is an AUXFIELD that holds the formatting of the input box. The $ is a fill string symbol. The following field assignment in the Rules section establishes the location, width, color and the 3D look.

auxAuxFieldPos := 'left=380 top=5 width=200 height=18 focusedcolor=#CCFFFF ctl3d=true'

The auxAuxFieldPos field is used for each name, telephone and email input box. Since we wanted each input box to display in a single column, using the fill string, auxAuxFieldPos, all the formatting attributes of each input box control were set the same.

The telephone input box has a mask to guide the entry of the telephone number. To display a mask the editmask attribute was added to the telephone input control.

```
<input $auxAuxFieldPos editmask='(999)999-9999' >
```

The last field of the First Name line is FirstNameDateLastAction. This field has the DATETYPE type and a Basil label control.

```
FirstNameDateLastAction
    BASIL"<question>
            <label topmargin=5 $auxDateLastAction
             text='<b>%tis.FirstNameDateLastAction </b>'>
          </question>"
     : DATETYPE
```

When the FirstName is updated, this field is also updated with the SYSDATE function. The Rules method is a .SHOW.

The last control to be discussed on the P-TIS screen is the button control on each telephone line. The button for the Best Phone is associated with the field named BestPhoneWhy which has the user defined type typPhoneStatus. The phone statuses include response, busy, no answer, answering machine, disconnected, no contact, dial and appointment.

```
BestPhoneWhy
    BASIL"<question>
           <button top=15 $auxDialButton
            caption='%tis.BestPhoneWhy'
            onclick='TracingProcedures.procUpdateDialStatus(""%tis.BestPhone"","
                     "BestPhone"",""^TName"")'>
          </question>"
     : typPhoneStatus
```

The button control for this field uses the attributes, caption and onclick. The caption is the text that is displayed on the button and for this implementation uses the category code of the typPhoneStatus. At startup, the field is set to Dial. The onclick attribute indicates the Maniplus procedure that is to be called when the button is clicked. In this case the procedure, procUpdateDialStatus in theTracingProcedures setup is called.

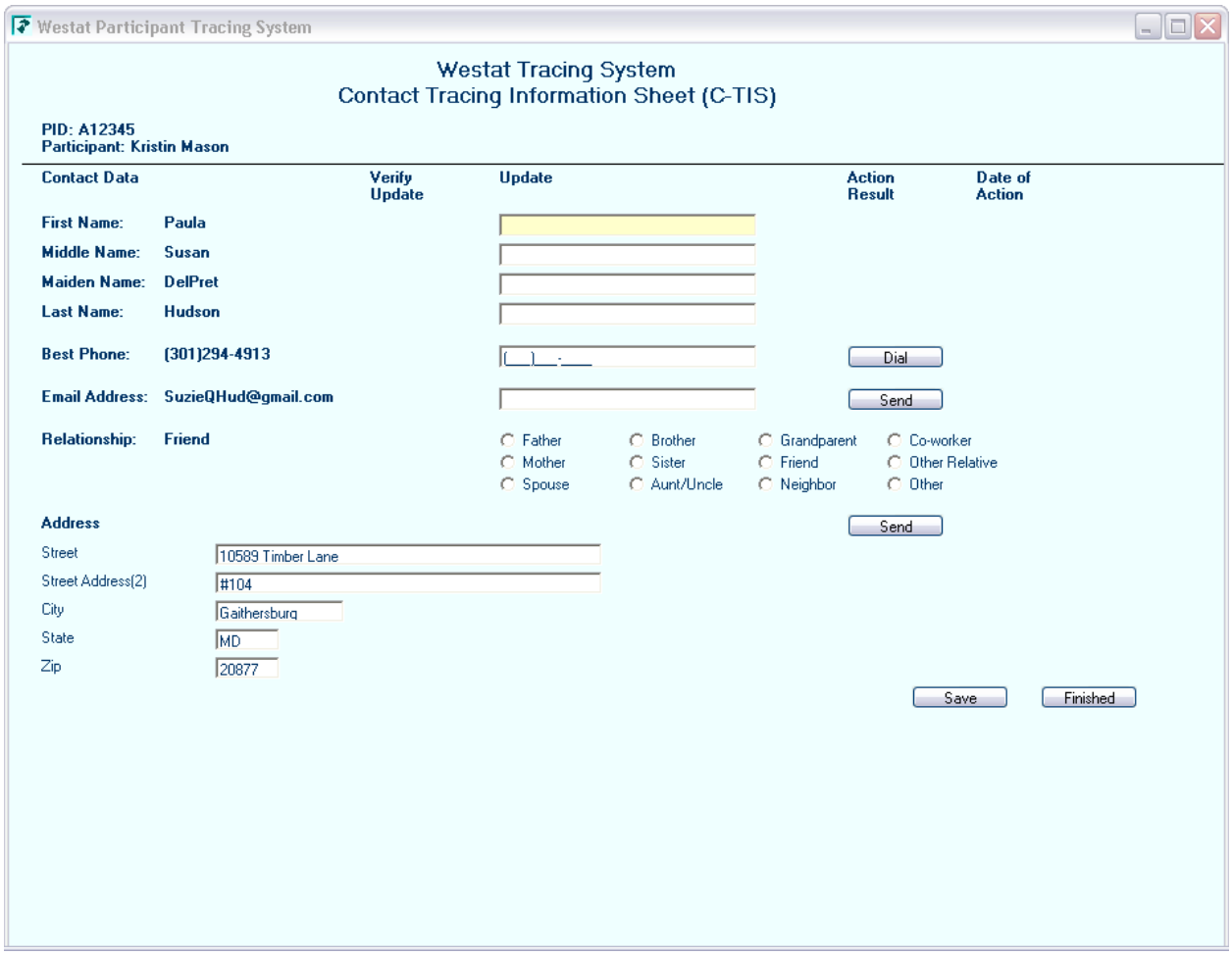


Figure 5. Contact Tracing Information Sheet

The Contact Tracing Information Sheet, as show in Figure 5, uses the same Basil instrument as the P-TIS and is stored in the same database. To differentiate between the participant and a contact, the primary key includes a contact number. A contact number of zero identifies the participant, any number greater is considered a contact.

In the P-TIS and C-TIS, an AUXFIELD is used to display the column headings.

```
EventLabel
    BASIL "<question>
            <line left=10 top=3 width=975 height=1 color=black>
            <label left=25 top=7 width=100 height=30 text='<b>^PartConTitle Data</b>'>
            <label left=280 top=7 width=100 height=30 text='<b>Verify<br>Update</b>'>
            <label left=380 top=7 width=100 height=30 text='<b>Update</b>'>
            <label left=650 top=7 width=200 height=30 text='<b>Action<br>Result</b>'>
            <label left=750 top=7 width=200 height=30 text='<b>Date of<br>Action</b>'>
          </question> "
    : STRING [1]
```

A separate label control was used for each heading in order to make it easier to align the headings with the fields. The line control adds the line above the headings just to give visual separation between the title information and the column headings.

The display of the Relationship field and of only one address is determined by the Rules when evaluating the ContactNumber. The Relationship field has the user defined enumerated type, typRelToParticipant. The input attributes for this control include tfillorder and columns.

```
auxRelationship
    BASIL "<question>
            <input  left=280 height=50 width=400 top=15 tfillorder=vertical columns=4>
           </question> "
    : typRelToParticipant
```

3.3.2 Interchange

Interchange and the Manipula extensions bring together a powerful combination allowing for real time access to data presented in a truly customized format. In the tracing system, there is a requirement to use external sources, such as directory assistance or the white pages, when all currently know sources have been exhausted. The tracer would record the numbers obtained from the external source and begin calling the numbers. We will use this requirement to show the relationship of Basil and Manipula.

Selecting the External Source button on the Select a Participant screen launches a Manipula dialogue , Figure 6, from which a tracer can select an external source.

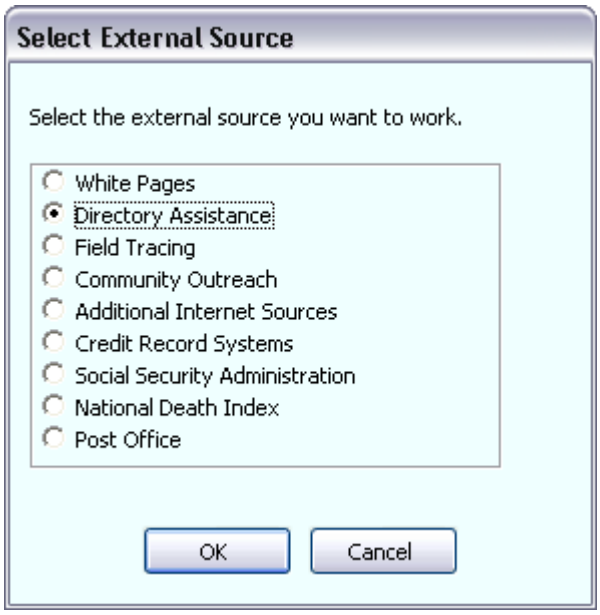


Figure 6. Select External Source

It was decided that the recording of the telephone numbers would be in a separate Basil instrument. The External Source Work Sheet was formatted to look like a popup and contained twelve rows for up to twelve telephone numbers. The primary key was consistent with the P-TIS and the Tracing master database, but also included a secondary key to identify the external source from which the telephone numbers were received.

After the selection of a source, a simple EDIT function call is used to launch the External Source Work Sheet instrument, Figure 7. This code follows:

```
auxResult := EDIT( 'TracingTWS /FTracingTWS /K' + auxTWSKey + ' /G /N ' +
                   '/Q=ParticipantName=' + '"' + auxParticipant + '"' +
                   ';tName=' + '"' + auxTName + '"',BASIL)
```

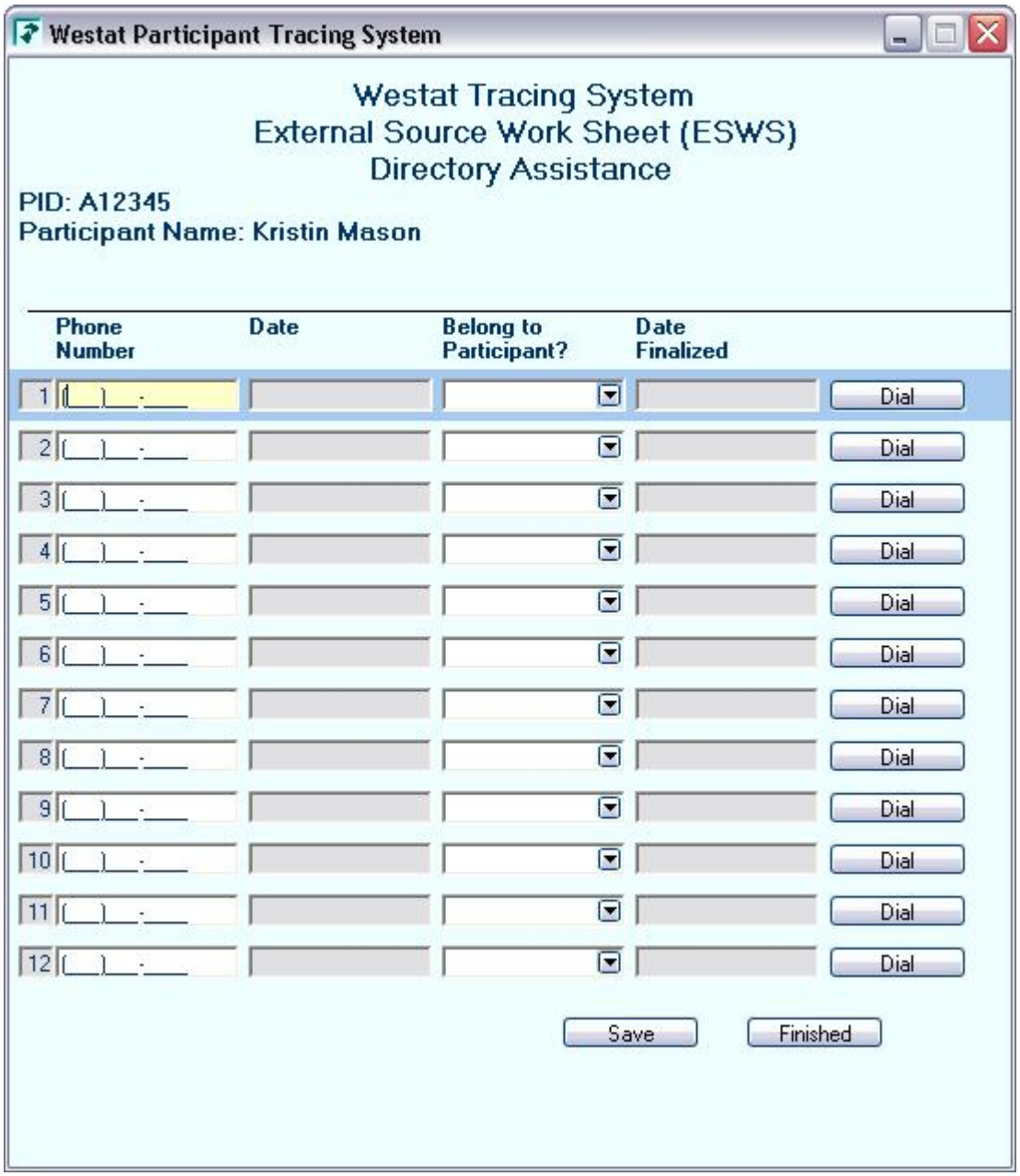


Figure 7. External Source Work Sheet

After collecting telephone numbers from the external source, the tracers begin calling each number. They will enter the telephone number and then select the Dial button for each number. The Maniplus procedure, procDialPhone, is called from which a dialogue will display the number to call, predefined text and a call result. See Figure 8. As can be seen from the following code, the DialNumber uses a button control to call the procedure procDialPhone in the TracingProcedures Manipula setup:

```
DialNumber
    BASIL"<question>
            <button $auxDialButton
              caption='%TWS.ContactSummary[^pSubScript].DialNumber'
               onclick='TracingProcedures.procDialPhone
```

```
                    (""%TWS.ContactSummary[^pSubScript].Telephone"",""%TName"")'>
      </question>"
  : typPhoneStatus
```

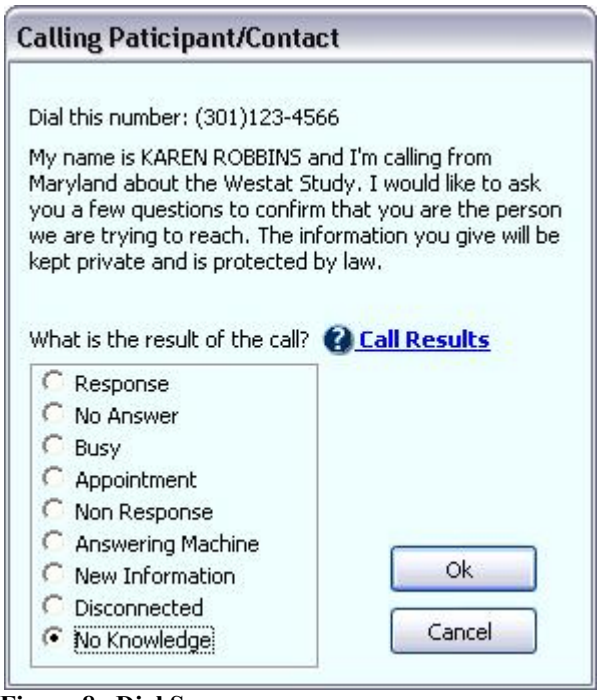


Figure 8. Dial Screen

After the call has been made and a result selected, the procedure will then use Manipula extensions to update the work sheet with the call result and the date. The Dial button will be changed to reflect the call result and the date field will now contain the current date. See figure 9.

Figure 9. Updated Work Sheet

The procDialPhone procedure uses two simple Manipula extensions, ACTIVEFIELD and PUTVALUE as the following code snippet shows.

```
aFieldName := ACTIVEFIELD
ufGeneric.PUTVALUE(aFieldName,STR(typPhoneStatus))
ufGeneric.PUTVALUE(aFieldNameDate,DATETOSTR(SYSDATE,DDMMYY))
```

Using the INTERCHANGE=SHARED Manipula setting allows for this communication (the transfer of data) between the Manipula setup and the Basil instrument.

Further examination of the Button control of the Basil shows the prefix $ in front of an AUXFIELD used for formatting and a % in front of the fully qualified field.

```
<button $auxDialButton
     caption='%TWS.ContactSummary[^pSubScript].DialNumber'
```

The Caption attribute of the DialNumber button is updated when the value of the DialNumber field from the procDialPhone is updated. The % in front of the field name ensures the caption attribute is updated with the current value of the field.

The standard Blaise fills (field names preceded by the ^ symbol) can be used but are evaluated during the execution of the rules. In Basil there are two alternative fill symbols available: the $-fill and the %-fill. They are both evaluated when Basil decides that they are needed and are both replaced by the value of the indicated field at the moment of evaluating.

4 Relational Database Storage

4.1 Blaise Datalink

The Blaise Datalink facility allows the data used by a Blaise data model to actually be stored in a non-Blaise format, such as Microsoft SQL Server, Oracle, or other relational databases. Datalink implements this functionality using Microsoft's OLE DB (Object Linking and Embedding, Database) technology. This means that it is possible to implement Datalink for any data source for which an OLE DB driver exists. As a practical matter, the type of Datalink system that we required is much easier to implement for a subset of the most common relational data sources, for which Blaise takes into account the characteristics of the particular data source, such as the maximum number of columns it can have in a single table. Blaise does provide such support for SQL Server, which is the database we are using for our tracing system.

4.2 Tracing History and Versioning

Management is always looking at how a tracer is progressing in the search for a missing participant. And a big part of this is looking at what has been done to date. In the past Blaise instruments were written to handle historical data as well as current data. This often posed challenges for developers. Using the Datalink Versioning feature removes the need for an instrument to keep track of historical information and uses the database to keep track.

We chose to use the Generic feature of Datalink with flat blocks and versioning as our MS SQL database configuration. Flat blocks mean each block in our Basil tracing instruments will correspond to a table in the SQL database. Generic allows for the sharing of tables across instruments. Instrument data is stored in a centralized way in only a few predefined and fixed table structures. Versioning is used to store multiple versions of a Basil record in the database. The versions are either current or historical. Each time a changed record is written to the database, the previous record is marked as 'historical' and the new record is inserted in the database and marked as 'current'.

The tracing system had a requirement to view all participant and contact address changes. When using Manipula to extract data or the Blaise Database viewer to inspect the data, only the current data is

extracted or displayed. To get the historical records you need to use other tools like SQL report writers or the Blaise Data Center. But each by itself is impractical in that they are not easily incorporated into the tracing system.

An alternative is to use the OLE DB Tool Box to generate a Blaise datamodel based on a SQL view. In this case a SQL view was created showing all the address changes for each participant or contact. Using the OLE DB Tool Box and the SQL view as input, a Blaise datamodel and a corresponding Boi file were generated. A Manipula lookup dialogue was then programmed to display the address history for each participant or contact. This technique would be used to view other record changes such as name and telephone numbers.

5. Conclusions

5.1 Lessons Learned

In the process of taking forms designed for a manual paper process and converting the process into computer data entry system using Basil and SQL databases we learned the following:

- When designing the input screens we learned the importance of creating Auxfields for the common formatting of Basil controls. For example, if you have a column of similar fields you probably want them all to be vertically aligned. You would create an Auxfield with the common attributes, such as, left, top, width etc. So in the Rules you would have:

  ```
  auxColumnAlign := 'left=10 width=100, top=15'
  ```

  where auxColumnAlign would then be used by the control for each field in the column. Thus, any changes are made in one place instead of in each control.
- The concept of a tracing system is different than the collection of survey data. Survey data are essentially collected at one time from one source, whereas, the tracing system must allow for the collection of disparate data at different times. Storing data in different Basil instruments allows modularizing code for easy maintenance and testing.
- During the development phase use Blaise databases instead of a SQL database. Until the Basil metadata is finalized there is no reason to do the extra work that would be involved in the frequent regeneration of the Datalink files and SQL tables.
- At the same time you are designing the instrument you should be building in lockstep the database load utilities. As you add/delete questions to the Basil instrument the load utilities should be modified accordingly. Thus, in the case of the tracing system, when development is finished the data sources for each participant are identified, the load utilities are up to date, and are ready to go at the click of a button.

5.2 Summary

Our impressions of using Basil to build the Westat Participant Tracing System prototype are overall, very good. Formatting the Basil instruments takes a little getting used to, but the ability to do your own formatting can result in a rich and cleaner looking user interface. Basil also features free form data entry and the ability to freely move from one module to another.

Large-Scale Survey Interviewing Following the 2008 WenChuan Earthquake

Zhang Huafeng and Jon Pedersen, Fafo, Institute of Applied International Studies, Oslo, Norway

Introduction

The May 12, 2008 Wenchuan Earthquake was one of the most devastating earthquakes in recent years, currently ranking as the seventh earthquake in terms of number of deaths caused since 1900 (U.S. Geological Survey, 2008)[1]. The Chinese government launched a major rescue effort, and a reconstruction plan — "The overall planning for post-Wenchuan earthquake restoration and reconstruction" (Government of China, 2008)— was published on September 23, 2008. One of the inputs to the reconstruction plan was the Post Wenchuan Earthquake Rapid Needs Assessment, a household and community survey carried out in July 2008 in 3652 households. In order to assess the reconstruction, a second survey was carried out one year later in July 2009 in the same communities covering 4014 households.

Both surveys were large-scale computer-assisted personal interviewing (CAPI) surveys. The surveys had their overriding purpose to assist the Chinese government on the provincial and national level in early and mid-term recovery planning. Especially the first survey operated on a very strict timeline as data were required for the development of the General Plan. The use of sub-notebook computer aided interviewing was essential for completing the surveys within strict timelines. Nevertheless, several challenges were also noted during the surveys.

The purpose of the present paper is to discuss some of the advantages and limitations of applying computer aided interviewing in the large-scale Rapid Needs Assessment survey and the subsequent Reconstruction Survey.

Advantages of the computer aided interviewing with Blaise

Description and choice of CAPI with Blaise

To provide the data for the Chinese government in the early reconstruction plan, the Post Wenchuan Earthquake Rapid Needs Assessment Survey had to be finished quickly. The go-ahead for the survey was given as late as June 20th, while the government asked that the initial report from the survey should be delivered by the 15th of July, 2008 in order for it to be used. At the same time, as the basic household data needed for sampling were not available from the provincial government, much time was needed for obtaining the data needed compiling a data frame. Therefore, there were severe limitations on the time that could be spent on the data collection, cleaning and analysis.

Computer-assisted interviewing undoubtedly benefitted the data collection. During the field work, each interviewer was equipped with an Asus Eee-pc sub-notebook ("net-book") computer running Windows XP and using Blaise 4.8 as a computer aided interviewing data entry tool (Statistics Netherlands, 2002). Computer aided interviewing was on the whole very successful. It saved the time needed for data entry, and perhaps most importantly the computer aided interviewing allowed for continuous tabulation and quality control while the survey was on-going. Moreover, as the data were sent from the field every day,

[1] The six earthquakes with higher death tolls are the 1976 Tangshan (China), the 2010 Haiti, the 2004 Sumatra (the Asian Tsunami), the 1920 Haiyuan, Ningxia (China), the 1923 Kanto (Japan) and the 1948 Ashgabat (Turkmenistan) (U.S. Geological Survey, 2010).

the source code for the production of the tabulation report could be developed as the survey progressed, taking account of real data. This made possible the production of a report before the deadline. Several different options for building the CAPI data entry applications, such as availability of software, the type of the hardware being used and the characteristics of the survey, were considered. The Blaise system had been used by the team for several previous PAPI surveys for data entry purposes; and therefore the interface and programming of the Blaise system was familiar to the team. Even though this was the first time the team used Blaise on a CAPI survey, the previous experiences benefitted the development of the instruments in the limited time available. As computers running standard Windows XP were used, standard Blaise could be used. Given that the data entry, especially for the first survey, had to be completed very quickly, the fact that a Blaise programmer can produce data entry without having to do detail formatting of data entry forms was also an important consideration. Furthermore, other researchers have reported that Blaise could be a useful and powerful environment not only for normal data entry applications but also for CAPI data entry applications (Rosemary Crocker, 1999).

Advantages compared to PDAs:

An alternative to using net-book computers would have been to use PDAs. These have successfully been employed in many small and large surveys, including in emergency situations (Pedersen, 2010). In general, available software for PDAs was at the time either cumbersome or difficult to use, expensive or too simple to handle a complex questionnaire. The cumbersome category includes software that supports more generalized data base systems, such as Handbase[2]. The expensive category included CSPRO-X, the commercial version of CSPRO. A free version of the CSPRO data entry system has since been released with implementation for PDAs. The too simple category includes software such as Pendragon Forms[3] that does not handle hierarchical files well.

The surveys in question had a relatively complex set of questionnaires, with sections for households, individual household members and a randomly selected person in the household. The survey covered the issues such as housing, infrastructure, basic demographic information, health situation, and mortality during earthquake, employment before and after the earthquake, household economic activities, economic support received, social network, and migration. One randomly selected member in the household was interviewed on their psychological health after the earthquake, participation in the rescue activities, trust in persons and institutions, attitudes to various forms of assistance, satisfaction with services and assistances. Even though there are apparent physical disadvantages with lap-tops, such as comparatively low battery life, heavier machine and so on, there are advantages of lap-tops over PDAs. The large screen and normal keyboard enables a friendly user interface. In particular, the larger screen of a net-book makes it possible to present more contextual information to the interviewer than what is possible on a PDA. While open-ended questions and qualitative interviews are nearly impossible with PDAs, this is not the case with the lap-tops. Formerly, PDAs had a definite price advantage over fully fledged computers. With the advent of net-books, this is no longer the case.

A final reason for choosing net-books rather than PDAs was that net-books, running vanilla Windows-XP are much better supported in the internet user community than PDAs. The same is true for Blaise, compared to PDA based software, perhaps with the exception of CSPRO.

Dramatically reduced the time needed for a survey:

The ability to collect high-quality data while dramatically reducing the time from collection to analysis could prove extremely critical in early stages of a humanitarian emergency. One apparent advantage of a computer-assisted survey is that neither paper questionnaire transportation nor data entry after the field

[2] http://www.ddhsoftware.com/handbase.html

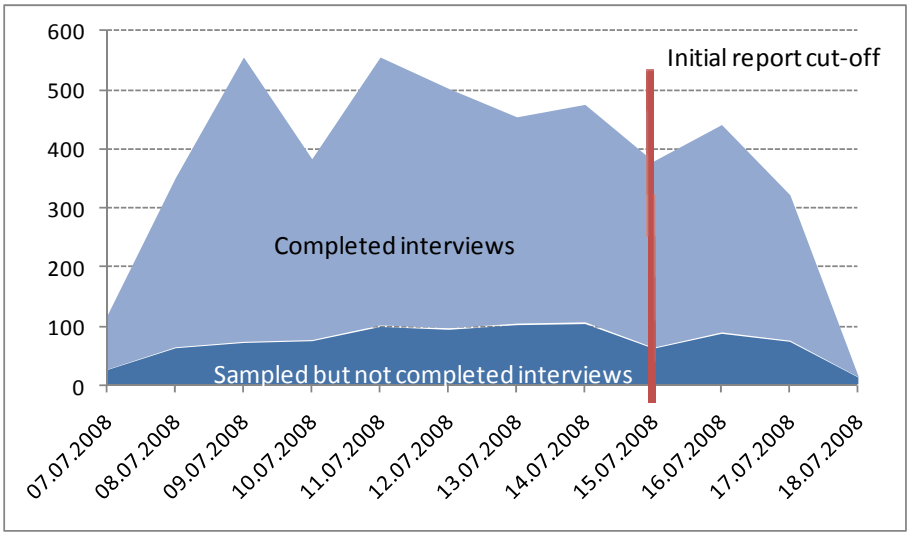
[3] http://pendragon-software.com

work is needed. Substantial parts of the work for data entry, data consistency checks and data cleaning were incorporated with the interviewing. The use of CAPI reduced the time needed for data cleaning, thus making more time available for data analysis.

The automatic skip and logical branching by the Blaise instrument not only improve the quality of the data, but also reduce the interview time (D. Forster, 1991). Even though we did not formally test how much time was saved by computer during the interview, there was evidence indicating that it did reduce the interview time. During the pilot, experienced researchers used paper questionnaires and reported that it took on average an hour to complete one interview. However, in the first field practice during the training, most trainees managed to complete the interview in less than an hour.

Even though a computer-assisted survey increases the technical requirements of the field work operation, it simplifies survey management. Paper questionnaire management and questionnaire transportation are not needed. The need for field editing is also reduced, as almost all the logical errors have been controlled by the data entry application during the interview. This will be discussed later.
All these features have contributed to a quick and high quality survey. The field work of the survey formally started on 7th July 2007, and the data from 2739 (75%) households were collected by 14th, July. Figure 1 shows the progress of the field work, and the first initial report was submitted to the Chinese government on 15th, July 2008. It would have been impossible if CAPI had not been used for the interviews.

Figure 1 Field work in the Post Wenchuan Earthquake Rapid Needs Assessment Survey



Technical benefits and advantages:

Several features of Blaise were found to be beneficial in facilitating the interviews during the field work. Interviewers navigated the lengthy survey much more easily on the electronic questionnaire. The Blaise questionnaire displays one question at a time, and only advances to the next question once the current question is answered. The Blaise data entry application keeps automatic track of which question should be answered next. Automatic routing also made it impossible to inadvertently skip a question, or ask a question which should have been skipped. CAPI eliminated most skipping errors, and is also reduced the missing errors (Norman M. Bradburn, 1991). The program could also limit response choices on subsequent questions depending on answers to prior questions, thereby avoiding the long listings of answer choices. In addition, the data entry application always interprets the logical branching correctly, without any effort on the part of the interviewer. For example, when some questions are only designed for

a specific age-group, Blaise can be designed to automatically show the questions only when the person in question is eligible.
A well designed Blaise program also performs internal consistency checks. Such checks include range checks and consistency checks. Range checks compare the given response to the range of possible responses. Consistency checks are more complicated checks, which analyze the internal consistency of several responses. In the case of surveys that use paper questionnaires, internal validity checks have to be conducted at the data cleaning stage that usually follows the data collection stage. Then when errors are detected, they can often only be recoded to a missing data code because it is no longer possible to ask the respondents what they really meant. However, a computer-assisted survey provides an opportunity to carry out range checks and consistency checks when the data are entered, and forces the interviewers to stop and check their mistakes. These factors are likely to eliminate many of the most common data entry errors and missing data. With a paper questionnaire, survey administrators have to spend much time training the interviewers on how to reduce data entry error, and a field editor is usually needed to detect these errors in the field. Therefore, using CAPI saves a lot effort in training and field editing, and yields higher quality data. The computer-assisted survey has a significantly lower overall error rate and it also significantly reduces the inter-individual variability in the data accuracy when compared to the paper version survey time (D. Forster, 1991).

Another advantage of CAPI is that it can improve random selection of respondents within the household. Both the Sichuan surveys included a questionnaire targeting a randomly selected adult household member about his/her psychological status, attitudes and opinions. In a normal PAPI (Pencil-and-paper interviewing) survey, after the information for all the household members is collected, the interviewer selects one member based on some predetermined random procedure, such as a Kish table or the household member who had a birthday most recently. If the interviewer strictly follows the instructions, there will be no selection bias. However, interviewers tend to select household members who are at home so as to avoid extra effort to find the selected member, and may also have preferences with regard to age and gender. CAPI makes it possible to design the program so that one household member is selected by the program randomly. It not only saves lots of effort by the interviewer to follow the instructions for selection, but also reduces the potential bias during the random selection.

Quality assurance during field work:

With paper questionnaires, some days or weeks are normally required to enter and clean data before the survey managers can evaluate the quality of the field work. In many cases, it might be already too late if serious problems are found. Without the data entry phase, the computer-assisted survey makes it possible to carry out rapid data analysis and quality control every day right after the field work. The result of initial data analysis and feedback on the performance of each interview team and each interviewer could be sent back to the field in time, so that the quality could be improved to the largest extent possible. It turned out to be extremely useful especially in the early stage of the field work.
A benefit of CAPI systems is that they generally allows information about the interview process itself to be recorded. This can be used both to monitor the behavior of the interviewers and to gain data on how individual questions work – for example if a question normally takes much more time than others to complete. The audit system of Blaise is extremely extensive, - a main problem with it is that it easily drowns one in data,

There are other potential benefits which might improve survey quality when using CAPI. Notes and the interviewer manual can be incorporated into the Blaise data entry program. It was not done in the need assessment survey due to the limitation on the time for the application development; however it would be useful for the interviewers to be able to search for help any time they want. It would also be possible to exploit other capabilities of the computer such as recording sound, thus enabling direct storage of the verbal interview itself. This might be of help in resolving inconsistencies detected during data analysis or cleaning.

Limitations of computer aided interviewing with Blaise

Hardware limitation and challenges in the field:

One of the main constraints with lap-tops when we carried out the first survey in 2008 was the limited time of operation on a single battery charge. In some areas seriously hit by earthquake, the electricity supply had been destroyed. At that time the best battery for the Asus Eee-pc sub-notebook we could obtain only allowed three hours of operation. Although every interviewer was equipped with a spare battery, this was not enough for an eight-hour working day. Therefore some interviewers were forced to use paper questionnaires towards the end of the day. However, even in 2008 the areas not supplied by the electricity were not very common so the limited battery capacity did not become a major problem. In cases where both batteries were out of power and no electricity could be had from the household, the interviewer would use the paper questionnaires, and enter the data at night when they could charge the lap-top. However, this constraint with CAPI should disappear with the improvement of technology. In fact, it is quite possible today to get net-books with a much longer battery life than what we could get in 2008.

Yet another challenge of CAPI is data loss due to hardware failures or human error in using the computers. Therefore extra work with data management and equipment maintenance is needed during the field work. In the Sichuan surveys, the program was set to automatically save every minute. Each team was instructed to copy the data to the supervisors' USB, and the data were sent by email to the headquarters every day. After the data were sent to the headquarters, copies of all the data were still kept on interviewers' lap-tops and supervisors' USB during the whole field work, therefore the loss of data in any one source would not be a catastrophe. Before the start of the survey equipment maintenance was one of the worries of the survey team. One technician was available at all times by phone in case a technical problem arose. All the interviewers always carried a few paper questionnaires in case the lap-top crashed. Only one lap-top of the 80 in use crashed in the first survey, and was replaced by the vendor immediately. In the second survey two lap-tops crashed during interviews, and paper questionnaires were used that day. Possible crashes were planned for, so the crashed net-book was replaced within the day. In neither of the surveys were any data lost due to technical problems with the computers.

Screen readability presented a problem in the field, particularly in bright sunlight. After the first survey, feedback from the interviewers was that the font chosen for display was so small that they had difficulty in reading the screen. This was then improved in the second survey. Standard Blaise is flexible in altering the font size and characteristics for different part of the questionnaire, and increasing the font size helped to some extent. However, the Asus Eee-pc sub-notebook only has an 8' inch screen, and to increase the font size caused some questions to require more than one page to show all answer categories. The interviewers found this not only inconvenient during the field work, but were unsurprisingly also prone to forget the invisible answer categories.

Programming challenges

Due to the time limitations on the surveys, the Blaise data entry application had to be developed and tested in parallel with the development of the questionnaire. This required good organization and cooperation between the programmer and the questionnaire designer. After the first draft of the questionnaire was sent to the programmer, the questionnaire administrator started to log all subsequent changes. When part of the program was ready, it was given to all the team members every day for testing. In both surveys, when the questionnaire was finalized, the data entry system was ready for the training. Nevertheless, some errors were also found by the trainees during the training.

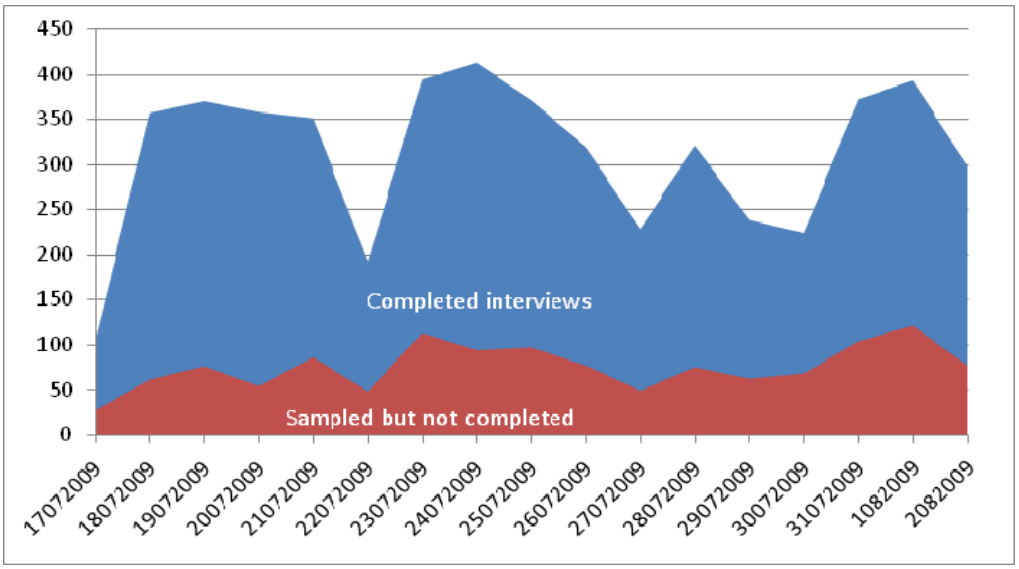
While one would like to have the CAPI program completely correct by the time the field period begins, it is likely that some bugs will show up during the actual field interviewing even with the most rigorous testing beforehand. This is of course also a sad fact of life as regards pencil-and-paper interviewing questionnaires. The possibility of correcting errors after the field period begins is relatively small with CAPI, but, not impossible. Fortunately, the errors are normally detected in the early stage of the field

work. When the field work started, the teams were sent to the same area in the first two days and they spent the first two nights in the same place, so that it was possible for the program to be updated when errors were found. In the first survey, a problem was only found on the first day of the field work and the program was modified and updated on all the 80 computers in the field. The strict rules that Blaise enforces for changing the data entry application when data have been entered are actually quite beneficial here, because it forces the data entry programmers to ensure that compatibility of data files are taken seriously and that changes are implemented in a controlled manner.

Mis-keying errors

Even with all the benefits CAPI has provided to improve the data quality, we found that the mis-keying errors during interviewing do exist and are not easily detected. As mentioned before, the Blaise data entry application made internal validity checks through the whole interviews. Some mis-keying problems can be partly solved when the out-of-range data or inconsistencies are detected. However, many mis-keying errors cannot be detected with internal validity checks, as they constitute logical answers. In PAPI surveys, such errors are also inevitable during field work and the data entry phase. Unfortunately, there is little data that can be used to compare the occurrence of mis-keying errors between CAPI and PAPI. However, with carefully designed data entry, such as double entry, it is possible to minimize the mis-keying errors during the data entry stage in PAPI surveys. Thus, while the CAPI method has the advantages of integrating several survey steps into one single activity, it also has its disadvantage of introducing more undetectable keying-errors during the field work. The second Sichuan survey used the same sample as the first survey, down to the household level, thus creating a panel. This made it possible to find out the mis-keying errors which were not easily detectable when just considering a single survey. The second survey in most respects followed the design of the previous survey. The questionnaire was changed somewhat, in that the questions that most directly related to the immediate consequences of the disaster was removed, and questions relating to the reconstruction were added. The exception for this was households living in camps, where a new sample was drawn. Camp inhabitants could not be followed from one survey to the next because some camps had not been established at the time of the first survey, and many of those that had been constructed had already been dismantled by the time of the second.

Figure 2 Field work in the Reconstruction Survey



The data from the rapid needs assessment survey and the following reconstruction survey indicated that the interviewers committed more mis-keying errors when they were under pressure. The first survey was carried out two months after the earthquake. To provide data following the government planning schedule, the first survey followed an extremely strict timeline. There were only six and a half days' for field work

before the initial report, with 75 percent of the data collected; and the team spent another three extra days after the initial report to finish the remaining interviews. In the first stage before the initial report, the team completed on average 365 interviews per day; and in the second stage, on average 304 interviews per day. In 2009, the second reconstruction survey took place July 17 to August 2, 2009. Figure 2 shows the field work progress for the second survey. With the same number of 80 interviewers, the team completed on average 243 interviews per day.

Partly due to the exceptional migration after the earthquake, and partly because the panel was not planned when the first survey was carried out, it was a challenge to revisit the households interviewed in the first survey. When the data from the two surveys were matched, household members' basic information was also manually checked to confirm the match. In all, 2030 households and 6433 people were matched. However, the inconsistencies of the household members' gender, age, residency status and etc. between the two surveys indicated that there were mis-keying errors during the field work. The mis-keys of age could not easily be corrected, while gender and residency status often could be corrected with the assistance of other information in the survey. People's name, relationship with household head and other information were combined to manually correct people's gender and residency status. When no information could support people's gender and residency status, random imputation was applied, but this was only used in a few cases. Around 90 cases (1.4%) were found to have mis-typed gender, and 86 cases (1.3%) have mis-typed residency status in the first survey; while (0.5%) mis-typed gender and 24 (0.4%) mis-typed residency status in the second survey. Apparently, there were more mis-keying errors in the first survey.

When interviewers have to concentrate on both interviewing and data entry, especially when they are under pressure (for example because of reluctant respondents), mis-keys are probably more frequent than with the traditional Pencil-and-Paper Interviews. Extra work in the program design had to be done in order to avoid errors.

Strategies for avoiding errors depend to some extent on how the interviewers use the computer. If the keyboard is used, then the mis-keys can probably be minimized by assigning values a bit farther away from each other for the questions such as gender or "yes/no" questions, i.e. assigning 0 and 1, instead of 1 and 2. If the mouse is used, increasing the font size and adding extra space between the answers may reduce the errors. Finally, auto-skip in Blaise is often used for the data entry with PAPI, so as to avoid extra typing. However, auto-skip with CAPI probably introduces more mis-keying errors, since interviewers have less chance to double check what was typed.

Mis-keying of large numbers

Apart from the general mis-keying errors, mis-keying of large numbers was found to be especially common and may seriously affect data quality. Normally, there is no easy way to identify such mis-keying errors, as long as the data are in the reasonable range. However, during training we found that the CAPI facilitated the test on the performance of the trainees, and helped pin-point the weak parts where most errors were found. We had a mock interview between the trainees and the trainer during training, and at the same time all the trainees were supposed to enter the interview into their computers. After the mock interview, all the data were gathered and analyzed. It took very little time to find out the performance of each interviewer and the common problems. One of them was the problem in entering large values correctly. A very high percent of the trainees have mistyped the values of income, fees and cost (table 1). It turned out that the errors were not only due to the mis-keys but also due to the misunderstanding of the reported unit. As table 1 indicates, the mis-keys are far from random – rather they introduce a definite downward bias for large numbers and an upward bias for small numbers. Furthermore, the mis-keys of the large values occur more frequently than other questions; and the outcome of the mis-keying affects the data much more seriously than other questions.

Table 1 Mis-keying errors found during the training[1]

	Percentage of trainees who answered correctly	Correct answer (Yuan)	Mean of all the answers (Yuan)
Family business income	62.3 %	120,000	86,589
Living assistance from government	81.8 %	2,700	8,400
Personal income in the past one year	79.5 %	14,400	20,209
Annual salary in the past one year	77.6 %	120,000	99,342
Current monthly salary	96.1 %	10,000	9,776
School fee in the past one year	61.0 %	5,000	5,319

[1] The table was calculated based on 76 trainees

Although erroneous entry of large numbers is also a problem for many PAPI surveys, with CAPI surveys, extra safe guards may be added to reduce the impact. When the problem was identified during the interviewer training, the data entry system was revised, so that a warning box showed up after each such large value entered, and the large value was translated into values in Chinese character to be confirmed (for example, 10,000 was translated into ten thousand in Chinese). Although it adds extra work for the interviewers to double check the value they enter, it is worthwhile because it helps reduce the risk of mistyping.

Conclusion

Blaise turned out to be an effective tool for developing data entry applications rapidly for CAPI use in emergency settings. The CAPI approach eliminated the lag between field work and reporting. This was essential for the policy use of especially the first Sichuan survey. Net books served as an excellent platform as they are lighter than ordinary laptops and easier to use and more versatile than PDAs. The ease by which they can be programmed using standard data entry software such as Blaise was a major point in their favor.

The flexibility of the Blaise data entry programming language and the automatic handling of screen lay-out in the Blaise data entry module made concurrent questionnaire development and data entry programming possible. For ordinary surveys concurrent development is best avoided, but for time pressed settings, such as those of the first earthquake survey, it is unavoidable. Rather than be a hindrance, the strict version and type control on the data structure inherent in Blaise facilitated the rapid development.

Bibliography

D. Forster, R. H. (1991). Evaluation of a computerized field data collection system for health surveys. *Bulletin of the World Health Organization 69 (1)* , 107-111.

Government of China. (2008). *The overall planning for post-Wenchian Earthquake Restoration and Reconstruction.* Beijing: Planning Group of Post-Wenchuan Earthquake Restoration and Reconstruction of the Earthquake Relief Headquarters under the State Council.

Norman M. Bradburn, M. R. (1991). A Comparison of Computer-Assisted Personal Interviews (CAPI) with Paper-and-Pencil Interviews (PAPI) in the National Longitudinal Survey of Youth. *Presentation at the annual meetings of the American Association for Public Opinion Research in Phoenix, Arizona* .

Pedersen, J. (2010). The methodology and design of the post Wenchuan Earthquake Rapid Needs Assessment . *International Disaster Reduction Conference IDRC 2010.* Davos, Switzerland.

Rosemary Crocker, R. E. (1999). *Computer Assisted Personal Interviewing Solutions in Australia.* Australian Bureau of Statistics: ESCAP workshop on introduction of new technology to population data, background paper for session 3.3.

Statistics Netherlands. (2002). *Blaise Developer's Guide.* Heerlen: Statistics Netherlands.

U.S. Geological Survey. (2008). *Earthquakes with 1,000 or More Deaths since 1900*. Hentet 12 2, 2008 fra U.S. Geological Survey Earthquake Hazards Program: http://earthquake.usgs.gov/regional/world/world_deaths_sort.php

Multi-Center Collaborative Survey Research with Blaise

Jim O'Reilly, Westat

1. Introduction

A number of important research projects using Blaise have been conducted as collaborations of investigators. Some are structured group enterprises from the outset. Others arise when the original researcher's work is identified as valuable and applicable in other settings, and is then applied elsewhere. Both models extend the reach of the research method in important ways.

This paper looks at six of these research projects primarily in terms of the research scope and application, how the collaboration was structured, and any special Blaise aspects of the work. Our goal is primarily to make the members of the Blaise community aware of the broad range of collaborative research using Blaise.

2. World Mental Health-Composite International Diagnostic Interview

The WMH-CIDI is the most widely used Blaise application in collaborative research. The survey is designed "to obtain valid information about the prevalence and correlates of mental disorders in the general population, unmet need for treatment of mental disorders, treatment adequacy among patients in treatment for mental disorders, and the societal burden of mental disorders." (Kessler & Ün̈stün, 2004)

The CIDI survey instrument is made up of a screening module and 40 sections covering diagnoses, functioning, treatment, risk factors, socio-demographic correlates and methodological factors. Kessler & Ün̈stün (2004) describe in depth the instrument's background, components, validity assessments, methodological research, diagnostic sections and assessments of risk factors, consequences and treatment.

A central issue with psychiatric diagnostic survey instruments conducted by interviewers who are not clinicians is the validity of the diagnostic assessments. Kessler & Ün̈stün describe the major methodological problems--question comprehension, task comprehension, motivation, and the ability to answer accurately--and adaptations of the survey instrument and interviewer training to address them.

The large-scale, cross-national implementation of CIDI was directed by the WMH Survey Initiative centered at the University of Michigan's Institute of Social Research. A key challenge was how to reconcile "research traditions of individual countries, which vary widely in methodological rigor" and a "one-size-fits-all methodology that naively imposes the same procedures and protocols across all countries and cultures" (Pennell *et al.*, 2008).

Pennell and colleagues describe the WMH-CIDI collaboration based on the experience of the 17 countries that had finished data collection and processing by early 2007. They discuss design and methodological considerations, development of the CAPI and PAPI versions, pre-testing, interviewer recruitment and training, field structure and implementation, interviewer effects and interview privacy, field supervision and quality control, and data preparation.

There were four major revisions of the baseline English interview schedule from 1999 to 2007, necessitating the implementation of a "comprehensive set of version control procedures to allow for comparisons across countries" (Pennell *et al.*, 2008 p. 37). Each country adapted the English version to add country-specific content.

The core Blaise CIDI instrument was developed by Karl Dinkelman of ISR. A few years ago, Dr. Ronald Kessler, the lead investigator of the overall CIDI research enterprise, moved from Michigan to Harvard

Medical School, and the responsibility for maintenance and development of the ongoing Blaise CIDI work also shifted from ISR to Harvard (Chardoul & Pennell, 2010).

Alison Hoffnagle (2010) manages the CIDI's continuing development and maintenance at Harvard Medical School. Hoffnagle said version control and monitoring changes and modifications of the baseline English CIDE are key activities. Among the changes is greater modularization to allow investigators who are interested in certain elements to select and implement only those sections. As well, changes are underway to CIDI to include the latest DSM and ISM disease and related health problem diagnoses. Harvard also developed a household listing application for laptop case management for a study in China and now provides it to all CIDI users.

CIDI training centers have been funded in Netherlands and Spain, and those groups update the Dutch and Spanish versions of the current standard CIDI. Older versions of CIDI in a number of other languages are still available from Harvard and Michigan. Also, arrangements can be made with Harvard to support translation and application of CIDE in other languages and countries. Two such efforts are currently going on in Poland and Saudi Arabia. The Saudi version involves extensive modifications of the core instrument. Harvard is designing and programming the English version while a team in Saudi focuses on the Arabic translation (Hoffnagle, 2010).

Michigan's CIDI Training and Reference Center offers a summer CIDI training course for principal investigators, study managers, and data analysts, and at other times a three-day CAPI training course is offered (Chardoul & Pennell, 2010). The number of follow-on CIDI research projects is approximately 40 in North America.

3. PRISM

PRISM, the Psychiatric Research Interview for Substance and Mental Disorders, is also a psychiatric epidemiology survey. It is focused on psychiatric diagnosis when subjects/patients drink heavily or use drugs. The PRISM assesses DSM-IV disorders and differentiates primary disorders, substance-induced disorders and the expected effects of intoxication and withdrawal (PRISM, 2010).

The PRISM method was developed by Dr. Deborah Hasin of Columbia University and the New York State Psychiatric Institute. Chistina Aivadyan has been responsible for Blaise development and related activities of PRISM. Aivadyan said that originally PRISM was a paper instrument with an average interview time of about two hours. Interviewers had difficulty executing the challenging instrument in PAPI without significant errors (Aivadyan, 2010).

Dr. Hasin used Blaise successfully for a study in Israel conducted in Hebrew and Russian—"Alcohol in Israel: Genetic and Environmental Effects". This lead to the decision to use Blaise to computerize PRISM. Several years were spent programming and testing the PRISM-CV (computerized version). In the last year, a certified version has been released, training has been conducted, and the PRISM has been distributed to users—to US and international medical and academic research centers. Many of these organizations had been using the PAPI PRISM.

The gains from using computer-assisted interviewing and Blaise are the familiar ones—complex branching, modularity, error minimization and correction, more standard interviewing techniques, and superior data quality. Average interviewing time dropped to 70 minutes compared to PAPI (Aivadyan, 2010).

The ability to modularize the survey has enabled clinical researcher to apply just key sections related to diagnostic measures for screening purposes.

PRISM-CV is in English. Versions in Spanish (Castilian) and Norwegian are being developed by groups in Spain and Norway.

Dr. Hasin's team at New York State Psychiatric Institute has developed the supporting materials and processes to enable other research teams to apply PRISM. These include a Users Guide, Training Manual, training course, and certification process.

Development is continuing in a number of areas: integration of DSM-5 changes as they come out, and a number of significant measurement and methodological studies of PRISM and other research approaches to substance abuse and psychiatric disorders (Aivadyan, 2010).

4. Collaborative Study of the Genetics of Alcoholism (COGA)

COGA is a national collaborative study addressing the familial transmission of alcoholism and identifying susceptibility genes using genetic linkage. COGA includes nine different centers where data collection, analysis, and storage take place. The study has been supported since 1989 by the National Institute on Alcohol Abuse and Alcoholism (NIAAA) and the National Institute on Drug Abuse (NIDA).

The COGA web site describes the study: "The COGA investigators, in the course of carrying out their own genetic linkage studies of alcoholism, have assembled a collection of >300 extended families densely affected by alcoholism, consisting collectively of >3000 individuals. They have collected extensive clinical, neuropsychological, electrophysiological, biochemical, and genetic data, and established a repository of immortalized cell lines from these individuals, to serve as a permanent source of DNA for genetic studies. In order to promote the most rapid possible progress in identifying genes influencing vulnerability to alcoholism, NIAAA is supporting COGA's distribution of these data and biomaterials to qualified investigators in the broader scientific community." (NIAAA, 2010)

Interviewing and testing of COGA families is conducted at six university centers: SUNY Downstate Medical Center in Brooklyn NY, University of Connecticut, Indiana University, University of Iowa, University of California in San Diego, and Washington University in St. Louis. The scope and scale of COGA necessitates a complex data management system. Washington University is the overall data management and repository coordinating center (Washington University St Louis, 2010).

Nan Rochberg (2010) has done virtually all Blaise programming for the COGA project, as well as converting the Blaise interviews into SAS datasets, and publishing and maintaining the compiled datasets. She developed the diagnostic algorithms for the interviews and is responsible for many coordinating data issues. Rochberg is currently in the process of modifying one of the COGA interviews for use by a group at the University California, San Diego (Rochberg, 2010).

Blaise was chosen when it was time to move the interviewing to CAI more than ten years ago because the university had already used it successfully in a similar study of the genetics of nicotine dependence.

Data collection takes place in clinical and research settings. As a family study, people are first ascertained for eligibility in a clinical setting, then interviewed in a research space. It starts with someone in a hospital or other treatment facility for alcoholism. There are a clear set of rules on enrollment, particular family structure and the number of siblings in a family. As a prospective study, it looks at children and adolescents up to age 25 and asks questions about many other psychological disorders.

The Blaise instruments include an adult interview, two versions of the child interview with many as five children, and a family history. There are 3500 to 4000 variables per case.

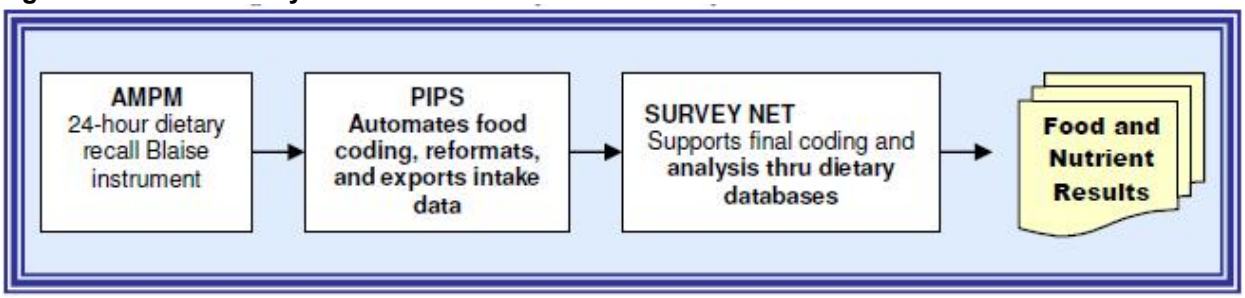
The COGA project website lists more than 225 scientific publications based on the study.

5. Automated Multiple-Pass Method

The Automated Multiple-Pass Method (AMPM) was developed by the USDA Food Surveys Research Group for their dietary survey. It collects 24-hour dietary recalls for the US National Health and Nutrition Examination Survey. During the interview, individuals recall the foods and beverages that were consumed the day before the interview. Details about each food and beverage are collected along with a description of the amount consumed, the time of day the food was eaten, the name of the eating occasion, and where the food was obtained.

The structure of the AMPM system is described by Anderson and Steinfeldt (2004). From the AMPM Blaise database, the intake data are extracted, reorganized, automatically coded, and reformatted by the Post Interview Processing System (PIPS) to be used by the computer-assisted food coding and dietary data processing system (Survey Net). Trained food coders use Survey Net to search the Food and Nutrient Database for Dietary Studies (FNDDS) to code foods and amounts reported during the AMPM dietary recall interview. Government agencies and other researchers use the final data to examine nutrition and food safety issues affecting the US population.

Figure 1. AMPM Data System



The Blaise instrument has more than 2,500 questions and more than 21,000 responses. “The capability of the AMPM Blaise instrument to ask only pertinent questions and collect all the necessary details for a particular food is a critical feature for implementing automated coding.”

With its proven ability to collect, organize and code dietary information, the AMPM application and the supporting food coding database and PIPS and Survey Net system have attracted significant interest from other researchers.

Alanna J. Moshfegh (2010), research leader of the USDA Food Surveys Research Group, said the system was developed for internal use and when others began requesting access to it “we took the philosophy driven by limited resources that if others could use it as it was, we would be happy to enter into appropriate agreements for that. But we would not program or adjust it in any way for that kind of use. With that said, it has been used by other organizations.”

Moshfegh said they try to be clear to those inquiring about the system that it’s not something you can simply download by email or run on a thumb-drive. Rather, significant investment is required to realistically implement the system—an IT specialist, at least 100 interviewers, three to four days of training at the USDA, and a signed agreement. A number of outside organizations have conducted research using AMPM, as shown below.

Table 1. Research by external organizations conducted using AMPM

Collaborators	Project/Study	Status
Statistics Canada	Canadian Community Health Survey	Completed
National Center for Health Statistics, Westat	National Health and Nutrition Examination Survey (NHANES)	Ongoing
National Institute on Aging, National Institutes of Health	Healthy Aging in Neighborhoods of Diversity Across the Life Span (HANDLS); 20 years duration	Ongoing, Longitudinal study
University of Maryland School of Medicine, Johns Hopkins University Bloomberg School of Public Health	Mothers in the Special Supplemental Nutrition Program for Women, Infants, and Children (WIC) Program in Baltimore, Maryland	Completed
Food and Nutrition Service, USDA	School Nutrition Dietary Assessment Study-III (SNDA III)	Completed
National Cancer Institute, Westat	Food Attitudes and Behaviors Survey	Completed
National Cancer Institute, National Institutes of Health	Automated Self-Administered 24-Hour Recall (ASA24) adapted AMPM format and design for use of ASA24	Ongoing
Pennington Biomedical Research Center	Preventing Obesity Using Novel Dietary Strategies (POUNDS LOST) intervention study	Completed
Environmental Protection Agency, Colville Confederated Tribes, Westat	Upper Columbia River Tribal Use and Consumption Survey	Ongoing
Research Triangle Institute International	Healthy Eating and Active Living in TRICARE Households (HEALTH) Intervention Study for the Department of Defense	Completed
University of Maryland School of Medicine	Toddlers Overweight Prevention Study (TOPS). Conducted with WIC participants	Completed
University of Vermont	Relationship between television viewing and eating	Completed

6. Pathways to Desistance

Pathways to Desistance is a longitudinal study of serious adolescent offenders as they transition from adolescence into early adulthood. Beginning in 2000 1,354 adjudicated youths from the juvenile and adult court systems in Maricopa County (Phoenix), Arizona and Philadelphia County, Pennsylvania were enrolled. The 14- to 18-year-old youths were predominantly found guilty of felonies, along with some non-felony sexual assault or weapons offenses.

Each was followed for a period of seven years past enrollment, with a baseline interview plus followup interviews every six months for three years and annually thru the seventh year. The study provides a

comprehensive picture of life changes in a wide array of areas over the course of this time. The study was designed to:

- Identify distinct initial pathways out of juvenile justice system involvement and the characteristics of the adolescents who progress along each of these pathways.
- Describe the role of social context and developmental changes in promoting desistance or continuation of antisocial behavior.
- Compare the effects of sanctions and selected interventions in altering progression along the pathways out of juvenile justice system involvement (University of Pittsburgh Medical Center, 2010).

The University of Pittsburgh is the project coordinating center. Blaise instruments were developed there and distributed to the collaborating organizations—Temple University in Philadelphia and Arizona State University in Phoenix. Data collection ended in April of this year and analysis and interpretation of the research is now underway. A total of 20,000 interviews were completed, including 19,000 subject interviews. The study has been described as the "largest longitudinal study of serious adolescent offenders ever done" (University of Pittsburgh Medical Center, 2010).

Carol Schubert, the study coordinator, said important findings from the study have already been published, with more to come based on the entire body of data. The sponsors and researchers believe "it should have a fairly significant impact on the field" (Schubert, 2010).

7. SHARE - Survey of Health, Ageing and Retirement in Europe

The Survey of Health, Ageing and Retirement in Europe (SHARE) is a multidisciplinary and cross-national panel database of microdata on health, socio-economic status and social and family networks of more than 45,000 individuals aged 50 or over. SHARE is designed to "deliver truly comparable data, so we can reliably study how differences in cultures, living conditions and policy approaches shape the quality of life of Europeans just before and after retirement.". (Börsch-Supan and Jürges, 2005). Data collected include variables on health, bio-markers, psychology, economics, and social support. Twelve countries participated in Wave 1 (2004). Others countries were added in 2006/7 and 2008/9.

Coordinated centrally at the Mannheim Research Institute for the Economics of Aging (MEA), SHARE is harmonized with the U.S. Health and Retirement Study (HRS) and the English Longitudinal Study of Ageing (ELSA). CentERdata of Tilburg in the Netherlands partnered in day-to-day management of the project as well as having primary responsibility for the development of the Blaise survey instrument and a number of related systems, most importantly the Language Management Utility and the Case Management System.

The LMU language translation system provides a comprehensive platform to support the translation of the generic English-language Blaise instrument into other languages. As well, the system generates the new language version of the Blaise survey application. A full discussion of the system was presented at the 2009 Blaise Users Conference (Martens *et al*, 2009*)*.

Also at the 2009 conference, Alek Amin et al. (2009) and colleagues described the use and extension of the LMU system to translate and generate a single Blaise questionnaire in 10 languages (English, Arabic, Bengali, Cantonese, Gujarati, Punjabi (Gurmukhi script), Punjabi (Urdu script), Somali. Urdu, and Welsh) for the UK Household Longitudinal Study.

In a recent discussion, Maurice Martens (2010) described developments with LMU since the last conference. The multi-Blaise generator process has been further refined and used in other settings, including the SHARE Wave 4 multi-language instruments for Latvia, Luxemburg, and Switzerland. "We have some structures that set some boundaries on how you should break up your questionnaire, and use some special identifiers to define questions and know where answer types are etc. But if you keep to those rules then the multi-Blaise generator will work." Martens said.

8. Conclusion

This review highlights the broad range of significant collaborative research supported by Blaise. These six studies vary widely in terms of content, design, research approach, and institutional and organizational framework. But all successfully accomplish challenging and important research, and the Blaise software system is at the core of the survey data collection effort for each of them.

Key aspects of Blaise including its object-oriented block structure, the rules engine, support of cross-sectional and longitudinal designs, multiple languages, and integration into the overall survey process are central elements of the Blaise system that enabled the development, maintenance, and implementation of these surveys that have been so successful.

9. References

Aivadyan , Christina, 2010. Telephone Interview September 14, 2010.

Amin, Alerk , R. Boreham, M. Martens, & C. Miceli, 2009. "An End-to-End Solution for Using Unicode with Blaise to Support Any Language". 12th International Blaise Users Conference, Riga Latvia: International Blaise Users Group.

Anderson, Ellen & Lois Steinfeldt, 2004. " Blaise Instrument Design for Automated Food Coding". 9th International Blaise Users Conference, Gatineau Canada: International Blaise Users Group.

Börsch-Supan, Axel and Hendrik Jürges (eds), 2005, The Survey of Health, Ageing and Retirement in Europe – methodology. Mannheim: Mannheim Research Institute for the Economics of Aging (MEA).

Chardoul, Stepanie & Beth-Ellen Pennell, 2010. Telephone interview September 17, 2010.

Hoffnagle, Alison (2010). Telephone interview September 15, 2010.

Kessler, Ronald C. & T. Bedirhan Ü̈stün, 2004. "*The World Mental Health (WMH) Survey Initiative Version of the World Health Organization (WHO) Composite International Diagnostic Interview (CIDI)*". International Journal of Methods in Psychiatric Research, Volume 13, Number 2

Martens, M., Alerk Amin & Corrie Vis, 2009. "Managing Translations for Blaise Questionnaires ". 12th International Blaise Users Conference. Riga Latvia: International Blaise Users Group.

Marten 2010. Telephone Interview September September 24, 2010.

Moshfegh, Alanna J. 2010. Telephone Interview September 14, 2010.

NIAAA, 2010. "Collaborative Studies on Genetics of Alcoholism (COGA)" http://www.niaaa.nih.gov/ResearchInformation/ExtramuralResearch/SharedResources/projcoga.htm#Biomaterials [accessed September 27, 2010]

Pennell et al., 2008. “Implementation of the World Mental Health Surveys”. In Kessler, R.C. & T.B. Üstün (eds) , *The WHO World Mental Health Surveys: Global Perspectives on the Epidemiology of Mental Disorders.* New York: Cambridge University Press.

Prism, 2010. “PRISM Semi-structured Diagnostic Interview” http://www.columbia.edu/~dsh2/prism/ [accessed September 27 2010]

Rochberg, Nan, 2010. Telephone Interview September 15, 2010.

Schubert, Carol. 2010. Telephone Interview September 22, 2010.

Centralizing the Mink Survey at the National Agricultural Statistics Service

Roger Schou and Emily Caron, National Agricultural Statistics Service, USA

1. Introduction

The National Agricultural Statistics Service (NASS) is an agency of the United States Department of Agriculture. NASS is responsible for collecting, editing, and summarizing agriculture data. We are the sole agency for producing Agriculture Statistics for the United States.

NASS has had the unique challenge over the years of disseminating Blaise instruments across forty-four field offices from our headquarters (HQ) location in Washington, DC. With the introduction of Blaise 4.81, we could begin the process of centralizing our instruments in order to create a seamless approach to CATI and paper data collection and editing across the U.S. The first survey to undergo this transformation was the Annual Mink Survey. But before we delve into the details of centralizing this survey, let's provide a little background about how we have handled Blaise surveys in the past.

2. The Old Days

Our older LAN-based Blaise instruments are invoked with a Visual Basic 6 menu system comprised of numerous builds. Whenever a new instrument is created or an update is needed for an instrument or the menus, a LAN Update is sent out to each affected field office (FO). LAN Updates generally consist of zipped up files and a .bat file that, when run, will save the new files to the FO LANs. Someone with administrative rights needs to be on hand to run these programs. Between the preparation, dissemination, and the running of the LAN Updates at the FO level, much time and effort is spent on this process. It can sometimes be the case that LAN Updates are run out of order which, depending on the nature of the update, can cause problems.

Current LAN-based Blaise instruments at NASS use Blaise 4.80 and write to Blaise datasets. Since these datasets are located on each FO LAN, HQ survey administrators are not easily able to view the progress of a given survey. In order for collected data to be analyzed or summarized at the national level, it needs to be read out of the Blaise dataset and sent by some means to a common location.

Over the years, NASS has expanded six of its FOs to act as Data Collection Centers (DCC). An FO might require assistance with phoning due to having a particularly large sample or a small or even non-existent phone interviewer staff, in which case they will put in a request to become a Client State (CS). A DCC will be assigned to them and will take over the responsibility of phoning the CS's sample. Working in a decentralized environment means moving zipped up datasets between CSs and the DCCs and depending on communication links being up in order to do so. Some of the surveys we conduct also make use of Estimation Centers (EC) which requires more data movement.

Considering the need for real time national status reports, seamless data transfer and the agency's overall direction toward enterprise architecture, it was time to make a change.

3. Centralized Mink Basics

In April 2010, NASS conducted its first survey in a centralized database. We chose the Annual Mink Survey because of its small sample size (only 415), relatively long data collection time frame and simplistic nature. It was also a good candidate because the data is collected and edited in only two states, Wisconsin and Utah.

We started out using Blaise 4.82 build 1529 and stored the data directly into a MySQL database. In order to maintain a common structure across all future centralized Blaise instruments, we used Generic BOI files. The in-depth data partition type was selected and versioning was turned on.

4. Updated CASIC Menu and System Requirements

We created a Web-enabled menu in Visual Basic .NET which is used to invoke Blaise and Manipula and manage the survey. Whereas our old VB6 menu was made up of numerous builds all working together, the new menu is one build and can be updated on the fly without needing LAN Updates.

Certain settings are needed on user workstations in order to run the new CASIC menu and centralized Blaise. Both the .NET Framework and MySQL driver must be installed. Eight Blaise DLLs must be registered and an oledb.dbi file saved on the workstation. In addition, ODBC System data sources need to be set up to provide connection information to MySQL. All of these system requirements were pushed out to workstations in the FOs involved in the Mink Survey prior to the survey start date.

5. User Access

One of the first things the VB.NET code checks when a user attempts to access the new CASIC Menu is whether or not the user has been granted access to it. We have accomplished this by assigning certain fields in the Windows AD table for valid users. We are able to tell where they are located (FO Fips or HQ), what their username and employee number are, and what role they have in the office (Statistician, Stat Assistant, Supervisory Interviewer, Interviewer). As long as the corresponding AD fields have valid entries, the user is granted access to the menu.

6. CASIC Tables

In addition to the eight standard Blaise tables that accompany Generic BOI files, we found the need to create a few others: CASIC_SurveyInfo, CASIC_FAT and CASIC_Management.

The CASIC_SurveyInfo table is one of the first tables populated for a given survey. This table holds critical information about each survey, such as survey code; year, month and day values; Blaise instrument name; different survey type indicators; data collection dates and other useful information. This table allows us to limit which surveys are visible to the user on our survey selection screen. Many values from this table are passed as parameters into our menu and Blaise instruments so we can have each of them react as needed.

The “FAT” in CASIC_FAT stands for FIPS Assignment Table. An interface was created within the new CASIC Menu to allow HQ staff to interactively assign CSs to DCCs and ECs for each survey. Once these assignments are created, the resulting data is populated into the CASIC_FAT table and the menu will know which rights to provide to each FO. So if state A is having their data collected by state B (one of the DCCs) and state C (an EC) is editing all of the data, the menu will react accordingly. Users in state B will have access to menu buttons related to the call scheduler and data collection. Users in state C will

have menu buttons connected to edit functionality. State A users (CS) will have a very small collection of buttons in order to prepare their sample for initialize, read documentation and run certain reports. There was a total of 21 FOs involved in the Mink Survey. Of that number, two played the role of both DCC and EC which left 19 in the role of CS.

Originally we did not have the CASIC_Management table, but quickly realized it was a necessity. This is a flat table containing many key Blaise fields on which we often need to sort or limit the datasets. These fields are defined as indexes and used in RecordFilters in Manipula. Some examples of fields found in this table are DCC_Fips, EC_Fips and Batch.

7. Directory Structure

The new structure for centralized surveys involves web, application and data servers. There are three environments to each: Development, Beta and Production. CASIC Menu code is located on the web servers, Blaise code resides on the application servers, and the MySQL databases are on the data servers.

We went back and forth on how to set up the directory structure for Blaise code on the application servers. By the time the survey went live, we had determined we needed specific folders designated for any FO playing the role of DCC or EC. In the case of Mink, that meant special directories for Utah and Wisconsin which would house state-specific files such as CATI Specs, .IGL and those related to the interviewer practice dataset (which we kept as a Blaise dataset). Since we found the instrument .BMI file needs to be collocated with the .BTR, we also have copies of the instrument files in the FO folders.

A directory designated for HQ was created, as well. The purpose of this area is to gather files that FOs have prepared for initialize (explained later), back up those files once they are processed, and to gather any special reports or message files that are generated after HQ processes run.

The one external used for the Mink Survey contains the sample master and we decided to keep it as a Blaise dataset. The external datamodel was required to be in the same folder as the instrument.

8. Division of Tasks

In older decentralized Blaise instruments, each FO involved in the survey would receive the instrument from HQ and do all of the steps needed for sample initialize. In the new centralized Blaise, getting the Blaise instrument ready to go requires actions from both the FO and HQ. Each FO that makes up the survey sample will need to run what we call Initialize Preparation. This is a process that ensures all pertinent files are in place and runs numerous checks against the FO's sample to be sure it's ready for initialize. If there are any problems found, a report will be presented to the user so they can take action. If no major problems are found, that FO's sample will be queued up for initialize. The Mink sample consisted of records from 21 different FOs, so each one of those states ran Initialize Preparation over the course of a week.

As FOs were lining up, we ran a manual process in HQ to initialize the samples into the dataset. Utah and Wisconsin were able to begin collecting and editing data as soon as at least one of their Client States' samples was initialized. Having a centralized dataset gave those two main players the ability to run important reports for their regions that were required to complete the survey, which was a big improvement over the past method of manually sending the reports between FOs.

Special reports were created for HQ including a couple of different status reports, missing reports and an initialize status report. Data readout for Mink was done at the HQ level on a daily basis. Records

considered to be complete and clean from across the entire sample were read out to an ASCII file, which was then picked up by the analysis and summary tool.

9. Issues We Encountered and Lessons Learned

Throughout data collection and editing, users reported CATI Service errors on a very regular basis. While researching the problem, we were able to get them back up and running by simply downing the service and bringing it back up but that, of course, was not a permanent solution. After speaking with database experts in our agency and getting the advice of Statistics Netherlands staff, we changed some of the ODBC settings on the different application servers. Coupling that with an updated Blaise software version drastically reduced the number of CATI Service errors for the remainder of the survey period.

The performance (speed) of the Mink instrument as reported by FO users was not ideal. Originally, the Blaise software was being accessed by the menu from the central application server. Partway through the survey we decided to instead hit the software from the local FO LAN, and this helped to improve performance. This was around the same time that we received an updated version of the software, which also helped.

10. Future Plans

After wrapping up the Mink Survey, we knew we had lots more work to do before tackling a much more complex survey. We also wanted to be sure a proper database failover was in place which was not the case during Mink data collection.

Knowing that speed and performance would only decrease as samples and instrument complexity increased caused us some worry. However, our agency has plans to convert all FOs to a virtual environment by the summer of 2011. Preliminary tests of a virtualized FO using centralized Blaise have looked promising, so we have high hopes that performance issues will be a thing of the past. Virtualization should also eliminate the need for specialized workstation installs, as all setups, DLL registration, and the like will be done on the virtual image.

Discussions are underway to develop an Extract, Transfer and Load (ETL) process to pull data from the MySQL database (a transactional database) and populate it to the Work In Progress (WIP) database (an analytical database). The WIP database will house data from all of our different data sources (CATI, paper, our in-house WEB self-administered data collection (EDR), and our in-house small survey data collection instruments (EDC)) and will be the one source of data for analysis and summary programs.

It is possible we will decide to turn versioning off for future surveys. There were far too many rows being written to the dataset each time a record was touched, probably because we update a user stamp in the instrument whenever anyone accesses a record. We might look into using the ETL process to create an historic database off to the side including all iterations of the data, and thus keeping only the current instance of the record in the main database. However we decide to handle it, we will want the ability to easily access both the original and current instances of the data.

We will need to make use of UDL files, due to a security feature being put in place on the MySQL databases in the not-so-distant future. A generic username and password to access the databases will be stored in a separate dataset and the password will be changed every 30 days. We're considering using one shared, temporary UDL file for each survey which will use the current password.

When more surveys are lined up for us to convert to centralized Blaise, we will start actively researching CRON jobs to automatically run certain processes at night. These could include daybatch creation, initialize, data readout (until the ETL is in place), and possibly some others.

A deployment timeline is currently in place which has the next converted survey going live in late 2010. Our other regular surveys are slated for conversion through 2012, ordered according to increasing complexity where possible.

Management of Classification Lookup Files

Fred Wensing, Softscape Solutions, Australia

1. Introduction

A key aspect of data collection is the correct assignment of codes to represent text entries which are given in the interview. Where the coding frame is structured or the list is long, classification is often done through a lookup action on an external file.

When a survey spans a time period it is important that the systems are set up to handle the inevitable updates to the classifications which result from new entries and other revisions.

This paper will discuss the management of classification lookup files, their creation, the application of version control and the way that systems can be set up to manage the transition from one release to the next.

2. Classification basics

This section describes the classification options available in the Blaise software. It is included here for the sake of readers who may be unfamiliar with the options.

In the collection of statistical information it is usual to codify the data. This puts order and structure into the presentation of results. In particular, it avoids the incidental alphabetical re-arrangement of data which can occur to output presentation if data is stored as character strings.

Blaise supports various methods for the assignment of codes to the answers given in a survey:

- Enumerated field type
- Classify method (using a classification field type)
- Lookup from an external file using alphabetical search
- Lookup from an external file using trigram search
- Combinations of the above

Some details in this section of the paper draw on material in the *Blaise Online Assistant* and the Sample code provided with the Blaise software. It is included here for completeness. Please consult the *Blaise Online Assistant* for more comprehensive information.

2.1 Enumerated field type

The simplest and most common method of classification is the enumerated field type in which the interviewer selects the relevant code, which relates directly to the answer given, from a category list which is visible on the screen.

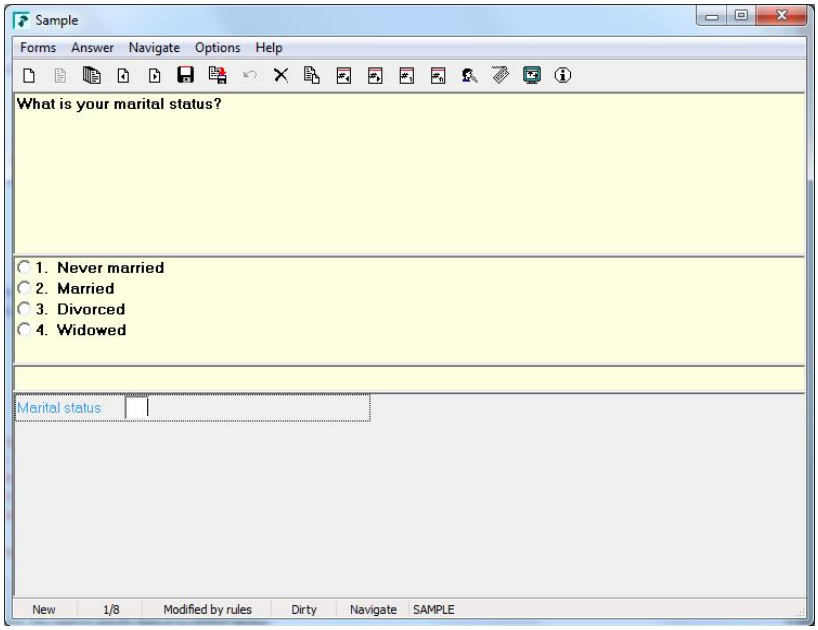
The field definition itself holds the classification. For example, in the following field definition:

```
MarStat "What is your marital status?"
      / "Marital status"
     : (NevMarr          "Never married"
       ,Married          "Married"
       ,Divorced  "Divorced"
       ,Widowed          "Widowed"
        )
```

The codes 1 to 4 are assigned for the four categories by default. A more explicit definition of the field reveals the classification:

```
MarStat "What is your marital status?"
      / "Marital status"
      : (NevMarr    (1) "Never married"
        ,Married    (2) "Married"
        ,Divorced   (3) "Divorced"
        ,Widowed     (4) "Widowed"
        )
```

When this question is encountered in the survey the list of categories shows on screen with relevant codes:



As you can see, by specifying the code number in the definition it is possible to assign a predetermined number to each category. The numbers do not need to be sequential and may contain gaps as can be seen in the following type definition for the response set of "Yes" (code 1) and "No" (code 5) which is often used in CAPI surveys:

```
TYPE
    TyesNo =
      (Yes  (1) "Yes"
      ,No   (5) "No"
      )
```

2.2 Classify method

Where the classification of an item involves a hierarchical code frame (For example: Occupation, Motor vehicles or Commodities) then Blaise provides a CLASSIFY method which will bring up the hierarchical code list for the operator to navigate through and select a corresponding entry.

To use this method it is firstly necessary to define the classification hierarchy to Blaise. This is done using a Classification type in which all the levels, codes and labels are defined. Essentially the Classification type is a series of nested enumerations.

An example of a hierarchical classification of motor vehicles is given in sample code provided with Blaise and part of the classification (also shown in the *Blaise Online Assistant*) is reproduced here for your information:

```
TYPE
 Automobiles = CLASSIFICATION
 LEVELS
   Make, Model, Body_Style
 HIERARCHY
   American_Motors (1) = (
     Rambler_or_American (1) "AMER Rambler/American" = (
       _2dr_Sedan_or_HT_or_Coupe (2) "2dr Sedan/HT/Coupe",
       _4dr_Sedan_or_H (4) "4dr Sedan/HT",
       Station_Wagon (6) "Station Wagon",
       unknown_or_other_style (88) "unknown" ) ,
...
```

An American Motors Rambler that is a two-door sedan, hard top, or coupe would have a code of 1.1.2.
If the list is to be updated during the life of the survey then the DYNAMIC attribute needs to be specified along with the length of the corresponding code structure. For example:

```
TYPE
 Automobiles = CLASSIFICATION DYNAMIC[9]
 LEVELS
   Make, Model, Body_Style
 HIERARCHY
   ...
```

The Classification type can be defined within the Blaise datamodel for that survey although it could also be set up as a separate type library so that it can be used by multiple surveys. A separate type library would also be preferable if the content is likely to be updated during the life of the survey.

Since the syntax for coding frames with complicated descriptions can be awkward to type by hand, particularly for large hierarchical coding frames, it is recommended that you build the classification using a Manipula program which converts a text description of the classification (which may have been exported from a classification register). For details on how that is done and information about sample programs on the subject of hierarchical classifications consult the *Blaise Online Assistant* Contents under:

How to Use... / Programming Languages / Advanced Topics / External Files / Hierarchical coding

Once the classification type has been defined it can be used like any other type definition. For example:

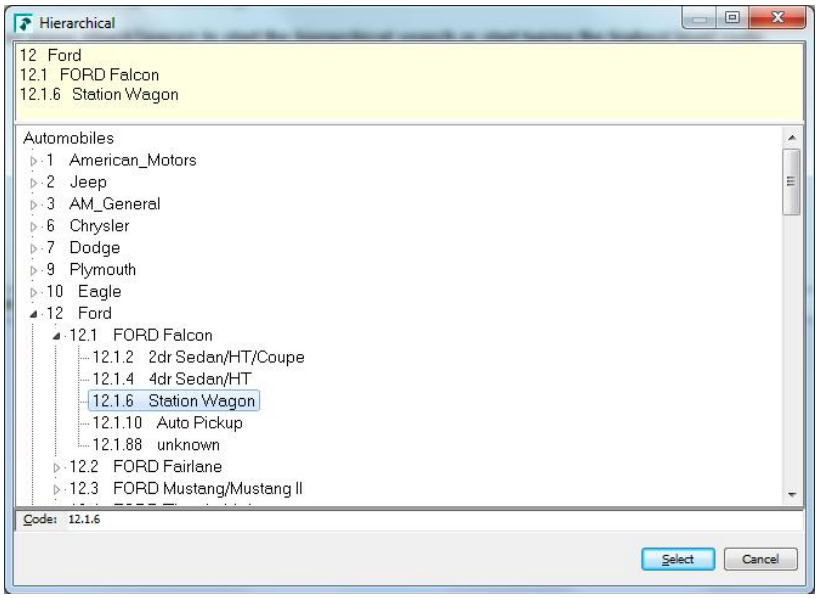
```
MyCar "Identify your car from the list" : Automobiles
```

To invoke the classification the field is placed on the path using the CLASSIFY method, rather than using the ASK method. For example:

```
MyCar.CLASSIFY
```

When the CLASSIFY method is invoked then the operator is presented with the hierarchy and then he/she navigates down the classification until the corresponding entry is located.

The following screen shows an example where the operator has located “Ford Falcon - Station Wagon” on the list:



Navigation through the hierarchy will return the code number for 12.1.6 for this entry.

2.3 Lookup from an external file using alphabetical search

When the list of categories is large it is not practical to set up an enumerated field. In this case it may be preferable to lookup an external file using one of the keys on that file. Use of an ordinary secondary key will result in an alphabetical list being presented to the operator.

For the external file to be accessible from the survey datamodel it is necessary to add a USES section to identify the external datamodel. For example:

```
USES
      mCountryList 'CountryList'
```

It is also necessary to add an EXTERNALS section within the relevant block to identify the external file:

```
EXTERNALS
      ExternalList : mCountryList ('CountryList',BLAISE)
```

The lookup is then activated in the RULES by associating the field in the lookup file with the field in the main datamodel. This is done with the piping symbol '|'. For example:

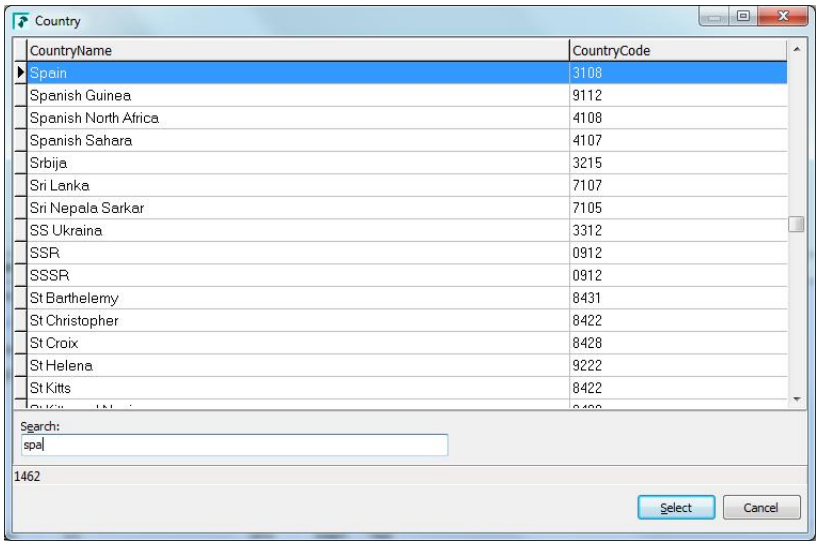
```
Country_of_birth | ExternalList.LOOKUP.Countryname
```

The lookup process will present the list of entries from the external file using the first or only secondary key. If the external file has more than one secondary key then search dialog will provide a radio button to enable the operator to switch between keys. If only one key is to be used in the lookup then the designated key can be added in the lookup instruction. For example:

```
Country_of_birth | ExternalList.LOOKUP(AlphaKey).Countryname
```

When the field is encountered in the survey then the lookup dialog will be presented on the screen. As the operator types the first few letters of the search string the lookup dialog will jump to that point in the list. The value which is returned by the lookup process is the content of the field named after the LOOKUP keyword, in this case **Countryname**.

The following screen shows the alphabetical lookup on a list of birth places after the first few letters of "Spain" have been typed in:



Once an entry has been selected then the identified field is returned to the survey data.

2.4 Lookup from an external file using trigram search

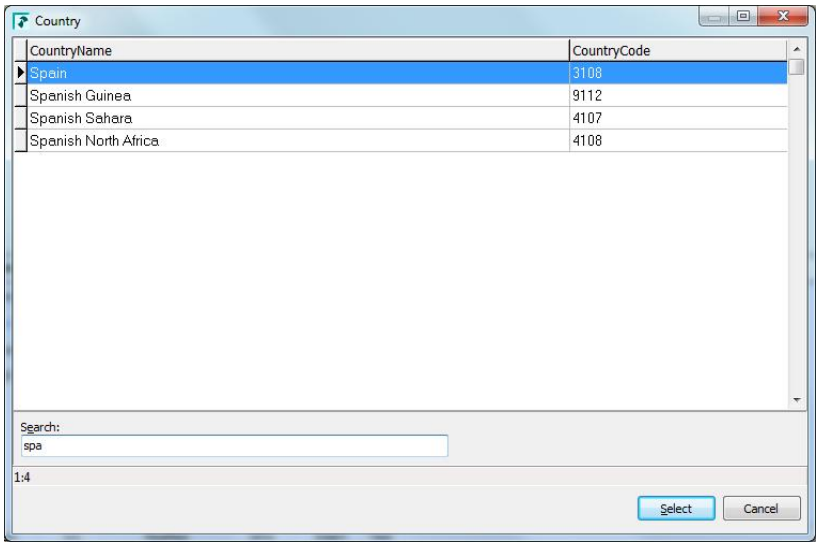
Blaise provides a more comprehensive search mechanism on text strings through the application of trigram indexing. Trigram indexing causes a text string to be indexed on all the subsets of three letters extracted from the text string. For example, the word Blaise consists of the trigrams #BL, BLA, LAI, AIS, ISE, and SE#, where # represents word boundaries.

By using the trigram index in the lookup process Blaise will return to the screen all entries that match a defined number of trigrams found in the search text. The more text which is entered into the search field the more trigrams will be searched and the shorter the list of possible matching entries will become.

Because Trigram searching occurs on patterns of three letters it does not matter where those three letters appear in the indexed field. Trigram searching is therefore ideal for long descriptive texts.

The activation of lookup using trigram search is the same as for alphabetical search except that the designated key to be searched must be of type TRIGRAM.

For more details about Trigram indexing and setting or changing the parameters which control the search behaviour please consult the *Blaise Online Assistant.*
The following screen shows the trigram lookup on a list of birth places after the first thee letters of "Spain" have been typed in:



Notice that the list presented to the user is more refined than the alphabetical list.

2.5 Using a start value for lookup operations

The lookup process also allows for a start value to be passed into the lookup. That value can help the operator to narrow the search process. Where the start value is part of a Classification code then the subsequent lookups, whether alphabetical or trigram, are also filtered by that code.

2.6 Retrieving other fields from the external file

The value which is returned by the lookup process in the above two subsections is the content of the field named after the LOOKUP keyword, in this case **Countryname**.

In order to retrieve the values of other fields from the external file it is necessary to search and read the external file after the lookup has been done. For example, to obtain the country code for a selected country the following code can be included. For example:

```
IF ExternalList.SEARCH(Country_of_birth) THEN
    ExternalList.READ
    Country_Code:=ExternalList.CountryCode
ENDIF
```

2.7 Combining lookup methods

By combining the various lookup methods it is possible to provide considerable flexibility for any coding system.

3. Preparing Classification Lookup files

Most classification lookup files are created from classification data that is extracted from classification registers or systems. Such data is usually placed in a text file, which may be fixed-format or character-separated, which can then be loaded into Blaise using a Manipula program.

Before loading the classification data into a Blaise file that can be used in lookup and/or search processes you need to consider and decide on the following.

3.1 Use of a hierarchical classification

Classification frames are often developed as a hierarchy because it provides structure and lends itself to convenient addition of new codes.

If a hierarchical classification is used then there is an option to make use of the hierarchical classification type in Blaise. The definition of a classification type in Blaise is somewhat complicated but an easy-to-use conversion program is provided in the Blaise Sample files. Consult the *Blaise Online Assistant* for more information.

Use of a hierarchical classification type will give access to the Classify method and allow the operator to navigate the hierarchy branches to make a selection.

Make sure that you add the Dynamic option if you expect the classification to be extended over the life of the survey.

If the Classify method is not to be used then the hierarchical code can be converted to a simple code string and assigned through the other lookup methods.

3.2 Which entries to put into the lookup list

To make the lookup list as useful as possible, the entries should be drawn from a list of synonyms linked to the relevant classification rather than from the classification titles themselves. For example, if “Great Britain” is the title for an entry in a list of countries it would be better if the lookup list contained not just that entry but also some synonyms like “United Kingdom”, “UK” or even “England”.

The more entries which are in your lookup list the better it will be for the coding process.

If you do not have a comprehensive list of synonyms then you can plan to update your list from time to time based on “Other – specify” entries that are extracted from your survey and sent to an expert for coding. Once a decision has been made about an appropriate code the newly coded description(s) can be

added to the lookup list. Updating of code lists and coding unassigned entries is discussed in more detail later.

3.3 The types of indexes to provide

The decision to include a Trigram index in your code list will depend on how structured the coding is expected to be. Trigram indexes are best suited to coding where there is an unambiguous match between the text and its possible code (for example in a list of commodities or localities). Trigram searching generally ignores the structure of the coding frame.

For some coding frames which are highly structured (for example Occupations) then trigram searching may not be appropriate. You will need to decide this in consultation with classification experts.

An alphabetical index should generally be provided in the code list (as a Secondary Key) so that it can be used if desired.

For most applications, then, two Secondary Keys should be provided, one as a Trigram index and the other as an Alphabetical index.

3.4 Adding and retrieving other fields from the code list

The code list can be enhanced to provide other information that may be useful for the survey. These additional items could be things like edit tolerance levels for application in Signals or Checks, or weights for calculation of exercise indexes. It is particularly useful to provide this kind of data through the code list if it is expected that refinements or additions will be made during the life of the survey.

When other information is added to the code list then it will be necessary to add a Primary Key to the code list so that it can be searched. Once a matching external record has been located then it can be read and the values returned to the survey data.

3.5 Whether to add any special entries to the list during loading

For situations where a suitable entry may not be found in the code list, it may be appropriate to add a special entry, such as “Other – specify” (which would have a non-specific code number attached to it). The datamodel could be programmed to obtain a verbatim description for such cases. These verbatim entries could then be extracted from time to time for coding decisions to be made.

Another special entry which may be useful could be a “Delete me” entry. When this is chosen the datamodel could be programmed to remove the selection and its resultant code from the data. This may be the chosen solution to removing unwanted code selections because the delete key does not function on a field which is controlled by a Lookup. For example, the delete logic could look like:

```
{Ask third activity using trigram coder}
Activity_3|CodeFile.LOOKUP(TriKey).Description
{Remove activity if it starts with 'DELETE'}
IF POSITION('DELETE',Activity_3)> 0 THEN
    Activity_3:=''
    Code_3:=''
ENDIF
```

There are other solutions to the “delete” problem, such as adding a menu button to clear the field, but this method might be an easy one to implement.

3.6 Make adjustments for punctuation which may affect Trigram indexes

When the text strings which are used to create the lookup files contain punctuation such as commas, full stops, brackets etc. then they will affect the trigrams into which those strings will be indexed. Given that the operators will only be entering a few letters (and probably not any of the punctuation) then it is best to adjust the search texts so that the punctuation is separated from the words around it by a blank character or space. For example: a search string of "Food prep (cook, grill, bake)" would be indexed better if it was converted to "Food prep (cook , grill , bake)". Alternatively, the punctuation could be removed completely.

The following procedure in Manipula applies such an adjustment to the texts before indexing:

```
PROCEDURE proc_FixString
  PARAMETERS
    IMPORT OldText : STRING
    EXPORT NewText : STRING
  INSTRUCTIONS
    NewText := OldText
    NewText := REPLACE(NewText,',',' ,')
    NewText := REPLACE(NewText,'(','( ')
    NewText := REPLACE(NewText,')',' )')
    NewText := REPLACE(NewText,':',' :')
    NewText := REPLACE(NewText,'/',' / ')
ENDPROCEDURE
```

Even though this kind of adjustment is made for the indexes, the original text is still retained and returned to the survey data file when the lookup is used. The following code illustrates this:

```
Activity|CodeFile1.LOOKUP(TriKey).Description
```

Where the Secondary Trigram Key (TriKey) is based on a field called **SearchString** which is a copy of the **Description** field that has been "fixed" using the above procedure.

3.7 Make adjustments for language/keyboard differences which may affect Trigram indexes

When preparing a multilingual instrument, it may be necessary to make adjustments to handle differences in lettering due to accents. In particular, those adjustments should be made so that it is possible for the operator to use an English/American keyboard to enter search texts without the accents.

For example, Spanish words commonly contain special letters such as 'ñ' and the Spanish language also employ accents which are placed over vowels. It is not easy to access these special letters on an English/American keyboard. In order to make trigram lookups in Spanish more readily accessible to the coder, these special characters are converted to (English) versions that do not have the accents.

For adjustments to Spanish texts, the above procedure is extended with the following extra lines:

```
    ...
    NewText := REPLACE(NewText,'Ñ','N')
    NewText := REPLACE(NewText,'ñ','n')
    NewText := REPLACE(NewText,'á','a')
    NewText := REPLACE(NewText,'é','e')
    NewText := REPLACE(NewText,'í','i')
    NewText := REPLACE(NewText,'ó','o')
    NewText := REPLACE(NewText,'ú','u')
ENDPROCEDURE
```

Once again, the Trigram key will make use of the modified text whereas the Lookup will return the original Description text.

3.8 Adding the code numbers to a Trigram index to increase functionality

Sometimes coders will become familiar with the code numbers used in the coding frame and so it may be useful to enable them to search on those as well. This can be enabled by appending the code number to the search text used in the Secondary Trigram Key. For example:

```
SearchString := Description+' '+Activity_Code
```

Here also, the Trigram key will make use of the modified text whereas the Lookup will return the original Description text.

Using this technique can also provide a convenient search mechanism to locate all the entries coded to the same number.

4. Deployment of code list (lookup) files

Code list lookup files must be accessible to the systems in which they are to be used. When it comes to deployment, the main issues to consider are whether the lookup files are likely to be used by multiple surveys, how often they are likely to be updated and, in the case of network configuration, performance.

4.1 Distributed computer deployment

For distributed systems where each interviewer has his/her own notebook computer there are two sensible options for placement of the lookup files:

- in the same folder with the survey files
- in a common folder placed next to the survey folder

The first location is appropriate when the code list is unique to a survey and/or there are likely to be some updates during the life of the survey. Management of the installation tends to be easier because it just entails replacing the files in the survey folder with an updated set.

However, where code lists can be applied to multiple surveys then it is appropriate that you place all the code list files in a single folder which is parallel to the survey folders. For example, the following folder structure could be used:

```
C:\DATA\EXTERNAL
C:\DATA\Survey1
C:\DATA\Survey2
...
```

In this case, all the shared external lookup files would be placed in the EXTERNAL folder. This also has the advantage that there would be only one copy of each code list installed on the computer rather than one copy per survey.

If the code list files are always placed in the EXTERNAL folder (and parallel to the survey folder) then the file references to these external files can be easily managed using relative referencing. For example, the following Uses and Externals statements would locate the relevant country list in the EXTERNAL folder:

```
USES CountryList '..\EXTERNAL\CountryList_1_021'

EXTERNALS ExternalList : CountryList
   ('..\EXTERNAL\CountryList_1_021',BLAISE)
```

Using these references (involving **`..\EXTERNAL\`**) means that the survey datamodel will always find the external file(s) provided they are in a parallel folder called EXTERNAL.

4.2 Network system deployment

With network systems, the placement of code list lookup files should be done to provide the best performance response. Despite the improvements with network performance in recent years the best location for lookup files is still the local computer of the operator. This applies even if the survey data is to be held on a network location but it applies even more so if the lookup files (when placed on the network) would be being accessed by multiple operators at the same time. Local placement will give exclusive and speedy access.

Local deployment for network use, involves using a designated folder on the C drive where the lookup files can be placed. When the network system is invoked the local pathname can be supplied as a parameter to the Manipula program or Data Entry Program (DEP). This can be done using the relevant command line option or by using a Blaise Command Line Options file.

For example, the following Manipula statement invokes the DEP and sets the external search path to the C drive location (**`C:\DATA\AllCoders`**) using the command line option:

```
aResult:=fInterview.EDIT ('/K'+aIndic
        +' /E"C:\DATA\ALLCoders"'
        +' /X ')
```

or using a command line options file:

```
aResult:=fInterview.EDIT ('/K'+aIndic
        +' @SVY_System.ini'
        +' /X ') {make sure the /X option is after the Ini file}
```

where the options file (**SVY_System.ini**) could look like:

```
[DepCmd]
ExternalSearchPath=C:\DATA\ALLCoders
MenuFile=CAPIMenu.BMF
ConfigFile=AuditTrail.DIW
[ManipulaCmd]
ExternalSearchPath=C:\DATA\ALLCoders
```

Notice that the external search path is specified for both DEP and Manipula so that either program can find the external files.

Where an external search path is to be supplied at time of execution, the references to those lookup file within the survey datamodel are done without the folder names being mentioned. For example the Uses and Externals statements previously shown would now look like:

```
USES CountryList 'CountryList_1_021'

EXTERNALS ExternalList : CountryList('CountryList_1_021',BLAISE)
```

When installing the survey it will be necessary to place the datamodel definition files (*.bmi, *.bdm, *.bxi) for the lookup files in with the survey. The other alternative would be to set the Meta Search Path in the same way as the External Search Path by using a command line option **(/H)** or adding it to the options file.

5. Version control

It is almost inevitable that coding lists will be updated. It is therefore important to consider a system of version control so that you know which version of a coding list was used with a particular cycle of a survey. Two methods are suggested here.

5.1 Version number as part of the code list file name

A version number can be incorporated into the name of the code list. For example, for release 1.01 of the Activity coding list the suggested datamodel and file name could be Activity_1_01. Whenever a new release is made the version number is updated and therefore the file names are also updated.

By having the version number embedded in the name it is clearly visible to installers of the system which version is being used. Embedding the version number in the name also means that different versions of the same coder can be available at the same time (for different surveys or time periods).

This solution is appropriate for code lists which are shared between surveys and which do not change often. In this situation a typical survey would make use of a particular release of a code list, probably for the life of that survey. Of course the name of that code list file would be referenced in any Uses statement, and any Externals section of a datamodel so it would always be known which version was being used. For example, see the Uses and Externals for the country list above.

Using this method means that any application of the code list in Manipula or a Blaise datamodel will need to refer to the correct version by its name. If the current file is not found then the error message will name the file (with version number embedded) and it will be easy for the maintainers of the system to know which file is not in place.

Using this method does mean, however, if a correction is made to a code list and a new release is required then the file name(s) would be changed to match the new release number. It would be up to the managers of any affected survey to decide whether to adopt the new version and change and recompile any datamodels or programs. Provided that the structure of the datamodel used for the code list file did not change then the modified and recompiled survey datamodel will be compatible with the previous version.

Updating a code list while a survey is in progress does raise the issue of how to refresh the lookups and searches which were done in the earlier period of the survey. This is discussed further later.

5.2 Version number as part of the folder name

If a code list lookup file is expected to be changed a few times during the life of the survey then the version number can be embedded in the name of the folder where the external files are located. That way the name of the code files can remain constant and the system programs will not need to be modified or recompiled.

By having different folders for each release it will be necessary only to update the external search path used by the DEP or Manipula. In that case the use of a Blaise Command Line Options file would be advisable. Being a simple text file this would be the only part of the installation which would need to be updated or replaced whenever a new code list was released. The following is an example of a Blaise Command Line Options file used to operate the Physical Activity Recall (PAR) survey:

```
[General]
Version=1_053
Installer=Fred Wensing
SysFolder=H:\Blaisedata
CoderVersion=3_013A
CoderLocation=C:\DATA\PARCoder
Managers=4,9,52
[DepCmd]
ExternalSearchPath=C:\DATA\PARCoder\Version_3_013A
MenuFile=PARMenu.BMF
ConfigFile=AuditTrail.DIW
[ManipulaCmd]
ExternalSearchPath=C:\DATA\PARCoder\Version_3_013A
```

As you can see the folder containing the coder lookup files is named Version_3_013A. When a new release is made then another folder name would be used.

This INI file (called **`PAR_System.ini`**) is used to define various system settings and is interrogated by the Maniplus program which controls the survey on the network. The same INI file is passed to the DEP as a command line options file. Whenever a new coder is released the system is programmed to update this INI file to reflect the change of version number and the changes to the folder name(s). This would then flow into all the uses of the system.

6. A management system for Classification lookup files

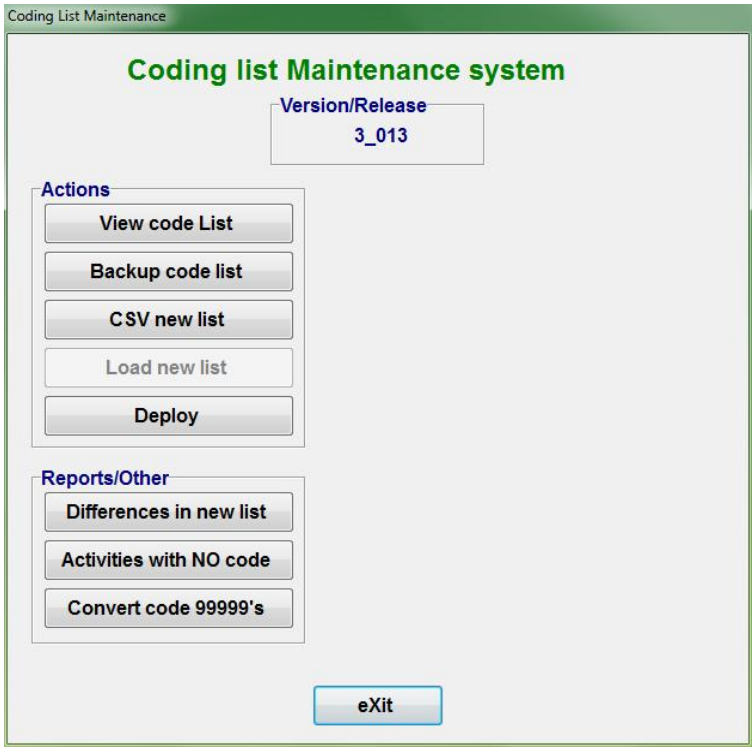
When code lists are likely to be updated during the life of a survey it would be appropriate to develop a management system to handle all the steps involved.

In a typical classification management system the following processes would need to be supported:

- knowing and updating the version control number
- backing up the current coder(s)
- extracting a list of entries from the survey data file(s) which could not be coded – that list would be sent to coding experts to make decisions and update their synonym list
- importing an updated list of synonyms into the classification management system
- obtaining a report on the code changes made in the updated list (in case there are unexpected changes)
- preparing the updated code list for deployment
- applying the updated coder to attempt to code those which were not coded previously

A system to handle these processes can be prepared using Manipula/Maniplus.

The following screen shows what such a system could look like:



Behind each of the buttons in this interface is a procedure or a Manipula program which carries out that step. The current version number is stored in an INI file which is interrogated by the system before being displayed on screen or used in other steps.

Maintenance tasks involve running the "Action" processes in order. The other processes can be run as needed.

6.1 Notes on the maintenance system implementation

The code list maintenance work is done in a folder which is separate from the current survey system even though the maintenance function is accessed from within the survey system.

The datamodel which defines the code list includes a field to record the code which a particular entry may have had in the previous release. The old code is obtained by checking whether an entry is in the previous release and noting what the code used to be. Storing both the current and previous codes makes it simple to produce a report of any changes that may have occurred. The differences report process also checks the backup code list for any entries that may have been removed before reporting the results.

In the main survey data file, entries that cannot be coded are "entered" as a literal string (using a button on the menu). Because those entries are not (yet) in the code list, the system is programmed to record the code number 99999 against those entries making it easy to identify them for listing and subsequent update when the lookup list has been improved.

The coding maintenance functions themselves do not interfere with the operations of the survey. Installation of updated code list files to their correct location (on the local computer of the operator) is triggered by the main system programs once the version number is updated.

6.2 Some technical details of the maintenance system implementation

Within the maintenance folder there is a copy of the current code list file and all previous releases of the code list. Each old release is named such that the suffix on the name matches the version release number.

The **View code list** button will open the current version of the code list using the Data Entry Program in the Browse Forms mode. That way progress with the update can be checked by the operator.

The **Backup code list** button runs a procedure that copies the current code list file to one which has the same name plus a suffix which matches the current version number (namely that shown on the screen). The system will only allow this button to be pressed once per release. Once the backup exists then the current version becomes the draft for the next release.

The **CSV New list** button opens a "Selectfile" dialog to locate a suitable CSV file to be loaded.

The **Load New list** button runs a procedure which loads the code list entries from the identified CSV file. During the loading process the text entries are "fixed" for punctuation and language issues as outlined in Section 3. In addition, as each entry is loaded, the backup copy of the code list is checked to see whether the entry existed previously and if it did then the old code number is copied into the "Previous code" field.

The **Deploy** button copies the current (now updated) code list files to a staging folder in the survey system where they can be accessed for installation later.

The **Differences in new list** button runs a Manipula program that reports on the changes in the current code list from the previously backed up copy. It identifies new entries, deleted entries and changes in the codes. The text report is written to a folder where it can be accessed later. That way all the changes between the different releases are stored for future reference.

The **Activities with No code** button runs a Manipula program which examines the records in the survey data file and produces a report of all the description which do not have a valid code (may be coded to 99999). That text report is also written to a folder where it can be accessed later.

The **Convert 99999's** button runs a Manipula program which examines the records in the survey data file and, for records that contain one or more entries coded to 99999, it applies the Checkrules method with the setting of CheckrulesUnchanged=YES. A report on records that have been "fixed" and the number of remaining non-coded entries is also produced and written to the same reports folder.

7. Handling some special classification issues

This section discusses some special issues that may arise with using external lookup files.

7.1 Refreshing searches following update of code list

When a code list has been updated it may be necessary to refresh the searches to ensure that any data based on the lookup entries is up-to-date.

If an affected record is to be reopened using the Data Entry Program then all the searches and retrieval of external data for that record will automatically be refreshed.

To ensure that all the searches are refreshed across the whole data file, however, it will be necessary to open every record and reapply the Rules. This can be done using a Manipula program but it will be necessary to apply the additional setting of CHECKRULESUNCHANGED=Yes. This is because the default checking process will only check blocks that have changed, and where external files are involved, such a change will not be known to the block. The following sample program from the *Blaise Online Assistant* shows how it can be done:

```
USES
    MyModel 'LFS98'
UPDATEFILE
    Lfs: MyModel ('lfsdata', BLAISE)
SETTINGS
    CHECKRULES = YES
    CHECKRULESUNCHANGED = YES
MANIPULATE
    Lfs.WRITE
```

7.2 Not losing data when the external code list file cannot be found

If the external lookup files are placed in a parallel folder, or on a folder on the local computer, then the potential exists for data to be lost because the search could not find the file. This is particularly possible when the survey files are moved around during processing.

One way to avoid this happening is to add a KEEP instruction to affected fields in the survey datamodel when the search is unsuccessful. For example:

```
IF ExternalList.SEARCH(Country_of_birth) THEN
    ExternalList.READ
    Country_Code:=ExternalList.CountryCode
ELSE
    Country_Code.KEEP
ENDIF
```

This technique is always valid when the same external file (**`ExternalList`** in this case) is used for both the lookup and the subsequent search/read. The assumption is that once a value has been obtained using the search (from the same unchanged file) then the result of the most recent successful search would still be acceptable.

This technique should not be used when the content of the lookup file has been changed during the life of the survey because the assumption may not hold true.

7.3 Use of ASCII file for small lookups and ease of updating the code list

Where code lists need to be updated often or tailored for particular uses then a simple solution may be to use an ASCII file to hold the code list rather than a Blaise file. In that case the Uses and Externals statements would look like:

```
USES Citylist 'Citylist'

EXTERNALS ExternalList : CityList('Cities.csv',ASCII(SEPARATOR=',')
```

where **Citylist** is a normal Blaise datamodel with primary and secondary keys defined. In this case the external file is comma-separated which is why the separator setting is added. For a fixed format ASCII file the separator would be left out.

Using this method, separate city lists could be provided to different interviewers, matching their local area. There would be no need to reload the data for a new or modified list. Blaise also loads the list into memory which is faster for the lookup. The only disadvantage is that the Trigram indexing is not available for lists that are loaded into memory.

7.4 Keeping track of alternate External lookup files in a multilingual survey

For multilingual surveys it is preferable that the lookups will be done using files corresponding to the active language. So for every lookup instruction there will be conditional logic to direct the operator to the relevant language lookup file. The problem is that if the active language is changed then all the searches in the datamodel, including those already performed, would be switched to the new language and many of them would fail. There is a serious risk of losing some of the data already collected.

To control the problems which can result from a change in the active language it is necessary to note the language for each lookup when it is performed and store it in a flag field. That flag can then be used to control which lookup file is searched and stop the possibility of using the incorrect lookup file (for each instance).

For example, the Physical Activity Recall Survey was developed as a bilingual survey with all questions and lookups being available in both English and Spanish. The operator can change the language at any time to use the one which is relevant for the respondent. The following logic is used to set the language flag for once instance of the Activity question (which uses trigram lookup on a list of activities):

```
{Set the language for the Activity coder}
Activity_Lang.KEEP
Activity.KEEP
IF (Activity=EMPTY) THEN
    IF Purpose=EMPTY THEN
        Activity_Lang:=''
    ELSEIF Activity_Lang=EMPTY THEN
       IF ACTIVELANGUAGE=ESP THEN
           Activity_Lang:='Sp'
       ELSE
           Activity_Lang:='En'
       ENDIF
    ENDIF
ENDIF
{show the language being used}
Activity_Lang.SHOW
```

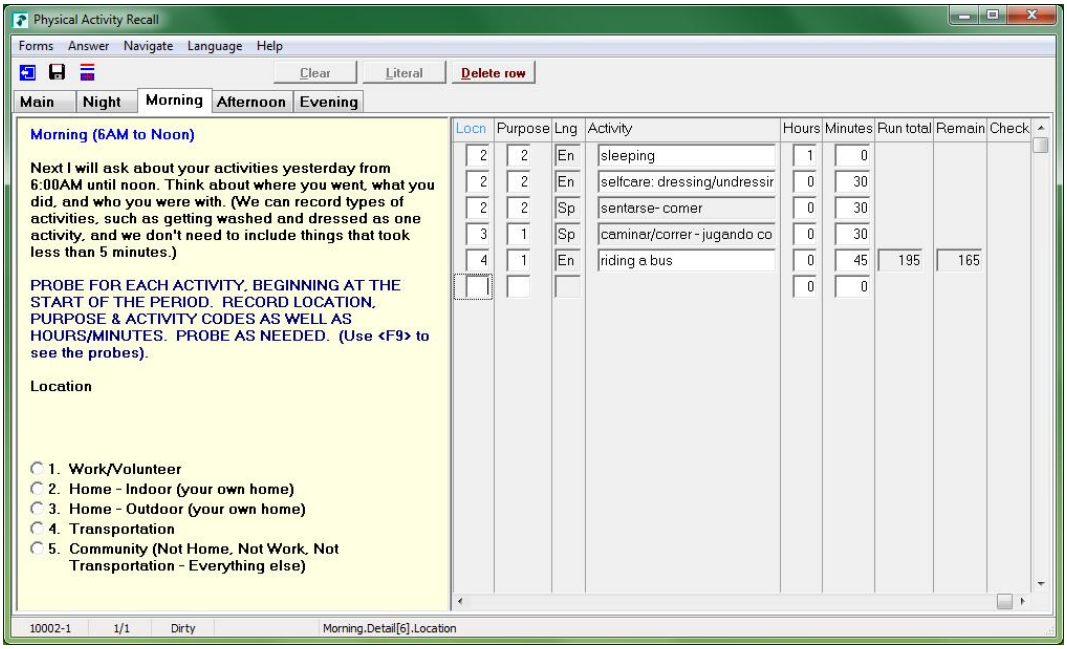
The language flag is checked and if it matches the active language then the lookup action is allowed, otherwise it is changed to just SHOW the results of the lookup when it was done in the other language:

```
IF Activity_Lang='En' THEN
    IF ACTIVELANGUAGE<>ESP THEN
        Activity|CodeFile1.LOOKUP(TriKey).Description
    ELSE
        Activity.SHOW
    ENDIF
ELSEIF Activity_Lang='Sp' THEN
    IF ACTIVELANGUAGE=ESP THEN
        Activity|CodeFile2.LOOKUP(TriSpanish).Descr_Spanish
    ELSE
        Activity.SHOW
    ENDIF
ENDIF
```

All follow-up searches of the external code list are also controlled by the language flag:

```
{Use the codefile to assign activity code number}
IF Activity_Lang='En' THEN
    IF CodeFile1.Search(Activity) THEN
        CodeFile1.READ
        Activity_Code := CodeFile1.Activity_Code
    ENDIF
ELSEIF Activity_Lang='Sp' THEN
    IF CodeFile2.Search(Activity) THEN
        CodeFile2.READ
        Activity_Code := CodeFile2.Activity_Code
    ENDIF
ENDIF
```

The following screen shows the effect of using this technique on the data capture process:



You can see the language used flag next to each activity text (obtained by lookup) and the (Spanish) entries which do not match the active language are displayed as SHOW text only. Those fields cannot be changed while the language is set to English. If the language is changed to Spanish then the English text entries would become SHOW text only and the Spanish text entries would be able to be changed.

8. Conclusion

This paper has covered a range of material associated with classifications and the use of external lookup files with the objective of explaining the processes involved and providing some techniques and solutions for the issues. Complimentary information and detailed technical descriptions may be found in the *Blaise Online Assistant*.

Coding Tricks To Save Resources

Barbara S. Bibb, Lillie Barber, Ansu Koshy, Research Triangle Institute (RTI), USA

Introduction

As computer aided survey administration has developed, survey instruments have become increasingly complex. Increased complexity in survey methodology seeks to gather the most correct data possible. For surveys to be appropriately administered under all possible circumstances, instruments are customized for individual respondents, reducing burden and complexity for the respondent but raising it for the programmer. As a result, the software applications used to present the instruments can be pushed to their limits. Innovative methods are therefore needed so that instruments can be built within the application's limitations and so they use resources conservatively.

This paper will discuss a Blaise instrument that would have grown out of control using straightforward instrument coding methods, but which was rescued by some tricks applied during development. The survey was about finances and insurance, difficult subjects at best for most people, and very difficult for respondents whose personal lives took many twists and turns. To elicit good answers, the questionnaire designers included loops, cross-references, recall aids and other techniques to help obtain the necessary data. This may have been helpful for the interviewer and subject, but caused difficulty during implementation.

For example, in the early stages of programming, one particularly complex module was only about a quarter complete and contained over 15,000 lines of code. Because of its length, simple "typo" coding mistakes would have been impossible to find, and there was concern that the program would break with each compile. This paper will present some of the ideas and constructs that were used to reduce the amount of code necessary to complete the instrument. Pros and cons of the techniques used will be discussed as well as some lessons learned.

Historical Background

In the middle of the 1900's census taking and simply counting was extended to create survey research by using questionnaires and statistical surveys to collect data about people and their attitudes, opinions, and beliefs. The effects of radio were investigated by collecting public opinion data to gauge reactions to and changes in the economic environment. At the University of Michigan, Rensis Likert and Angus Campbell (former employees of the U.S. Department of Agriculture's Division of Program Surveys), formed the Survey Research Center (SRC). Survey research expanded to cover behaviors and elements of interest in social sciences.

During this period, data was collected using complex paper forms containing directions that guided respondents around inappropriate questions. Later, computers made it possible for surveys to be customized for individual respondents, with obviously inappropriate questions automatically skipped. More sophisticated computer applications applied complex logic to create surveys that seemed to talk to the respondent directly, personalizing questions based on the respondent's answers and demographic characteristics. Thus, an effectively programmed instrument reduced the burden on the respondent.

As computer storage became more cost effective, more data could be collected while the effective programming of the instrument worked to reduce the burden on the respondent. In addition to the data collected from a respondent, computer applications have been asked to generate and maintain variables

used to appropriately guide the respondent through the maze of questions, administering only those appropriate for that respondent and his/her situation. The number of variables collected and stored within a survey instrument can rapidly increase when the instrument is designed to collect and maintain person level data within a household data collection effort. To be sure that no one in the household is left out the array sizes specified are large. Space must be allocated for each possible array element whether or not that space is needed in a particular administration. Application programs usually allow ample flexibility, but even so innovative projects push the limits.

Project Background

To test methodology in a recent project for the US Census Bureau, an experimental version of some standard questions was programmed. The respondent for the study was the person of appropriate age who answered the phone or was available to come to the phone. This person answered questions for the household and for each household member, so data being collected was both household level data and person level data. The person-level data was stored in arrays where the index of the array was mapped to each person's position on the household roster.

In addition to a fixed set of modules, the program contained three modules that each collected the same data in different ways. Two of these modules were question sets from existing studies. The third was an experimental module that was implemented for the first time. This module is the topic of this paper. Each respondent went through only one of these three modules, providing control groups and data for comparison. The project goal was to try to determine which set of questions would generate the most accurate (reliable and valid) data. The programming challenge was to code the experimental module.

Because human memory can be inaccurate, the experimental module was designed to collect information by repeating some questions a number of times in slightly different contexts. The respondent answered the sequence of questions in a number of ways, eventually covering all household members. Although sections of questions were repeated, the total respondent burden was reduced because each round of questions collected as much data as possible. In the later rounds, the only repeated questions were those needed to pick up data missing as a result of the previous question sequences.

In the worst case, the entire series of questions would be repeated for each household member. The program, however, reduced respondent burden by asking a question and then asking which household members could be included as having the same response. Flags within the program allowed the respondent to answer for all household members with different circumstances and prevented needless repetition of questions. To ensure accuracy of the data with respect to the respondent's memory of the past, questions to verify the data were added.

Programming began with a standard straightforward approach of asking the questions by grouping them into blocks that could be re-used when the question sequences needed to be repeated, setting the flags, and then applying logic to determine which sequences of questions needed to be asked next. To correctly drive the program, the backend variables needed to be set. Between the repetitious questions and the required backend variables, the section of the program that was doing this work rapidly became very large. There was concern that with each attempt to compile, the internal Blaise limits would be exceeded and the project would die. In addition, the path changed as the backend global variables were adjusted and updated with additional question responses.

To include every person in each household, this project had especially large person-level arrays. The project allowed for up to 16 household members. Therefore, each person-level array had an index of 16,

where the index number corresponded to the household member's position on the original roster enumerating the household.

To determine the status of each household member with respect to the data being collected, global "flag" variables were defined for each person. Data was being collected by month and there were 17 applicable months. In addition, there were 14 different classifications of the data or types of insurance coverage. Backend internal arrays were filled with variables needed to summarize the collected data and drive the program. The declarations of these internal person level arrays initially resulted in 3808 variables (16 household members * 14 types * 17 months). This variable count does not include any of the answers to the questions presented to the respondent that were needed to collect the data. The enormous number of variables presented problems. A program with such large numbers of variables could require more space than Blaise, the application program, allows. Blaise has internal limits that include both the size of the code files the program is able to compile and the number of variables that Blaise can maintain and can keep track of. Any application program would have such limitations.

To resolve the problems in handling this experimental module, the programming staff decided that it was essential to simplify the code if possible. This programming exercise had two challenges:

- To reduce the number of variables
- To reduce the number of lines of code

Before re-thinking the initial approach, one file contained over 15,000 lines of code, much of it repeated with only a change of index. That file only covered a quarter of the intended question sequences for the module. There was too much code to process, debug, and work with. Typos within the indices or any other part of a code file this size would never be found. After innovations were applied, the final module was reduced to two files of about 4000 lines.

Block development

Several steps were taken to try to simplify the code. Repeated questions were grouped into blocks which was a first step in reducing the number of lines of code. Each block could be repeated as necessary to collect the appropriate data for each household member. The paths through the blocks depended on which household members the respondent was answering for with the current question set as well as some global counters. Thus the block-level structure became more complex, but the body of code itself became simpler.

Introductory questions were grouped in one block, and parameters were used to pass appropriate data into the blocks of follow-up questions. As data was collected global counters were used to help determine the correct path. Global counters can create problems. For example, each question set was restricted for each respondent. The questions could only be repeated twice, so that no one was asked about having any more than two kinds of insurance. The number of insurances reported for each person was held in a global counter. Once that counter reached its limit, code was to be skipped. These counters were used in this application to determine the path through the blocks and whether or not a block should repeat, but they were dynamic. They changed as questions were answered by the respondent.

The global counters created two opposing problems. As the counter increased, certain sections of code were to be skipped which created a path change. Entry into a block depended on not having been there before and/or on the global counter as well as on the answers to the previously presented questions. As the counters were incremented, question sets moved from "on path" to "off path". This change in path without any intervention will lead to the loss of the data initially collected.

The second problem the global counter created is the counter example of the problem discussed above. Since the paths were dependent on the global counters, some instances caused a question to be added to the path from a block that was previously presented and presumed complete. The program would skip back to a question unexpectedly in that "completed" block once a counter increased to the requirements in the logic. To correct the presentation of extraneous questions from completed blocks and to prevent path change of previously presented blocks, if and keep statements were applied to the blocks. The block was only called if the "end" marker was empty. Once the "end" marker was encountered on the path and was answered the block was protected with a keep statement.

```
If end_block = empty then
      Present the block
Else
      Keep the block
Endif
```

This type of flow control is well documented in the Blaise literature discussing the protection of blocks from further change. (Statistics Netherlands (1999) Blaise Developers Guide, Blaise for windows 4.1, A Survey Processing System . p. 195). It is common to protect rosters and tables from change by the interviewer. We found that unexpected change created by the use of global counters and variables could be managed with the same techniques. To prevent the loss of previously collected data, the block calls were placed in 'if' statements with keep statements following the 'else' condition. In other words, when the questions were asked as part of the "if" code they were on-path, and when they were skipped within the "else" condition, data were protected by the keep statement.

As data was collected, this implementation essentially locked the administered blocks. The locking of these blocks prevented the interviewer from backing up to make corrections or adjustments but prevented loss of data regardless of which path was followed. The program went to production with this access-limitation approach in place.

Number of variables

As program design progressed, the size of the program and the number of backend variables quickly became large. Many of these variables were defined on a global level so that they would be available to all blocks and all modules. It was obvious that the number of variables should be reduced if at all possible so that the internal storage required within the Blaise application would not become so large that it would reach the application's limits. To this end, sets of classification variables were replaced with a single string variable. One was defined for each set of classification variables (14 of them), where the classification variable indicated a type of insurance coverage. A 15th classification variable was created and called "master". This master indicated if there was any insurance coverage of any type. These variables were defined as strings of length 17. Each column of the string represented a month and covered more than one year. Doing this reduced the number of global variables being stored for this particular piece of the project from 3808 variables (16 household members * 14 types * 17 months) to a mere 240 variables (16 household members * 15 types), a reduction in complexity within the instrument and more importantly on the resulting analytic dataset.

The string method for each classification variable was a very effective and novel approach to the problem of too many variables. Each column of the string represented a month. A method was devised to enable the columns to be set on and off as appropriate. Changing a single character in the string replaced setting an individual variable on or off. For example, the third character of the string represented the month of March. If the respondent answered "yes" for a month, such as by saying they had insurance coverage that

month, the third character would be set to “Y”. For this particular project, once a month was “on” it would not be turned off.

The 15th type, the master, was set if any of the types were set for a particular month. This allowed the program to evaluate a single variable to determine the insurance status of the respondent or of anyone in the household. Each household member had their own set of insurance “strings”.

Type1(i) := “NNNNNNNNNNNNNNNNN” initialized the entire string of months if type1 was indicated for the i th household member. At the same time the master for the i th household member was initialized.

Master(i) := “NNNNNNNNNNNNNNNNN”

Strings were left empty for all kinds of insurance that the respondent indicated were not used by a particular household member.

A sequence of questions asked to determine if any household members were covered by each particular type of insurance. Initializing the string to all off was the first step in capturing the data about the insurances carried by members of the household. A range of dates was captured through the question sequence, so the problem that remained was to collect the information together and set the appropriate columns of the strings from ‘N’ for no insurance that month to ‘Y’ for covered by insurance that month. The position of the column translated to the month covered.

A procedure was written to set the columns (months of coverage) of each string where the string represented the type of insurance (one of the 14 string variables). This procedure was also used to set the master string, representing coverage by any insurance carrier. The procedure used parameters generated from the collected data to turn appropriate columns on. Parameters were required. Inside nested procedures, parameters are the only way to be sure that the procedure is acting on the correct data. The procedure TypeSTR called a second procedure which set an individual column.

```
PROCEDURE TypeSTR
  PARAMETERS
    TRANSIT typestr : string
    TRANSIT MASTER : STRING
    IMPORT m : INTEGER {person index}
    IMPORT BMON : INTEGER
    IMPORT BYR : integer
    IMPORT EMON : INTEGER
    IMPORT EYR : integer
    locals j:integer
           k:integer

     RULES
{the j loop sets the first 12 columns of the insurance string }
     for j := 1 to 12 do
     set_ FLAG(Typestr, J, j, 2009, bmon ,byr, emon , eyr )
     set_ FLAG(MASTER, J, j, 2009, bmon ,byr, emon , eyr )
     enddo
{the k loop sets the last 5 columns of the string}
     for k := 1 to 5  do
     set_ FLAG(typestr, k+12, k, 2010, bmon ,byr, emon , eyr )
     set_ FLAG(MASTER, k+12, k, 2010, bmon ,byr, emon , eyr )
     enddo
```

```
ENDPROCEDURE
```

The first procedure, TypeSTR, used indices and affected the entire string. It called a second procedure, SET_ FLAG, which set only one column of the string to 'Y'. The loop in the first procedure called the second procedure to set the single column the number of times indicated by the data collected. Since the string variable for this project covered more than one year (17 months), the indices were defined to cover the first year (12 months) and then the next 5 months.

```
PROCEDURE SET_ FLAG
  PARAMETERS
{The variable string being set} TRANSIT FLAG : string
    loc :integer
    flagmon : integer
    flagyr : integer
{beginning month and year being set}
    IMPORT BMON : integer
    IMPORT BYR : integer
{ending month and year being set}
    IMPORT EMON : integer
    IMPORT EYR : integer

  AUXFIELDS
    BTmpDate : DATETYPE, EMPTY
    ETmpDate : DATETYPE, EMPTY
    refdate :  DATETYPE, EMPTY
    m : integer
    n : integer

  RULES
{make dates of parameters being passed}
    BTmpDate := TOdate(BYR,BMON,1)
    ETmpDate := TOdate(EYR,EMON,1)
{define a reference date that is today}
    refdate := todate (flagyr, flagmon,1)
    m := loc-1
    n := loc+1
{if the months being passed are within the dates to be set}
if refdate >= BtmpDate and refdate <= Etmpdate then
    if  loc = 1 then
{set first column}
          FLAG := 'Y' + substring(FLAG,2,17)
       elseif loc >1 and loc < 17 then
{set middle columns}
          FLAG := substring(FLAG,1,m) + 'Y' + substring(FLAG, n,17)
       elseif loc = 17 then
{set last column}
          FLAG := substring(FLAG,1,16) + 'Y'
       endif
endif
ENDPROCEDURE {SET_ FLAG}
```

The SET_ FLAG procedure used the parameters sent by the TypeSTR procedure. It converted the parameters to dates to compare to a reference date. If the dates were within range, it used the parameter j to turn the j^{th} column on. SET_ FLAG is called from a loop and will be called 17 times to change the column values appropriately, one at a time.

Conclusions

Survey application software can be pushed to its internal limits as the complexity of survey instruments increases, requiring creative thinking to meet the demands of the methodologists and use the applications on hand. Some of the tricks developed on this project allowed an increase in instrument complexity without negatively impacting the survey application.

The programming changes we discussed allowed the project to be successfully completed in spite of high complexity. Several lessons stood out from our experience:

- Programmers should carefully review the use of blocks of code and global variables.
- Our use of blocks of code was not unusual, but the repeated use of the same blocks caused some unforeseen issues that needed to be addressed. These issues were solved by using procedures and parameters.
- By redefining sets of variables to a single string variable, the number of variables being handled became manageable. Complexity of maintaining these individual string variables was handled by using nested procedure calls to update them as data was being collected.

The changes made to the initial programming design used thousands less lines of code and reduced the number of variables.

Acknowledgments

This research is based, in part, on work that was supported by the U.S. Census Bureau. The authors of this paper and the programming staff would like to acknowledge the U.S. Census Bureau and the members of the BOC Survey of Health Insurance and Program Participation (SHIPP) team for their help and patience as the programming for the project developed. The views expressed here are those of the authors and not necessarily those of the U.S. Census Bureau.

Longitudinal Survey Data – "Move It Forward"

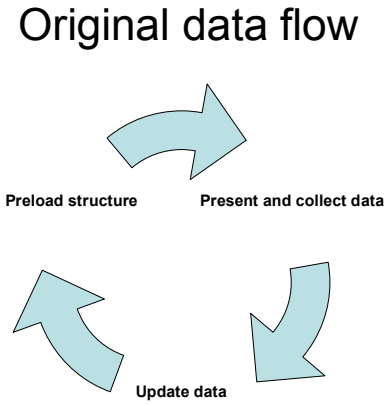
Rhonda Ash and Danilo Gutierrez, The University of Michigan

HRS Overview

The Health and Retirement Study (HRS) is a complex longitudinal survey covering a wide range of health issues and economic concerns related to aging and retirement. The average interview lasts more than an hour and is administered to over 22,000 respondents. The HRS began in 1992 and had been carried out for three waves at two-year intervals before it was merged in 1998 with its sister study AHEAD, which covered an older birth cohort and somewhat different content area. At that time two new age cohorts were added to fill in the age range above age 50, and a "steady-state" sample design was born which would maintain representation of the over 50 population by bringing in a new (younger) age cohort every six years. That change dramatically increased the ability of HRS to address the effects of events and policy changes within its scope and allows HRS to aspire to a very long lifespan with continual expansion and adaptation of its content as these changes occur.

HRS Preload Process - History

With a longitudinal survey about finance, family, and health, data collected from a prior wave could simply be presented and confirmed or updated, vastly reducing the time needed for the next interview. HRS developed a structure to house thousands of variables that were pulled forward from prior waves. As we were learning Blaise and the program's early version constraints in 2002, we made decisions about the preloaded data structure to attempt to ensure the program's performance would not be degraded. We developed a schema whereby the data was loaded into one location (structure) and referenced or updated in that location. As data was presented and changed the original location data would be changed. Hence, the term "preload" quickly became a misnomer.

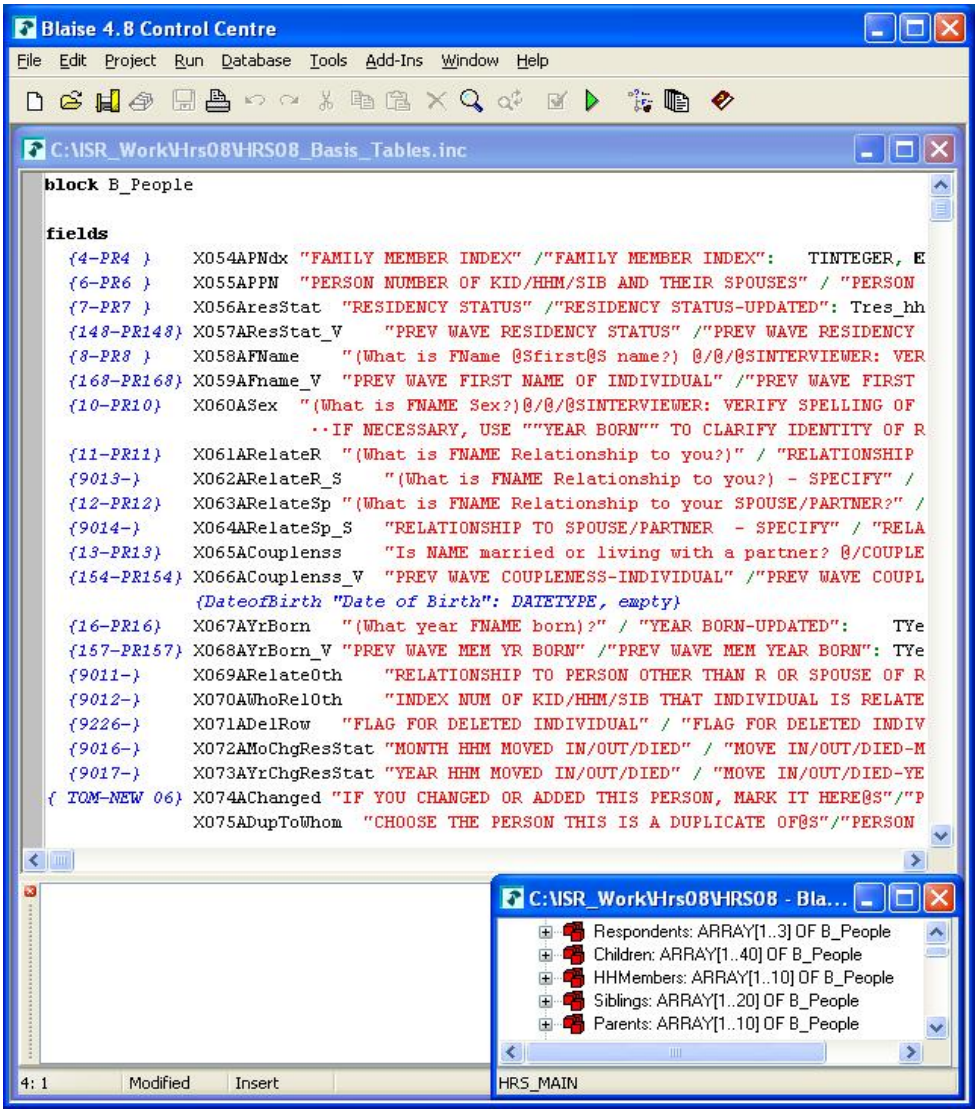


Early Blaise Restrictions

With an application the size and complexity of HRS we had to be very concerned with the amount of stored data that we carried in the instrument. Early in the process of working with Blaise, HRS ran into problems which caused us to be cautious about the number of fields in our very large instrument. The more fields we added to the route, the slower the application became until it failed -- or the RULES

would not compile at all. Since the preload structure contained a lot of fields, a scheme with a single copy of preload which would be updated as the interview progressed was baked into the design early. While learning Blaise, we quickly designed the original "write it back" process. A block of 25 fields that contained only people data was created in the coverscreen section of the instrument that collected such key information as respondent's name, gender, coupleness, and age. This block was replicated 83 times for the different types of people:

3 Respondents (current Respondent, current Spouse or Partner, any new Spouse or Partner)
40 Children and spouses with room for any new children if the Respondent recoupled
10 Household members who are not offspring
20 Siblings and
10 Parents



For certain variables we needed to compare the previous wave's statuses to the current wave -- for instance, married before (*X066ACouplenss_V)* versus married now (*X065ACouplenss)* -- in order to determine which questions or what version of a question to ask. As a result, we had to duplicate several fields in this preload structure and ensure that we never updated them. For example *X065ACouplenss* was

updated, to reflect current marital status, whereas *X066ACouplenss_V* contained the "virgin" preload value and *X066ACouplenss_V* was never updated.

Blaise Selective Checking and Gates

Certain aspects of Blaise, while helpful for many situations, presented certain challenges when dealing with the HRS coversheet's complexity. Some of these aspects are described here.

Selective checking: Blaise documentation stresses that

> **"**In a large instrument, there may be thousands of questions and thousands more rules that govern their implementation. In such a data model, the constant checking of all rules could overwhelm the computer. In order to know which rules to enforce, Blaise employs a selective checking mechanism based on parameters and blocks. Blaise uses parameters to optimize the performance of the checking mechanism during instrument use. By keeping track of parameters, both explicit and internal, Blaise knows which blocks to check."[5]

As the interview progressed, Blaise's selective checking mechanism would rerun the rules that were active "on the route" from the beginning of the application to the current field being asked. If we were updating fields that had already been on the route, that set of rules was rechecked.

Gates: Code that would stop the selective checking was added to the application to stop the process in certain conditions in order to take a field or block off the route.

KEEP statements: Used to maintain the data when a field is no longer on the route. When a field's value changed, other fields may be taken off the route; without KEEP statements the data in those fields are lost. With a different way of programming, we could dramatically reduce the number of KEEP statements needed.

Quite often we would add a field whose only purpose was to lock off other fields from the selective checking. Once we reached a point in the application, a field was presented and once answered, the logic of the RULES would remove fields or blocks from the route.

```
If A047_ = empty then
    Call Thatblock of fields
Else
   ThatBlock.KEEP
Endif
```

A question as simple as "Are you married?" became a programmer's challenge as the marital status changed and was written back to the original "Preload" field.

In this example, *Respondents[1].X065ACouplenss* was preloaded with a value of MARRIED. As the interview's coversheet progresses, we ask if they are still married to the same person at *A020TSameSpP*. If the answer to *A020TSameSpP* is NO, the value of *Respondents[1].X065ACouplenss* is changed to OTHER and the selective checking mechanism alters the route. Now *A026_Rmarried* shows up on the route. (*A026_Rmarried* asks a single respondent or a previously married respondent who is no longer married to the same person if they are now married to someone else.) If *A026_Rmarried* is answered YES the status of *Respondents[1].X065ACouplenss* becomes MARRIED once again. (Still with me?)

[5] Excerpts from Blaise 4.8 Online Assistant

```
Part 1:
IF (Respondents[1].X065ACouplenss= MARRIED OR
        Respondents[1].X065ACouplenss= PARTNERED_VOL))   THEN
        A020TSameSpP_A.Keep   {Keeps the data values when the field goes off the route}
        A022_.keep
        IF A020TSameSpP_A = empty then  {only ask if empty}
            A020TSameSpP_A {Is Clara still your wife?}
           If A020TSameSpP = NO then
                Respondents[1].X065ACouplenss := OTHER
           Endif
        Endif
Endif


Part2:

      IF A026_Rmarried <> empty {stop lock out when X065 gets assigned below}
      or
        ( piBA_HHX024_RelwHH_V = yes and
          (Respondents[1].X065ACouplenss= OTHER OR
          A020TSameSpP_A = NO )
        )  THEN
           A026_Rmarried
          If A026_Rmarried = Yes then
             Respondents[1].X065ACouplenss := MARRIED
          Endif
```

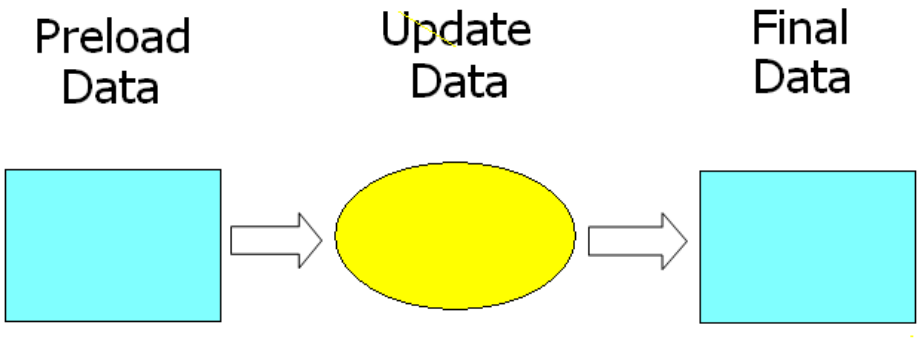
How To Fix The Problem

One of the more difficult issues with the "old" structure is that the selective checking mechanism unexpectedly changes the data values of fields on and off the route by re-evaluating the rules. After several waves of adding gates and reorganizing the code, the application became error-prone when new content was added or old content changed. In the example above, we referenced the *Respondents* block which is an array of three instances (current Respondent, the Spouse or Partner, new Spouse or Partner). We would load data from the prior wave into that arrayed block, and then alter the values as we progressed through the application.

To change that process we adapted a "Move it forward" philosophy. That is, once the data had been set into any field we would not re-assign it. We would move the data forward into a block or field that had not been altered.

We now have the same arrayed structure, but added a new location where the data values will not be changed. The prior wave's data is currently loaded into the *PRELOAD_Respondents* block. Any updates to the preload are recorded in the *SecA.Respondents,* , and the updated data are copied to the *Respondents* block. The *PRELOAD_Respondents* data are never altered after the data loading process, thus the Blaise selective checking mechanism is not triggered. The same philosophy is now followed for the *Children* data: we use three arrayed blocks for the preloaded, updated and final data. All updating is done in the coversheet section (*SecA*). Any reference past this coversheet section would look at the data in the final copies of each structure (*Respondents or Children)*.

Move it forward



Since it is important to maintain comparability in the instrument across waves with a longitudinal study, the thought of restructuring is nerve-racking. The data being collected must be able to be connected to the prior wave data, and be assured that the collection process does not change the meaning of any question's intent. The changes made were in how the data are stored, not in how the questions are asked. We are now able to store additional data structures for the *Respondent* and *Children* blocks. Restructuring allowed us to only alter the data in the update data locations and reduce or eliminate the back-writing. The code became easier to manage and understand. What took many lines of code before now takes only a few lines. We now need less code to handle the selective checking mechanism.

```
IF Preload_Respondents[1].X065ACouplenss = MARRIED OR
   Preload_Respondents[1].X065ACouplenss = PARTNERED_VOL   THEN
         A020TSameSpP_A

IF Preload_Respondents[1].X065ACouplenss = OTHER) OR
    A020TSameSpP_A = NO       THEN
         A026_Rmarried
```

Now, since *Preload_Respondents[1].X065ACouplenss* is never altered, we no longer need to protect against the selective checking mechanism. The rewrite gives us only one location of data to reference in future sections. The final data location helps to simplify and avoid interrelated problems between sections.

```
Respondents[1].X065ACouplenss :=  A038TCouplenss_A
```

Conclusion

Making any changes to our instrument, even small ones, was a matter for grave consideration because we never knew what unintended consequences might emerge. As a result, often times sometimes desired changes to improve question objectives were not implemented, especially if they were discovered late in the program development period.

Why was the coversheet rewritten?

- It was very difficult to manage code.
- Anticipating how the selective checking mechanism would behave became highly problematic in our very large instrument.
- The code became hard to understand especially for non-programmers.
- With any change, the whole instrument needed to be fully re-tested.
- We would have to check to make sure that data was not dropped "off the route" and lost.
- We had to make sure we did not unintentionally overwrite data.
- There were many interrelated problems between sections with the way the code was written.
- Changing the route in the coversheet adversely affected the logic in later sections. These effects were observed in other family data sections.

During the design of the re-write we considered these major issues. Each of these issues was examined and addressed. With the investment of time and planning in this re-write, the efforts will pay off into the future. The benefits have already been evident beyond our original hope. Now, with the easier code readability, non-programmers are able to look through the code and better understand the flow of the data collection. We are also much more confident that we can modify those areas of code without introducing unintended effects and endangering the whole instrument's integrity.

With the benefits that we have received from this investment, we plan to re-write other sections of the application. We have already applied this schema to the section that collects sibling data. Next, we plan on re-writing the section that collects children transfer data.

References

Statistics Netherlands, Blaise Department, Blaise 4.8 Online Assistant.

The Challenges of Implementing the China Family Panel Study (CFPS)

Jiahui Yao, Peking University, Gina-Qian Cheung and Youhong Liu, University of Michigan

1. Introduction

The Chinese Family Panel Study (CFPS) is the largest and most comprehensive national panel study ever undertaken in China. It is designed to collect three levels of data: individuals, families, and communities. The survey topics include society, economy, population, education, and health changes. The data are used for academic research and policy decision-making in China.

The CFPS survey has features modeled after the Panel Survey of Income Dynamics (PSID), the National Longitudinal Study of Youth (NLSY) and the Health and Retirement Study (HRS). The study is hosted by the Institute of Social Science Surveys (iSSS) of Peking University.

CFPS preparation began in 2005. Two paper-and-pencil surveys were conducted in Beijing, Hebei, and Shanghai in 2007. In 2008, CFPS had a pilot for 2,400 households in Beijing, Shanghai and Guangdong. A follow-up CAPI survey was launched in 2009. During the 2009 pilot, CFPS fully tested all CAPI systems, including Blaise data collection, real-time sample management, real-time technical support, data quality, and real-time monitoring. CFPS started production in April, 2010, and completed data collection in September, 2010. The total completed cases are about 85,000!

In this paper, we will first describe the CFPS CAPI system architecture. Then we will discuss the challenges encountered during implementation of Blaise.

2. CFPS CAPI System Architecture

As shown in figure 1, the CFPS CAI system includes the following three subsystems (discussed in detail below):

- Data Collection system
- Data Synchronization System
- Systems At iSSS Center

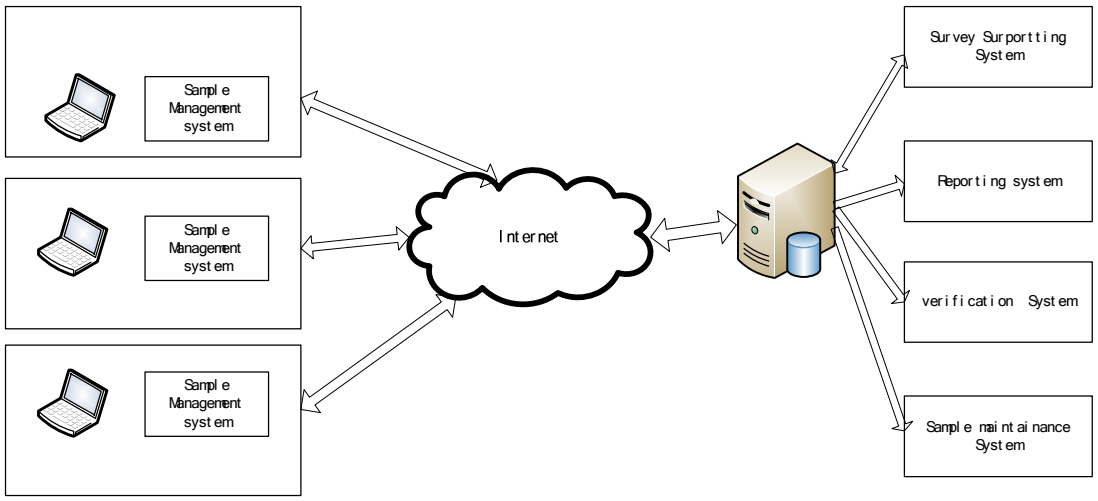
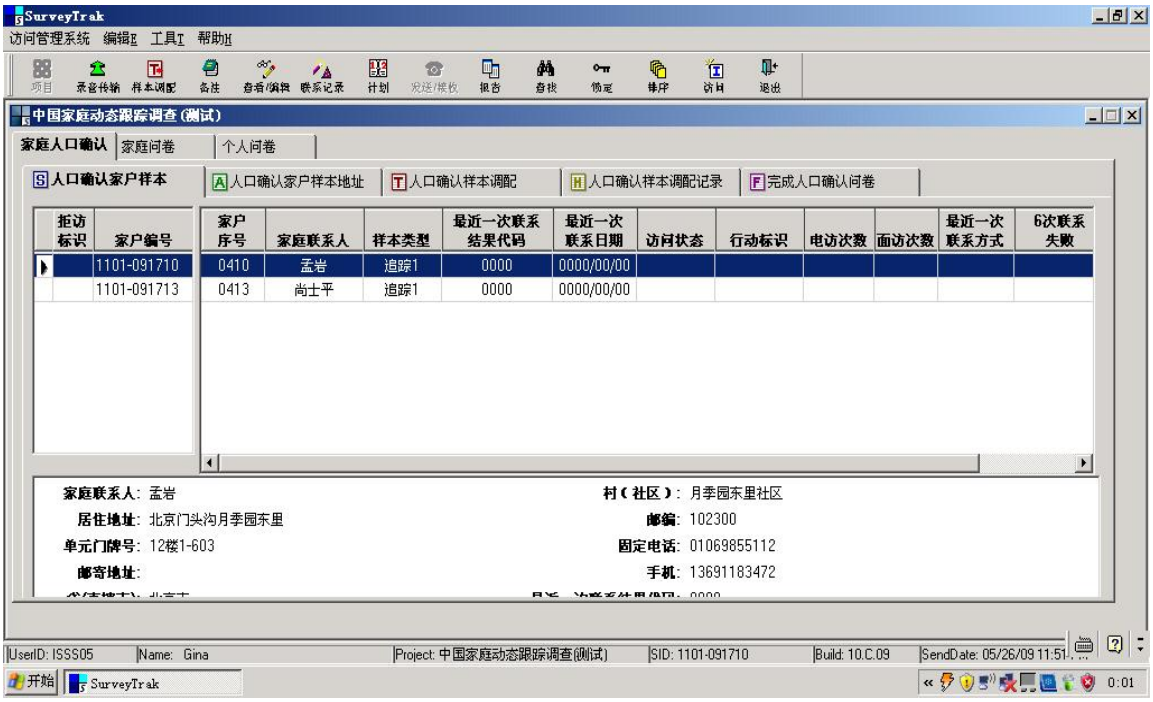


Figure 1 – CAI System

2.1 Data Collection System

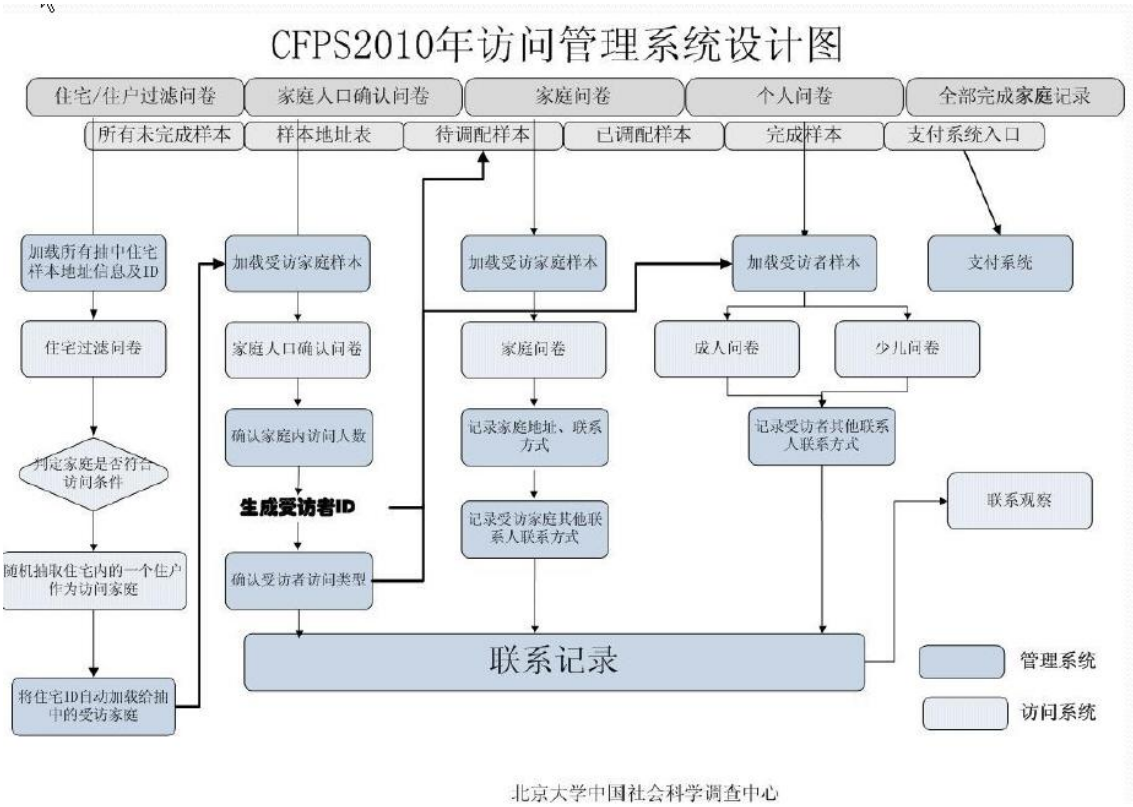
2.1.1 Sample Management System (SMS)

The CFPS Sample Management System evolved from the University of Michigan's Sample Management System, SurveyTrak. Staff from Michigan and iSSS worked closely together to develop the system for CFPS. The final product is all in Chinese.



2.1.2 Six Blaise Questionnaires

The six CFPS questionnaires are: Screening, Family Member Confirmation, Family, Community, Adult, and Child.



2.2 Data Synchronization System

The data synchronization system synchronizes the interview data and paradata between the interviewer's laptop database and the server database. The system has both server side and client-side databases. The basic principles are as follows:

- All database operations are recorded in the log table (such as add, delete, modify), and the last successful synchronization time (both upload and download) is recorded to another table;
- When uploading data, the data synchronization system compares the log table with the last successful upload time, and if there are differences, the latest updated data is uploaded to the server;
- When downloading data, the data synchronization system compares the log table with the last successful download time, and if there are differences, the latest updated data is downloaded to the client database.

2.3 Systems at the iSSS Center

To ensure production data collection runs smoothly and with high quality, several systems have been built to run at the iSSS Center with the server database.

2.3.1 Survey Supporting Systems

- Sample Transfer System - This is used by supervisors to transfer sample from one interviewer to another interviewer.
- Technical Support System - Supervisors use this system to submit technical issues raised by interviewers. They assign the problems for timely resolution to appropriate technical staff.
- Sample Help System – This is used to process special sample such as sample that were refused three times or that could not be contacted after six tries. In such cases, the iSSS Center contacts the respondent and assigns a special status code to the sample, and then informs the interviewer whether to further contact the respondent.
- Interviewer Management System – This is used to manage the interviewer's information, such as adding a new interviewer or modifying information for existing interviewers.

2.3.2 Reporting System

SAS Web Report Studio is used to report on daily survey processes, such as sample completion and sample distribution. The data are shown in dashboards, tables and curves.

2.3.3 Data Verification System

The data verification system selects samples to be verified according to the values of a selected questionnaire's variables and other user backend database variables. It is a web system using BlaiseIS. The verification can be conducted by telephone, by review of sound recordings, and by face-to-face interviews.

2.3.4 Sample Maintenance System

This system is mainly used for sample tracking and maintenance after the completion of interviews. Similarly, this is a web system that uses BlaiseIS to complete questionnaires and to record the results of sample maintenance. The other function of the Sample Maintenance System is to send respondents customized holiday greetings through Short Message Service and E-mail.

3 Challenges

3.1 Implementation Challenges in SMS Development

3.1.1 Implementing family level of survey with five subprojects

- The entry point of the family level survey is the Household Screening Survey. Certain criteria, such as how long the family has lived in its current location, are used to determine if a family is eligible for further interviews.
- The second survey is the Family Member Confirmation Survey, in which all members in the household are listed in a table of the Blaise instrument. After the survey, SMS generates sample lines in the Adult or/and Child projects. The number of projects generated is based on the answers from the Blaise family list table. For example, if there are two people in the family who are older than 16, then two sample lines in the Adult project will be generated.

Both of these surveys are screening tools used to select families and individuals to be interviewed. The main surveys are the Family Overall, Adult, and Child surveys

3.2 Challenges for Blaise Development

In this section, we will describe details that are unique for implementing the CFPS in Blaise.

3.2.1 Setting up the Blaise Chinese Environment

In 2003, University of Michigan had a paper discussing how to display Chinese in Blaise. It mentioned that the NJ Star program was used to display Chinese for a multi-language Blaise instrument. Since CFPS is not a multi-language instrument, it is not necessary to use a third party program such as NJ Star to display Chinese. There are certain computer system settings that can be changed to enable Blaise to display Chinese properly.

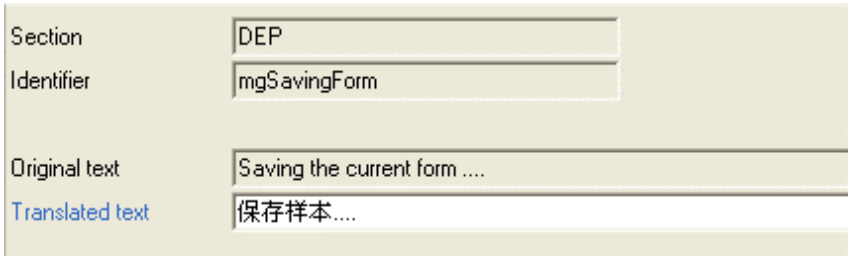
The following two changes to the Windows operating system are needed in order to display Chinese characters correctly:

1. Control Panel->Regional and Language Options->Advanced->Select a language to match the language version of the non-Unicode -> **Chinese (PRC)**
2. Control Panel->Regional and Language Options-Regional Options->Standards and formats->**Chinese (PRC)**

3.2.2 Translate Blaise System Text to Chinese

The above two settings in Regional and Language Options help to deal with Blaise questions and answers in Chinese language. To change Blaise System text to Chinese, we use the Blaise translation tool to set up a language INI file in Chinese. The INI file contains translated menus, dialogs, forms and error messages.

1. Translating: The basic object in the translation process is a Blaise database. *BlTxtTra.bdb* contains all the system text. The Blaise setup for this (*BlTxtTra.bla*) and the text database itself can be found in a subfolder of the Blaise system folder called *Translat*. To Translate: 1) open BlTxttra.bdb with Dep.exe; 2) **Setting Up the Blaise Chinese Environment** ocate the section and identifier that need to be translated; and 3) in the translated text field, input the Chinese text. See the figure below:



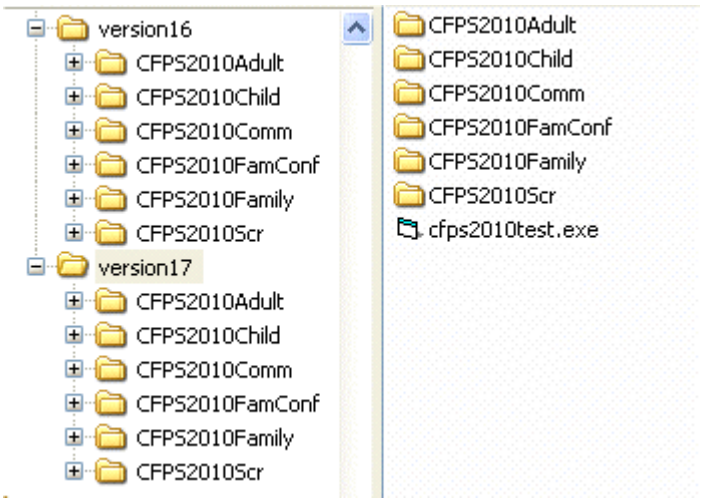
2. Transforming the translation to the INI file format: The Manipula setup BlTxt2IN.MAN in the Translate will conduct the transformation for us. The output of this Manipula program is a file called *Blaise.tra*.
3. Using the new language file: There are several ways to use the new language file. CFPS chose to use a DEP.exe parameter e.g. /#C:\blproj\CFPS\Blaise.tra.

3.2.3 Testing System for CFPS

There are six datamodels for CFPS 2010. As development evolved, it was necessary to build a testing system so that programmers could setup and update datamodels easily for the testers to test the latest changes. For quick development, VB6 was selected to construct the system.

The Testing system folder structure: When a new version is ready to test, first a version folder is created, e.g., version16. Second the VB6 cfps2010 is copied to the folder. Then, the datamodel files are copied to their corresponding directories.

By implementing this type of folder structure, a Blaise programmer can easily create and send the latest version to testers, and the older versions are still available for change verification.



3.2.4 Modify Blaise Preload in the Testing System

Some of the CFPS datamodels have complicated preload block structures. To ensure that different preload sceneries can be tested, the testing system must enable testers to modify preload values. The following steps in the Blaise datamodels and VB programs are set up for this purpose:

1. Define an Auxfield in the Datamodel: PreLoadForTesting
2. In the Datamodel's rules section, add the following code

PreLoadForTesting.KEEP

IF PreLoadForTesting = 1 THEN

{To enable preload modification during test}
Preload
ENDIF
Preload.KEEP

3. In the cfps2010test.exe VB program, the DEP parameters assign a value to this Auxfield. *i.e., /Q=PreloadForTesting=1*.

All three steps are defined so that in the testing environment, users can change preload field values freely because "PreloadForTesting=1" is fed to the database form. For the production environment, the preload block is locked. This is because the field assignment is not included in the DEP parameters. This is an elegant approach, because no code changes are required between testing and production to avoid potential programming errors.

The CFPS testing system described above is effective. However some system enhancements are required. For example, Bug Log is not implemented; users need to put their testing comments in a separate Excel or Word file. At the time development was taking place, CTT was not ready for CFPS, due to some conflicts between Chinese characters and CTT's backend SQL Server database design. With some database modifications and small code modification, CTT will be ready for testing Chinese projects.

3.2.5 Blaise Word and Math Tests in CFPS

CFPS2010's Adult and Child instruments include tests for math proficiency and word recognition. One of four sets of math tests and one of eight sets of word tests are randomly assigned to a selected respondent. The following steps are implemented in the Blaise instruments:

1. In the Family Confirmation instrument, we calculate which test group number a family would start with: RandomNum := Random (8) + 1. This number and the person's index number in the household list table are preloaded in the Adult and Child instruments.

2. In the Adult and Child instruments, the calculations below are used to determine which test group is used by the respondent. These calculations ensure that members in a family are assigned to different test groups:

   WhichMathList := (RandomNum + RespdentIndex-1) Mod 4 + 1
   WhichWordList := (RandomNum + RespdentIndex-1) Mod 8 + 1

3. For the word recognition level test, an external look up database was utilized to store the word list - see the structure and data below, note that there 34 words in each group:

\\SRC-Douvan\SRO\share\yliu\Blproj\CFPS2010\Dev\externalbdbs\

WordNumber	Group1	Group2	Group3	Group4	Group5	Group6	Group7	Group8
1	木	入	田	火	出	广	书	方
2	朵	色	比	真	乡	画	休	兔
3	贝	乐	桥	冬	父	刚	林	叫
4	百	丢	并	斗	丰	定	顾	连
5	如	甲	歌	际	折	铅	亚	景
6	航	农	虽	肯	科	史	困	欠
7	岛	末	鲜	股	瓦	冒	季	杜
8	扮	环	寿	芒	冈	滑	亏	壶

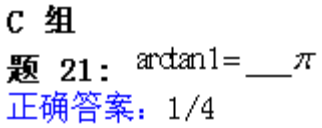
In the instrument, an array is used to store data read from the lookup database:

```
FOR I:=1 TO 34 DO
    WordData.Search    (I)
    WordData.Read
    IF WhichWordList = 1 THEN
        Words [I] :=  WordData.Group1
    ELSEIF WhichWordList = 2 THEN
        Words [I] :=  WordData.Group2
    ELSEIF WhichWordList = 3 THEN
        Words [I] :=  WordData.Group3
    …….
        ENDIF
    xWordFill:=Words[I]
    QWordTest [I]
ENDDO
```

The individual tests are represented in a field array. The fill xWordFill is determined with the above logic.

```
QWordTest (WordTest)
"@F&@F@I [出 示 字 表 ^GroupLetter[WhichWordList] ]  @I
@/@/字表^GroupLetter[WhichWordList]第@C^I@C个字: @I^xWordFill
@/@/@B访员注意: @B不能读出来!
@/@|请受访者用普通话读词. 对音调、前鼻音、后鼻音、平舌、翘舌不做区分.
@/@|不允许受访者查字典或询问他人.
@/@|如果受访人不知道或拒绝,输入""5"",F2输入理由  @I
 "  /"字词 - ^I" : ARRAY[1..34] OF (_1 "对", _5 (5) "错")
```

4. For the math proficiency test, html files are used. The files are named with a specific format, i.e., x_y.html, where x indicates the group number and y denotes the test number in the group. For example, 2_10.html is for test number 10 in group 2. The benefit of using html files is that we can link images into the html files. For example, in the figure below, the math representation cannot be presented by text. The technique here is to put the math image in an image file, then link the image file to the html file. The image files can be generated by a non-programming person to improve efficiency.



In the instrument, the file name is determined by the following code:

```
xQMathTest := str (WhichMathList )+ '_' + str(I) +'.html'
```

The field text in the math test is shown below. The xQMathText fill is the file name from the string concatenation above:

```
QMathTest (MathTest)
"@F&@F@I [出 示 数 学 表 ^GroupLetter[WhichMathList]]  @I
@/@/说出^GroupLetter[WhichMathList]组第@C^I@C题的答案:
@/<html><a href='c:/blproj/cfps2010/mm/^xQMathTest'>按这里看题目:</a></html>
@/@/@I@B访员注意: @B
@/@|记录受访者的答案!
@/@|测试过程尽量让受访者独立思考.
@/@|不允许受访者使用计算器或询问他人.
@/@|如果受访人不知道或拒绝,输入""5"",F2输入理由  @I
" / "数学-^I" : ARRAY [1..24] OF (_1 "对", _5 (5) "错")
```

Database

Just as at the University of Michigan, CFPS2009 used SQLAnywhere as our sample management database. Due to the availability of version 6 of SQLAnywhere in China, we had to update the database to SQLAnywhere version 11 with the help of our colleagues at Michigan. There were many problems encountered due to the instability of SQLAnywhere version 11. The main issue is data synchronization.

With the above problems in CFPS2009, we decided to migrate the database to MSSQL2005. The major effort for this migration was to develop a new data synchronization function that integrates the new database platform and the sample management system.

The new synchronization function operated smoothly at the earlier stage of 2010 interview. After two months, synchronization became very slow and problematic when the database reached 3 Gigabytes. iSSS and Michigan worked together to find a solution – regular database backup and weekly database clean-up.

3.3 BlaiseIS

CFPS used BlaiseIS in two systems – Data Verification and Sample Maintenance. The major issue with our original design for BlaiseIS was preloading data into the Blaise database, which would lock the database, preventing data saving after completion of a BlaiseIS interview.

After consultation with the Blaise developer, we found that the preload operation needed to be performed before the sample is accessed from BlaiseIS's web pages. After implementing this protocol, the database-locking problem was resolved.

4 Conclusion

After just one and half years of using CAI and Blaise, the iSSS Center has gained tremendous experience. Of course, there are many areas of improvement needed, such as:

- Capturing Journal data in BlaiseIS;
- Testing the BlaiseIS instrument (this will be accomplished by modifying the University of Michigan's CTT system as mentioned above);
- Implementing a CATI capability in future surveys.

In the process of implementing CAPI, the iSSS Center encountered many challenges and difficulties. Working with colleagues at the University of Michigan, they met these challenges. The Blaise developer also answered many questions, whether they were simple or complicated. And above all, iSSS successfully completed the CFPS2010 survey. It is a landmark in Chinese social research history.

Customizable .NET Event History Calendar: Looking to the Future

Roberto V Picha, Daniel Moshinsky, Mecene Desormice, Seth Benson-Flannery, U.S. Census Bureau

1. Introduction

The U.S. Census Bureau's survey of Income and Program Participation (SIPP) provides monthly information about the nation's income, wealth, and program usage. Currently, SIPP administers three interviews per year to each sample member; each interview's reference period covers the preceding four calendar months.

In 2006, the Census Bureau initiated a SIPP re-engineering effort, a key component of which is a shift to a single annual interview covering the preceding calendar year. An Event History Calendar (EHC) is employed to help address the concern that the longer reference period may impact respondent recall. To study the implications of the switch to the EHC method of data collection and the expansion of the survey's reference period, it was decided to conduct several field tests prior to the final production implementation.

In the beginning of 2010, the first field test of the re-designed SIPP instrument was conducted. There were 7,982 sample cases for this field test. This paper will describe some of the initial findings of the field test and lessons learned as they relate to the use of the Re-engineered SIPP (ReSIPP) instrument and its EHC for a second field test which is scheduled to take place in early 2011.

2. Background

ReSIPP contains some 40 sections. This includes sections relating to employment history, government program use, health insurance coverage, assets, utilization of health care services and demographic information. The ReSIPP is a survey that interviews and collects data for each household member, directly or by proxy for people over 15 years old and by proxy for children under 15 years old. Not all of SIPP questions are well-suited for being asked in the EHC.

It has been shown that the Event History Calendar method of interviewing is more effective than the standard question list method in improving recall of dates of autobiographical events for most topics (Belli et. al., 2001). Therefore, the types of ReSIPP sections that are asked in the Event History Calendar contain questions involving the recall of dates for changes in status throughout the reference period. For example, questions about the dates of job changes or changes in receipt of benefits are collected through the EHC.

On the other hand, questions relating to topics such as assets pertain to the reference period as a whole (e.g. "What was the combined value of your assets in the previous calendar year?") and do not involve recall of dates. Topics such as these have no transitions and are included outside the EHC section of the instrument. Thus, these sections can still be asked in the standard question list method and can be programmed in Blaise.

3. Features of the 2010 Instrument

For the field test in 2010, we designed a customizable EHC, programmed in C# .NET, and invoked it from Blaise 4.8 as a COM object DLL.

Twenty-five of ReSIPP's forty content topic sections were collected in the EHC. The interview began with introductory and demographic questions collected in Blaise. This was followed by the launching of the EHC. The respondent would answer the next 25 sections (containing several hundred questions) in the EHC environment. Then, the respondent would return to the Blaise DEP to complete the remainder of the interview.

Parameters from the Blaise DEP were passed into the EHC via the Blaise API, and data collected in the EHC were saved via the API in the Blaise database. One common Blaise database was used to store all data for the case. Question text, answer lists, topics, rules, and screen layouts within the EHC were created dynamically based on the Blaise datamodel and several external metafiles. Nearly all survey-specific content was described in the Blaise metafiles without rules – the C# EHC functioned as a user-interface and a shell for displaying that content. This design is described in full detail in the paper presented at the Riga Conference (Daniel Moshinky, Mecene Desormice, Seth Benson-Flannery, 2009).

The result was an easily customizable instrument that yielded a robust data collection system that not only included all the features of an Event History Calendar, but also aimed to mimic the question flow and error checking that our users have come to expect from Blaise instruments.

4. Results from the 2010 Field Test

Feedback from the 2010 field test has come from direct observations of live interviews, as well as from written and oral comments from field representatives and their supervisors who were queried during a number of debriefing sessions.

On the whole, the majority of debriefed interviewers have rated the Event History Calendar as a very positive experience and have commented on how much they enjoyed the greater freedom and conversational style of data collection with the EHC. The interviewers also offered a number of criticisms and constructive suggestions of the 2010 tests.

4.1 Survey content

The overwhelming consensus of the Field Representatives (FRs) was that the survey was much too long, with interview times of several hours being common. In addition, FRs complained that many of the questions were repetitive and redundant. For example, many of the questions asked who in the household was covered by various benefit and health insurance programs. When subsequent household members were interviewed, the instrument re-asked those same questions again, instead of pre-populating the fields based on the answers already collected. As a result of these shortcomings, respondent engagement waned, with many respondents becoming restless, annoyed, and uncooperative. It became clear that the survey will need to be shorter and "more intelligent" if it is to maintain the cooperation of the respondents.

4.2 Usability

Overall, contrary to our concerns about the FRs acceptance of this new technology, most did not report any problems using the EHC. Of the FR comments made available to us, none commented about problems within the Event History Calendar environment. Also, no problems navigating the EHC were

reported to our technical support staff during the interview period. This does not mean that the EHC was problem-free. It is more likely that the FRs focused their comments on the problem that was most pressing to them – that of the survey length and content.

4.3 Use of EHC techniques

Because audit trails and audio recordings are still being analyzed, full results about the patterns of the FRs' use of EHC interviewing techniques to aid respondent recall are not available yet. However, early feedback shows that the FRs were often confused about the meaning and purpose of certain topics such as the "Landmarks" line (some suggest it be renamed to something more descriptive, such as "Life Events"). The FRs used other topic lines for cross referencing more than Landmarks.

4.4 Data

Data output from the 2010 field test is still being reviewed. The initial results show good data integrity resulting from our single-database design approach, but it is too early to draw any further conclusions about the quality of the data collected using the EHC method.

In summary, the FRs we have debriefed strongly warn that the response rates in subsequent interview waves will drop significantly if the length of the interview is not shortened.

5. New requirements for the next field test

A number of major requirement changes were proposed to us to accommodate FR feedback, reduce respondent burden and improve data consistency. Firstly, the EHC reference period was to be extended to the interview month – resulting in a reference period ranging from 13 to 18 months, depending on the interview date. This was done in order to be able to feed data into subsequent interview waves that would jump-start interviews and reduce the number of questions asked.

Secondly, access to a number of EHC topics that depend on the respondent's income level would depend on a few income screeners asked in the beginning of the interview. This reduces the number of questions asked of respondents whose income is too high to be eligible for the programs involved.

Thirdly, common information would be copied across household members. For example, answers about marital status would be copied from one spouse to the other, and answers about residency would be copied across all household members who have lived and moved together as a unit.

Lastly, we were asked to create the "Time without a job" line that would be automatically computed based on the dates provided on the "Job" lines– rather than asking the FR to enter the "begin" and "end" months manually for respondents that reported being out of work as it was done in the 2010 field test.

In addition to these new requirements, we needed to address some technical shortcomings of the 2010 EHC instrument. As the number of questions asked in the EHC grew tremendously over the course of the EHC development, so did the complexity of the universe statements and edit checks involved. For the most part, the rules functioned as specified, but in some situations there were problems with rule re-execution when the FRs backed up and changed values inside the EHC. With the ever-growing complexity of requirements, our existing mechanism of using a metafile and some auxiliary functions (as described in our 2009 IBUC paper) became quite difficult to maintain. It was decided that a much more efficient and easier way would be to delegate all rule processing to the Blaise engine.

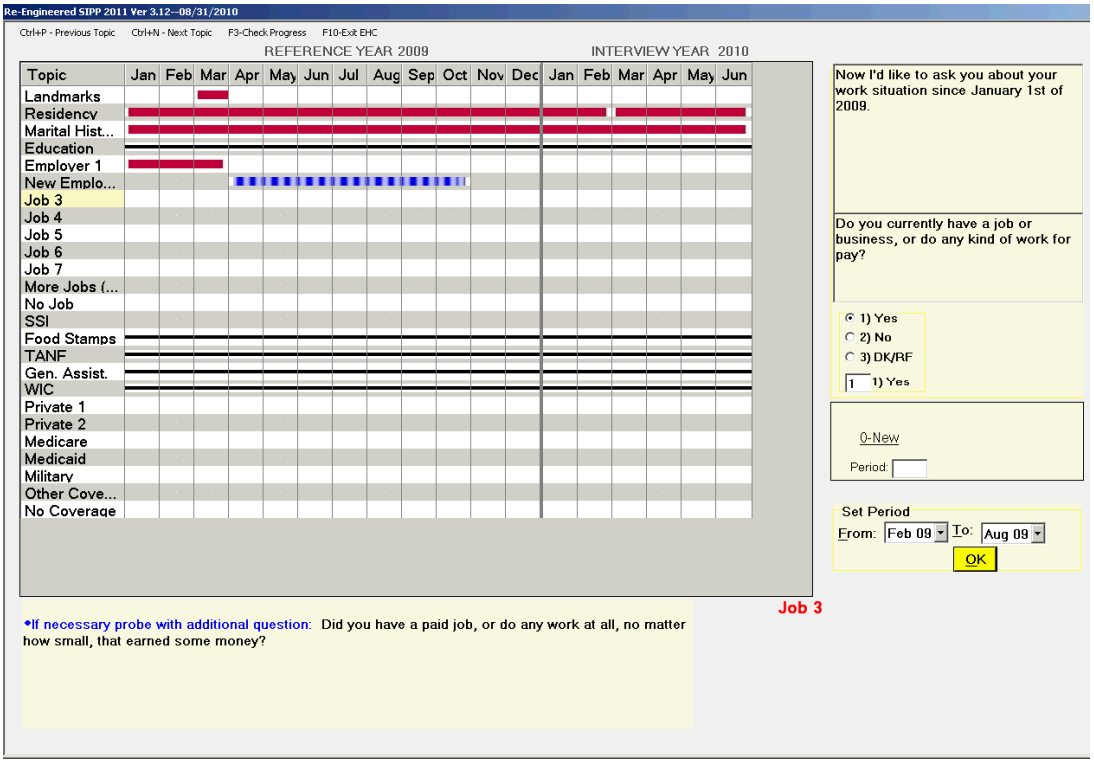
At first, we attempted to display the topics' detailed follow-up questions in the EHC by using the BCP and the Blaise API to re-execute the rules, evaluate universe conditions, and determine the next field on path. However, this approach would require us to store values to the Blaise database and force rules re-execution after every single entry in the EHC. Considering the size of the instrument, this approach carried a considerable performance burden.

We decided that the most straightforward approach would be to collect the "begin" and "end" months in the EHC, while moving the follow-up questions into the DEP and simply toggling between the two environments during the interview. Interview time would be reduced by expanding the use of screener questions to screen out respondents who are not eligible for certain sections (e.g. people with household income above a certain threshold), copying available data from one household member to another, automatically creating periods of unemployment, and improving the way the user interface guides the interviewer to the next action.

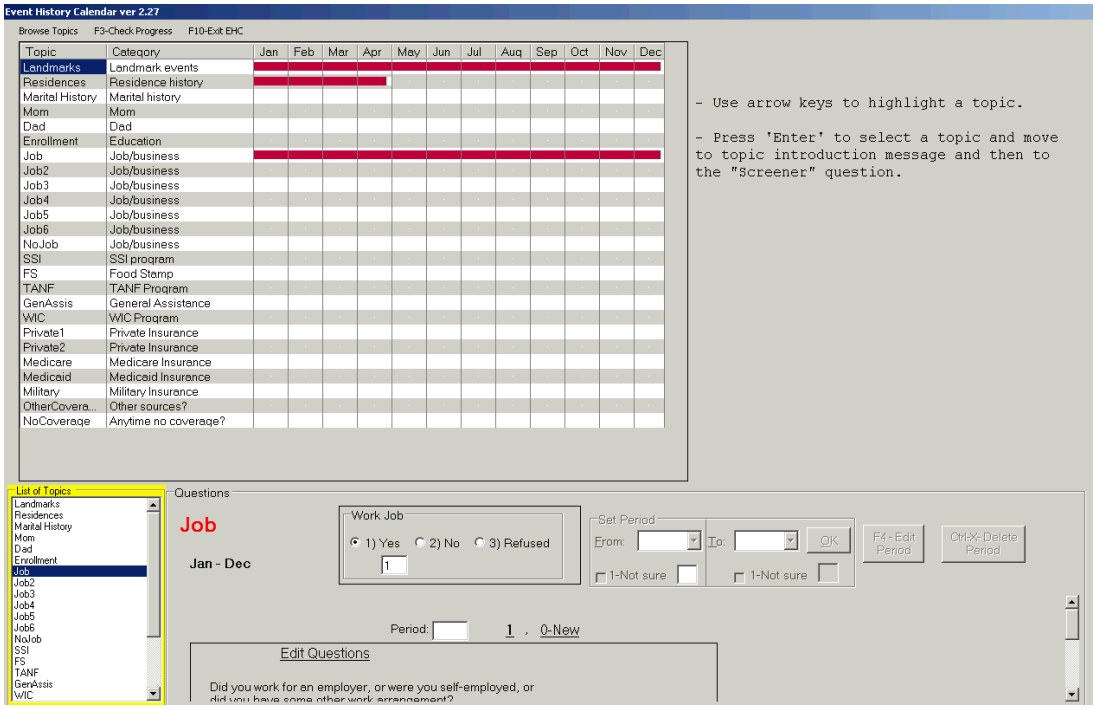
6. Design of the 2011 ReSIPP EHC

After the EHC is first launched, the user is taken to its main screen, which contains a list of topics and timelines. After a topic is selected, the user is asked one or two screener questions, and asked to create a spell. After the user creates a spell, the EHC main screen closes, and the user is taken to the DEP to answer the detailed follow-up questions for that spell period. After the last question in that topic (or earlier if a function key is pressed), the EHC appears on the screen again, and the user can either enter another spell period for the same topic, or begin data collection for a new topic. Thus, whereas in 2010 all of the follow-up topic questions were asked within the EHC, in 2011 they are being asked in the Blaise DEP.

The graphic below shows the new EHC screen for the 2011 field test.



The graphic below shows the EHC screen used in the 2010 field test.



6.1 Entering data in the DEP from the EHC

Within the ReSIPP instrument, each topic is stored as a block. (Depending on the topic, each of these blocks may also contain additional blocks.) All of the topic level blocks are stored in our TEHC block.

Upon leaving the EHC to go into the DEP portion of the instrument, three auxfields are populated via calls in the C# code. In the code presented below, these are **showQs** , which represents the topic selected in the calendar; **prdSelected**, which represents the instance of that topic (and spell period) that has been selected; and **AskReturning**, which is a flag set to "TRUE" if this is not the first time entering a particular period and "FALSE" otherwise.

If both **showQs** and **prdSelected** have a value, then we are to ask the detailed follow-up questions for a particular topic and period. In the case where **AskReturning** has been set to "TRUE," we will first ask an auxfield named **Returning** that only allows a value of "1." The purpose of this field is both to notify the user that they are about to enter a spell in which they have already entered data and to act as a gateway into a previously entered spell. When we leave the EHC, we need an empty question on route to land on. In the case where data have already been entered, **Returning** assumes that role. After **Returning** is answered, or if **AskReturning** = "FALSE", the user is taken into the **TEHC** block, passing in **showQs** and **prdSelected** as parameters.

```
AUXFIELDS
 {...}
  prdSelected  : INTEGER
  ShowQs       : STRING[200]
  AskReturning : STRING[5]
  Returning
             ENG "@/@Zs@Z @LYou are entering this period to edit your data.
                  Press '1' to continue.@L" / "Returning"
              : ARRAY[1..20] OF TContinue, NODK, NORF
{...}
RULES
{...}
 IF showQs = RESPONSE AND prdSelected = RESPONSE THEN
    IF UpperCase(AskReturning) = 'TRUE' THEN
       Returning[I]
    ENDIF
      TEHC[I]({import} CTRLNUM,I, {... more parameters} )
 ENDIF
{...}
```

Within the EHC block, these parameters (called **IN_showQs** and **IN_prdSelected** in the code below) are used to place the spell level questions on route. Using this method, only one instance of one topic can be on route at any time. (Please note that, although it is not displayed in the code below, when using this technique, all of the spell level blocks need to have a KEEP statement associated with them.)

```
PARAMETERS
    IMPORT In_ShowQs : STRING
    IMPORT IN_prdSelected : INTEGER
    {...}
RULES
  IF (uppercase(IN_showQs) = uppercase('BLandmark') OR uppercase(IN_showQs) = uppercase('BNoJob') OR
    uppercase(IN_showQs) = uppercase('BMedicaid') OR uppercase(IN_showQs) = uppercase('BNoCoverage')) THEN
    FOR I := 1 TO 6 DO
      IF I = IN_prdSelected THEN
        {...}
        IF uppercase(IN_showQs) = uppercase('BLandmark') THEN
          BLandmark[I]({Import} CTRLNUM, LNO, I,{... more parameters})
        ELSEIF uppercase(IN_showQs) = uppercase('BNoJob') THEN
          BNoJob[I]({Import}CTRLNUM, LNO, I, {... more parameters})
        ELSEIF uppercase(IN_showQs) = uppercase('BMedicaid') THEN
          BMedicaid[I]({Import}CTRLNUM, LNO, I, {... more parameters})
        ELSEIF uppercase(IN_showQs) = uppercase('BNoCoverage') THEN
          BNoCoverage[I]({Import} CTRLNUM, LNO, I, {... more parameters})
        ENDIF
      ENDIF
    ENDDO
  ELSEIF uppercase(IN_showQs) = uppercase('BResidency') THEN
    FOR I := 1 TO 5 DO
      IF I = IN_prdSelected THEN
        {...}
        BResidency[I]({Import} CTRLNUM, IN_LNO, I{... more parameters})
      ENDIF
    ENDDO
```

6.2 Returning to the EHC from the DEP

When the spell period level detailed questions have been answered, the interview is directed back into the EHC calendar. This is done by using an event to call a COM-object DLL associated to fields at the end of the block of detailed questions.

Because each spell period may contain many different variables that could be the last one on route, we opted to create a new shared **EndSpells** block that comes on route at the end of detailed questions for all topics. Within this new block, we have three questions that could be the last one on route (e.g. "Do you have any more periods to report?"). The last variable triggers the event to call the EHC DLL, which will determine whether to open up the EHC calendar or do nothing. If instrument logic dictates that the EHC is to be launched after the first question, then code within the DLL control will launch the calendar. Otherwise – if we are not ready for the calendar to be launched -- the code within the DLL control will simply return control back to the DEP without launching the calendar, and the focus will go to the next field on route. By coordinating the Blaise code and the C# code, we are assured that we always return to the EHC calendar at some point in this new block.

After the EHC is re-launched, DEP fields are used to guide the focus on the EHC screen and help guide the interviewer's transition. For example, if the interviewer indicated in the DEP that there are more time periods to report within the current topic, the focus is guided to the EHC field at which a new period can be created. On the other hand, if it was indicated that there are no more periods to report (or if the entire reference period has been covered), then the focus is guided to the calendar grid and onto the subsequent topic. Still, FRs maintain the power to navigate freely within the EHC and can enter any topic at any time. The intention is to aide the flow of the interview and to reduce the number of FR key strokes while preserving the flexibility to enter data in a free form way consistent with the conversational style of data collection within the EHC.

The transfer of follow-up questions to the DEP offers several key advantages. Firstly, the full power of the Blaise rule-checking can be used to route the instrument. Secondly, the inclusion of the EHC questions within the DEP provides users with a consistent user interface and makes the instrument appear more seamless and easier to use. In addition, the DLL launches quicker than it did in the 2010 instrument because less data now needs to be passed via the API.

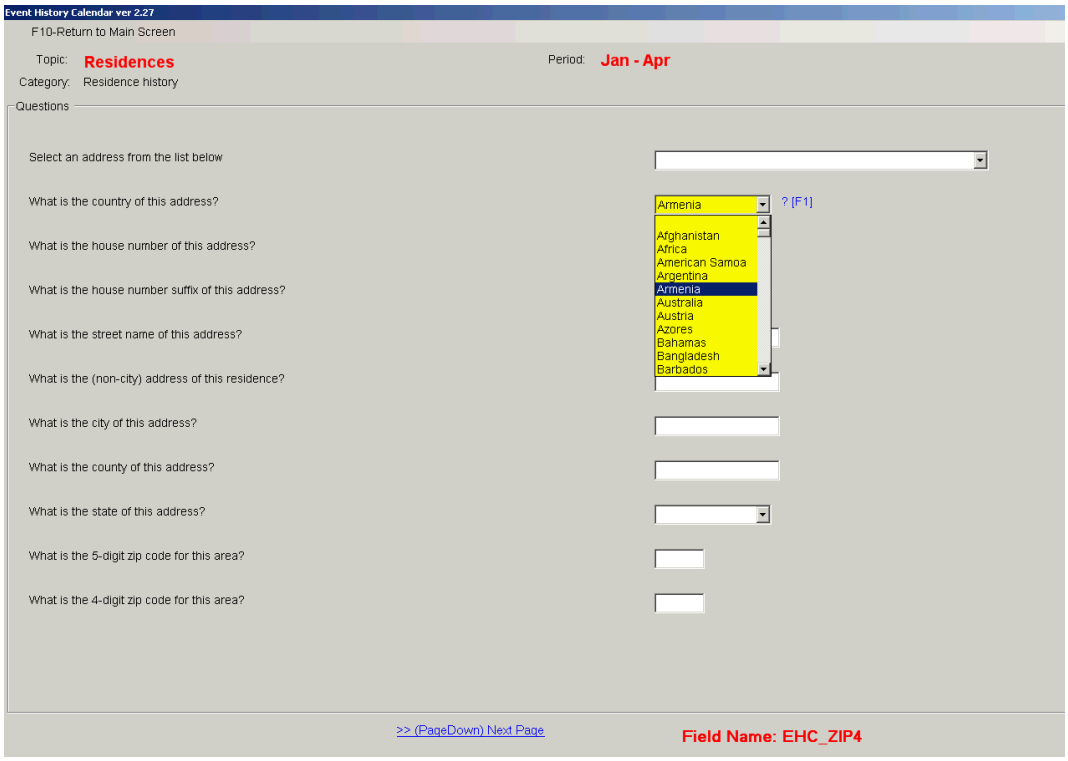
7. Challenges

7.1 Transition to the new EHC Design

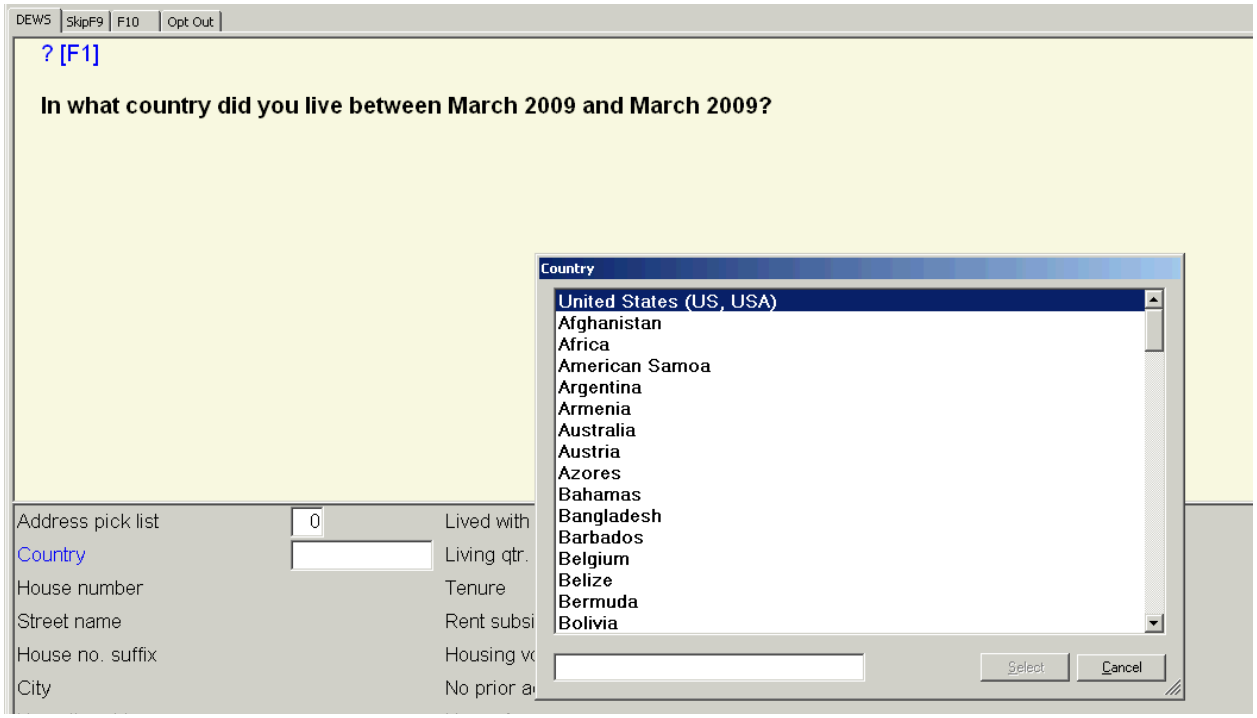
From a technical standpoint, the transition to the 2011 approach was fairly painless. All of the Blaise fields and blocks already existed in Blaise and were being read in by the EHC in the 2010 instrument. Rules had to be added, and the EHC had to be modified to return to the Blaise DEP rather than launch a window to ask the follow-up questions. The transition took us no more than a couple of weeks.

This was achieved because of the loosely coupled object-oriented design of the EHC, and our focus on making the EHC flexible and customizable. Because all of the instrument content had been driven by the Blaise database and the metafiles, very minimal changes were needed to accommodate the new approach. The balance of our development time has been spent incorporating additional requirements, making enhancements to the GUI of the EHC, and testing. A number of technical problems had to be overcome.

The graphic on the next page shows the Residency topic detail questions as they appeared in the EHC used for the 2010 field test. Note that the country combo box covered some other items below it. In order to browse through the combo box, the FR had to select the item again which was always difficult. Another issue we encountered was selecting the respondent's country as part of his address. In most cases this was the United States, and that option was not available as the first option.



The graphic below is the Blaise side of the data collection for the 2011 field test. A lookup table is now used to display countries list defaulting to the United States as the first option displayed.



7.2 Copying data

One of our biggest challenges revolved around copying data, both to populate data and to populate enumerated fields. For example, in our topic section on residency in our 2010 instrument, we were copying address information to individual fields (detail questions regarding the current address), as well as copying the entire address to an enumerated list so subsequent respondents could simply choose an address from a dropdown list rather than retype all the information. In 2010, this was done using a combination of Blaise and C#. However, the dropdown list was not as user friendly as expected. As a result, FRs were instructed to handle these items with extra care (extra keystrokes) since the dropdown automatically expanded as soon as the focus was active for this question blocking some other questions on route and confusing the FRs at times.

For our 2011 instrument, we have chosen to take advantage of Manipula for these tasks - evoking events in the DEP to call Manipula programs to populate our data. Our Manipula program is required to loop through numerous arrays, unduplicating addresses to populate an enumerated answer list to mimic the dropdown list we had in C#. We had concerns that this may add considerable time whenever this was called, but we found that the program worked quickly enough where no significant delay was detected. Impressed by the speed of this, we added another Manipula program, called more frequently, to populate data that would remain the same from person to person in a household (e.g., fields based on the residence). Again, we found that this did not create a significant delay to the program, and we have extended the use of this routine to many areas of the instrument.

7.3 Look-up tables

Another new requirement was to use lookup tables working in conjunction with one another to narrow choices down. The best example of this is our state and county lookup tables. When a state is chosen from a lookup table, we create another lookup table consisting of counties (or their equivalents) from that state. Because of the familiarity with the C# code, we opted to build these in C#, although we could have achieved this programming in Manipula had we opted to do so.

7.4 Audit Trail

In the 2010 instrument, we had one audit trail file for the DEP portions of the instrument and another audit trail file for the EHC portion of the program. Because the interview only transitioned to and from the EHC a few times per interview, it was relatively easy to reconstruct the flow of the instrument even with two audit trail files. However, in the 2011 instrument there will be many more transitions between the EHC and the DEP, and analyzing two audit trail files would not be practical.

To rectify this, we are now only creating one audit trail file. In order to do this, we have switched the "CloseFile" toggle in the .aif file so that the audit trail file is opened and closed every time a line is written to it (i.e. CloseFile=1). This removed the Blaise generated lock file after every Blaise entry, meaning that there is now nothing preventing us from writing to the Blaise generated audit trail file when we are in C#. Therefore, we are still able to use the techniques we were using to write to a C# audit trail file in our 2010 instrument, but we are now writing directly to the Blaise generated audit trail file.

In the future, we are planning to use a Manipula script in conjunction with the CloseFile setting at various points during the interview, and write additional information (i.e. paradata) to the audit trail file, such as the FR identification code, respondent name, outcome codes, and so on.

8. Conclusions

In the enhancements made for the upcoming 2011 field test, we worked to help our sponsors address the biggest concerns of the 2010 field test – that the survey was too long, and that it contained too many redundant questions. We have tried to create a smarter, more responsive and agile instrument with the help of the Blaise DEP and Manipula, using the Event History Calendar as a user-friendly shell for graphically selecting chronological information.

9. Disclaimer

This document is to inform interested parties of ongoing research and to encourage discussion of Blaise instrument design issues. The views expressed are those of the authors and not necessarily those of the U.S. Census Bureau.

Security Considerations in Blaise Environments: Options and Solutions

Mike Rhoads and Ray Snowden, Westat

1. Introduction

IT Security has become an increasingly important and costly component of survey research operations. This trend will continue for the foreseeable future due to the following general factors:

- The number and variety of applications and devices that are used to support survey data collection is continuously increasing as vendors develop new tools and technologies, such as smart phones, iPads, and faster and more reliable wireless communications, which can significantly improve the efficiency or reduce the cost of data collection. Each new technology exposes additional potential security risks, which must be understood and properly mitigated.

- The Internet has become the standard mechanism for business data communication and, given its open accessibility to the public, exposes confidential data and valuable IT assets to a wide variety of continuously evolving threats.

- The U.S. government, other regulatory bodies, and clients have become increasingly aware of the importance of IT security and have promulgated a steady flow of new security related regulations, guidelines, and best practices. The passage and implementation of the Federal Information Security Management Act of 2002 (FISMA) has created significant new compliance and reporting requirements for Federal systems.

Although the technologies, risks, and regulations continue to change, many of the scenarios that make security an important business concern remain very familiar. The following are just a sample of the common concerns that these regulations and controls are intended to address:

- A field system with personally identifiable information (PII) is lost or stolen

- Unprotected systems are exploited by a new virus

- Hackers exploit a web site and delete or deface valuable data

- An interviewer copies PII data to a flash drive which he loses

- A new OS security patch is installed and brings down 400 laptops for a week

- Employees are using work systems for Internet gambling

- An interviewer is at a respondent's house at 8:00pm and forgets her password

- Authority to Operate (ATO) is denied for an important project due to missing security controls

Given the scope of IT security, the variety of threats to be guarded against, and the number and complexity of the various regulatory requirements, IT security can seem to be an overwhelming

undertaking, particularly if one assumes that the objective is the achievement of a perfectly secure system and the elimination of all risk. Although the requirements and guidelines often appear to be unnecessarily burdensome and operationally inefficient, they are intended to reinforce a systematic and comprehensive approach to the planning, implementation, and monitoring of security controls to reduce the risks to an acceptable level, to detect incidents quickly when they occur, and to minimize the associated harm.

Blaise is an important component of a survey system and, as such, must provide security-related features, support secure development and operation, and be installed and operated in a secure environment. The next two sections of this paper summarize FISMA and the Federal Desktop Core Configuration (FDCC). Next, we look at some aspects of Blaise that specifically relate to IT security. We conclude by discussing security considerations for Blaise surveys on specific platforms. Although the specific details of FISMA and FDCC will be primarily relevant to U.S. Government agencies and contractors, the former in particular spells out a number of general principles that may well be useful for planning purposes even outside the United States.

2. FISMA as a Security Framework

2.1 Overview

Although numerous security regulations and frameworks have been developed and may impose specific requirements, FISMA has created a broad framework for IT security definition and implementation used by the Federal Government. Through the work of the National Institute of Standards and Technology (NIST), specific concepts, procedures, and guidelines have been developed and are experiencing widespread use. Given the broad scope of FISMA, these concepts and practices are relevant and applicable to many IT systems, even those not under governmental oversight, and will be used in this paper as a representative approach to IT security.

FISMA is the Federal Information Security Management Act of 2002. It was passed as Title III of the E-Government Act (Public Law 107-347) in December 2002. FISMA requires each federal agency to develop, document, and implement an agency-wide program to provide information security for the information and information systems that support the operations and assets of the agency, including those provided or managed by another agency, contractor, or other source. These agency programs are required to include the following types of activities:

- Develop an inventory of all information systems used by the agency, complete a risk assessment, and classify systems in terms of the harm resulting from accidental or malicious modification, damage, or loss in the data or supporting systems.

- Implement managerial and operational procedures and technical security controls that reduce the risks to an acceptable level in a cost-effective manner while satisfying any mandated security requirements.

- Provide training to staff and contractors about the importance of security, the risks to data and systems associated with their activities, and the policies and procedures that are to be followed.

- Perform periodic monitoring and testing, no less than annually, to ensure that policies and procedures have been implemented and are executed properly.

- Implement procedures to continuously review and evaluate security practices and identify and remediate deficiencies.

- Implement policies and procedures for detecting, reporting, and responding to security incidents.

- Implement plans and procedures to ensure continuity of operations for information systems that support the operations and assets of the agency.

2.2 Security as Defined by FISMA

FISMA defines the term "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.

Confidentiality, integrity, and availability of information and systems are the three central objectives of FISMA and these are also critical issues for any survey research project.

2.3 NIST Activities in Support of FISMA

The FISMA regulation also directs the National Institute of Standards and Technology (NIST) to develop standards, guidelines, and security requirements to be used in the development and implementation of agency security plans. The work of NIST to support FISMA has been organized as the NIST FISMA Implementation Project.

In support of FISMA, NIST has developed an integrated Risk Management Framework which brings together all of the FISMA-related security standards and guidance to promote the development of comprehensive and balanced information security programs by agencies. The name Risk Management Framework is used because the key organizing principle of FISMA and the focus of the framework is to identify and understand risk as a function of two related considerations: 1) the magnitude and prevalence of the threats to confidentiality, integrity, and availability that the IT system will be exposed to; and 2) the harm resulting from a loss of confidentiality, integrity, and availability to the IT system.

The RMF consists of the following 6 steps:

1. Categorize the information system and the information processed, stored, and transmitted by that system based on an impact analysis. **NIST Publication FIPS 199** provides guidance around the categorization of information systems using an approach that assigns a security level to each system of low, moderate, or high.

2. Select an initial set of baseline security controls based on the information system's security categorization, tailoring and supplementing the baseline based on organizational assessment of risk and local conditions. **NIST Publication SP 800-53** provides a detailed list of management, operational, and technical controls that are considered requirements for systems for each of the 3 security levels.

3. Implement the security controls and document how the controls are deployed within the information system and environment of operation. The documentation that is developed in this process and generated as the system is operated are key artifacts for the remaining steps in the RMF.

4. Assess the security controls using appropriate procedures to determine the extent to which the controls are implemented correctly, operating as intended, and producing the desired outcome with respect to meeting the security requirements for the system. This is often completed by a 3rd party security organization and involves examination of records, interviews with staff, server scans, etc.

5. Authorize information system operation based upon a determination of the risk resulting from the operation of the information system and the decision that this risk is acceptable. A security authorization package consisting of various documents including the system security plan (SSP), the results of the security assessment, and a plan of action and milestones (POAM) is submitted to a designated individual at the agency. An Authority to Operate (ATO) is the official designation by the agency that the risks of operating the system are understood and acceptable and that the system may be placed into production.

6. Monitor and assess selected security controls in the information system on an ongoing basis, which includes documenting changes to the system or environment, conducting security impact analyses of the associated changes, and reporting the security state of the system to appropriate organizational officials.

NIST Publication SP 800-53 provides a catalog of several hundred management, operational, and technical security controls and indicates the applicability of each to the 3 security levels (low, moderate, and high). Although selecting, implementing, and documenting these controls seems at first to be a very daunting process, SP 800-53 does provide a structure and degree of comprehensiveness that can be very helpful in the development of a security plan, which is often characterized by the troublesome combination of a very broad scope and numerous complex details.

The security controls that are defined in SP 800-53 are organized into 17 security-related areas. These controls can be classified into the following broad groups:

- **Security policies** – formal statements published by an organization which establish the framework in which security controls are defined, implemented, and enforced. Security policies are established by the organization's management and should reflect the importance to the organization of sound security practices.

- **Human controls** – policies and procedures which describe required, acceptable, and unacceptable behavior in the secure use of systems and data. This is typically reinforced through formal training and documentation. Also included are personnel security controls, covering position categorization, screening, transfer, access agreements, and termination.

- **Physical controls** – steps that are taken to secure and protect information assets and the physical environment within which sensitive data and systems are installed and operated, including locked cabinets, secure rooms with controlled access, fire suppression, etc.

- **Technical controls** – technical features and practices that are employed to protect data and applications such as user authentication, platform policy settings, encryption technology, backups, etc. These controls usually employ features of the network, OS, application systems, and specialized COTS products.

- **Systems management** – procedures to manage the acquisition, development, deployment, and operation of new systems, and to manage changes to these systems over time using security standards and principles.

- **Auditing and monitoring** – procedures and practices employed to actively check for and log security vulnerabilities and incidents and to verify that required security controls are in effect and functioning properly.

- **Systems continuity** – procedures and resources that are defined and implemented in order to ensure that the necessary availability of critical systems and data will be maintained in the event of technical, platform, or facility failure.

3. The Federal Desktop Core Configuration (FDCC)

As part of its overall IT security policies, the U.S. Office of Management and Budget (OMB) mandated the Federal Desktop Core Configuration (FDCC), which is a security configuration designed to standardize the configuration of desktop computers used by U.S. Government agencies. The FDCC was initially announced in a memorandum (M-07-11) issued by OMB on March 22, 2007, with agencies directed to comply by February 2008. On February 28, 2008, Part 39 of the Federal Acquisition Regulation (FAR) was modified to require that information technology acquired by the government comply with relevant security configurations. A subsequent OMB memorandum issued in August of 2008 (M-08-22) provided additional information about procedures that IT providers need to follow in order to certify that their products comply with the FDCC requirements.

On May 7, 2010, the U.S. government's CIO Council announced the United States Government Configuration Baseline (USGCB). Under this initiative, security configuration baselines were released (Major Version 1.0.x.0) on September 24, 2010 for Windows 7, Windows 7 Firewall, and Internet Explorer 8. USGCB appears to be carrying out the same types of activities as were previously performed under FDCC, which it officially replaces. However, FDCC settings and requirements for Windows XP and Windows Vista will remain in effect unless and until USGCB issues its own settings for these platforms.

Since the Blaise development environment runs on Windows desktop computers, it is required to meet these guidelines. Federal agency staff needing to run the development environment on a desktop may be

required to provide assurance of FDCC compliance either as part of the procurement process, or when they request government IT staff to actually install the software.

Westat, in its role as the licensor of Blaise software in North and South America, arranged with a third-party provider of assessment and certification services to perform the necessary validation testing for Blaise 4 under Windows XP and Windows Vista. This testing confirmed that Blaise complies with all FDCC requirements. Specifically:

- The desktop components of Blaise software are fully functional and operate correctly as intended on systems using the FDCC under Windows XP and Windows Vista.

- The standard installation, operation, maintenance, update, and/or patching of the Blaise desktop components does not alter the configuration settings from the approved FDCC configuration. Blaise uses the Windows Installer Service for installation to the default "program files" directory and is able to silently install and uninstall.

- Blaise applications designed for typical end users are able to run in the standard user context without elevated system administration privileges.

More information regarding Blaise compliance is available at http://www.westat.com/westat/statistical_software/blaise/fdcc_compliance.cfm.

4. Aspects of Blaise Relating to Security

Effective IT security measures take advantage of multiple layers of security, such as platforms and infrastructure, physical facilities, and organizational policies and procedures. A Blaise application is one of these layers. The existence of these other layers means that Blaise itself does not have to provide for every aspect of security – as we shall see, many important security practices and techniques are relatively independent of the specific software application involved. However, application software must be able to be gracefully integrated within the overall project security framework.

Fortunately, Blaise is a mature and well-designed data collection product that is highly suitable for operation in a secure environment. Numerous Blaise surveys for U.S. Government clients have successfully undergone certification and accreditation (C&A) and have been authorized for operational use. As noted in the prior section, Blaise also complies with all FDCC requirements.

Blaise also fits well into the set of security controls that relates to systems management, including development practices and change management. Tools such as Microsoft Visual SourceSafe can be effectively used for version control of Blaise code files. Software testing is also a critical component of sound development practices, and this topic has been addressed in many IBUC presentations.

4.1 A Blaise Coding Technique to Maintain Confidentiality

Although most aspects of security and confidentiality for Blaise surveys are typically implemented in security layers beyond the Blaise instrument itself, there is one relatively common situation involving confidentiality that can be effectively addressed within the rules of the Blaise data model. This occurs in CAPI instruments where some sections may elicit particularly sensitive information, such as participation in antisocial or even illegal activities. Since respondents may be reluctant to admit such behavior directly

to an interviewer, a commonly-used technique in such situations is to have those sections of the interview be self-administered, with the laptop turned away from the interviewer but facing the respondent. This allows the respondent to operate the laptop and then turn it back to the interviewer after finishing the sensitive set of questions.

When this method is employed, it is important for the interviewer to be able to truthfully assure the respondent that the interviewer will not be able to look at the answers that the respondent has recorded. The Blaise developer can accomplish this by inserting a "wall" into the data model, which will prevent the interviewer from going back and seeing the respondent's answers. The Blaise Online Assistant uses the following code snippet to illustrate this method.

```
RULES
  ThankYou.KEEP
  RespondentIntro
  NEWPAGE
  IF ThankYou = EMPTY THEN
    Ticket
    SmallOffence
    MajorOffence

  ELSE
    Ticket.KEEP
    SmallOffence.KEEP
    MajorOffence.KEEP

  ENDIF
  ThankYou
```

This code ensures that once the "Thank You" item has been presented to the respondent after having answered a sequence of private or sensitive questions, those questions and the populated answers can no longer be viewed. It works as follows:

This code uses the field (ThankYou) immediately following the sensitive questions to control the flow. The first time through, when the respondent is using the laptop, this field is empty, so the sensitive questions are asked. After responding to the ThankYou item, the laptop is returned to the interviewer. If the interviewer attempts to move backwards in the instrument, the rules branch around the sensitive questions, since the ThankYou item has already been completed. The KEEP method is used with ThankYou so that the rules can access the value that has been entered, and also in the Else branch of the If-Then-Else so that the values of the sensitive fields will be stored in the final Blaise data even if the interview backs up through the instrument.

For a complete code example illustrating this technique, see the keepdemo.bla sample file, which is in the Blaise sample library under \Datamodel\Basics.

4.2 Ability to Use Relational Databases for Data Storage

One important security feature in recent versions of Blaise is Blaise Datalink, which allows Blaise to utilize Microsoft's OLE DB technology to store its data in relational databases such as Oracle and Microsoft SQL Server. Most enterprises have a variety of established security practices for database

security, such as locating the database servers in special security zones. Using Datalink to store Blaise survey data takes advantage of these existing security procedures.

5. Security Considerations for Web-based Blaise Surveys

Applications that run on the Internet can be especially vulnerable from a security standpoint. There are many network and platform features that must be implemented to securely operate a web site accessible over the public Internet. This includes an appropriate network configuration with adequate firewall protection, Windows servers that have been hardened for use as web servers, timely software patching, virus protection, etc. This section discusses some of the important security considerations for web-based surveys.

1. One of the most basic yet important decisions to be made is the format to use for storing the incoming survey data (as well as any necessary preloaded data). The ability of Blaise to store its data directly into an RDBMS is especially important in a web environment, since enterprises are likely to locate their database servers in special security zones that are not directly connected to the public Internet. Using Datalink to store the Blaise Internet survey data in an enterprise database takes advantage of all of the security procedures that have already been established for the database servers, thus avoiding the need to implement customized methods to protect native Blaise data files.

2. User authentication and authorization is another important aspect of security for Blaise Internet surveys. Authentication is particularly complex due to the number of possible techniques and tradeoffs. The Blaise Online Assistant provides an extensive discussion of security considerations for web surveys, including several possible methods of implementing user authentication.

   A difficult issue for web-based surveys specifically can be the communication of authentication credentials to the end users. In some cases, where the sample for the survey is open, users may be asked to create their own login and password at a self-registration page of the site before taking the survey. In many cases however, the sample population is restricted and identified in advance. In order to implement an authentication mechanism that is secure and maintains a strong password, it is necessary to securely communicate an account and password to each user, and have the user change their password at their first login.

   Following the standard security principle of least privilege, users and applications should be authorized, through application, database, or OS permissions, to have access only to the specific resources required to complete the survey. This will help reduce the scope of damage that may be caused due to the activities of a malicious user or malfunctioning application.

3. Web-based surveys programmed in Blaise can be accessed over an encrypted connection using Secure Sockets Layer. SSL is implemented by acquiring, installing, and configuring a server-side encryption certificate under IIS and accessing the web site from the browser using the HTTPS protocol. Once this is done, IIS and the browser establish an encrypted connection over which all data is communicated. This technique protects the confidentiality of the data between the end-user computer and the web server.

   When setting up SSL it is important to ensure that all potentially sensitive data is encrypted including user credentials, case management data, and the survey itself. SSL should be configured as the required protocol to eliminate the possibility that an end user can mistakenly access the site

without SSL in effect. Also, the SSL encryption mechanism chosen should be FIPS 140-2 compliant.

6. Security Considerations for CAPI Blaise Surveys

CAPI surveys also present a number of unique security considerations, due to the characteristics of the CAPI environment such as the use of single-user systems, disconnected and connected operations, and the need to synchronize data and software with central systems.

The following are some key security considerations and controls when using Blaise in a CAPI study:

1. **Encryption** – encryption plays a larger role in a CAPI study than for the Internet, due to the fact that sensitive survey data are stored on the field device and must be communicated back to the central system in a secure fashion. There are three different areas where encryption technology should be considered:

    - **Full disk encryption** – in 2006 the OMB mandated that sensitive data stored on mobile devices must be encrypted. The use of a full disk encryption mechanism ensures that all data stored on a device are encrypted before being written to the hard drive. The data are only decrypted when an authorized user enters valid user credentials and logs into the device. This protects instances where a laptop is lost or stolen by eliminating the possibility that the hard drive cannot be removed from the device and simply connected to another computer.

    - **File encryption** – data files sent between the central system and field devices can also be individually encrypted before being sent. This provides another level of protection beyond the encryption of the connection itself and can further protect the data as it is stored on the field device or at the central system. Often the vendor that provides the full disk encryption technology can also provide an API that can be used to programmatically encrypt and decrypt data files.

    - **Encrypted connection** – as noted earlier, when connecting to the central system to synchronize data, SSL or some other encryption mechanism can be used to encrypt all data sent between the central office and the field device. If wireless connections will be used, make sure that adequate protections are in place to protect the data during wireless transmission.

    When using full disk encryption or file-based encryption, it is very important to ensure that the encryption keys are safeguarded, as they represent the only mechanism to access the encrypted data. Often the encryption package will provide master encryption keys which can be used to recover data if the primary keys are lost. When choosing encryption solutions for any of these areas, compliance with FIPS 140-2 should be a consideration.

2. **User authentication** – as with web-based surveys, user authentication in a CAPI environment is a critical security control and presents some unique issues.

    Providing an account and password to log into the field device is one of the primary mechanisms for protecting the applications and data on the device. The entry of a valid account and password "unlocks" the full disk encryption, if it is used, and also provides access to the case management

program, instrument data, and other field applications and data (e.g., e-mail) which may be installed on the device. It is essential that the user credentials and login process are configured and managed securely including:

- The use of a strong password or pass phrase. This includes a minimum length, minimum complexity, an expiration period, and limits on password reuse.
- Obscuring the password on the screen when it is entered.
- Disabling the account after a certain number of failed attempts to login.
- Displaying a password-protected screen saver after a period of inactivity.
- Logging all failed login attempts.

One of the most important security controls with respect to authentication is user training. Users must be taught the importance of security and the need to keep the password secret. This means not writing it down where it can be easily found, not revealing it to friends or family members, not allowing respondents to see it being typed, etc.

Another issue for CAPI projects is having a procedure by which field staff can quickly have accounts and passwords reset by the home office in the event of a forgotten password. This may include connecting to a help site or the configuration of a single-use emergency account. Security should be configured to provide necessary protections without substantively interfering with the productivity of the field staff.

3. **Platform controls** – a laptop connected to the Internet for data synchronization to the home office is exposed to Internet threats from two sources: 1) other Internet users or devices that may attempt to connect to and access or damage data; and 2) web sites that field staff may voluntarily connect to that include malicious code. For this reason it is important to approach the configuration of the field device as one would approach the configuration of a web server. The following are the types of controls that should be considered:

   - Disable any services under Windows that are not absolutely necessary to running the survey applications. This is another interpretation of least privilege and will eliminate potential security vulnerabilities that are not needed.

   - Adopt the use of one of the standard Windows security templates that are available from NIST and other security organizations. These templates include default definitions for numerous obscure system settings for which the default setting is unacceptable. It is important to test the full operation of the survey software with the template to make sure that all required services and features function properly.

   - Install and configure a firewall on the field device to limit the ports over which outbound requests can be made and to block any unauthorized attempts to connect to the field device. If possible, limit the external web sites to which a field device user can connect.

   - Install and use anti-virus, anti-spyware, anti-malware tools.

   - Disable USB ports or any other external data devices that are not required by field operations.

   The list of the specific controls and protections which may be required continues to develop as new devices and new threats emerge. Suites of security tools are available from various

companies that can provide a full range of protections that are controlled and monitored through a single integrated administrative and reporting interface.

4. **Configuration management** – another unique challenge for CAPI projects is the requirement to manage numerous, sometimes hundreds or thousands, disconnected devices in the field. Given the very dynamic nature of security threats and the survey project operation it is essential that tools are in place to manage and monitor the configuration of the field platform over time.

   The following types of configuration changes are common:

   - OS patches – new security threats are identified every day and many of them are the result of some vulnerability in the built-in OS controls. Security patches to the OS are generated periodically or as needed by the OS vendor. Patches which correct serious errors should be evaluated for immediate dissemination to the field.

   - Security application signature files - anti-virus software, intrusion detection software, and other anti-malware applications use signature files and other configuration information to detect and protect against threats. Updates to configuration files for this type of software should be updated at each connection.

   - Corrections or updates to survey instruments and case management applications are sometimes necessary to correct a programming error or add some critical new functionality.

   - Updates to the platform configuration or other security controls may be required either to protect against a newly discovered vulnerability or to allow a function that had initially been restricted.

   Introducing change to devices already active in the field is itself a source of additional risk to the confidentially, integrity, and availability of the field devices and data. A mistake in a system update can expose unintended vulnerabilities, damage data or introduce new bugs, or render the devices inoperable. Changes of these types must be designed carefully and tested thoroughly to ensure that they work properly, fail gracefully, and roll back appropriately. It is also very important to keep a log of which systems have received updates to make sure that all systems are maintained at the current configuration and to help troubleshoot problems when they might arise.

7. Other Security Considerations

As we have seen, web-based and CAPI surveys involve some unique areas of vulnerability, based largely on their physical exposure in the former case and electronic exposure over the Internet in the latter. Traditional, centralized CATI surveys do not involve either of these vulnerabilities. Of course, this does not mean that security is not a concern for such studies, but only that these particular risks are not an issue.

As with other platforms, however, evolving technologies and capabilities bring additional risks into the picture. With CATI, the most significant recent development has been the increasing use of home-based interviewers. This brings back the issue of remote access, as well as raising new risks due to the less-controlled physical environment.

A number of strategies can be employed to mitigate these risks. In contrast to CAPI, there exists the option of storing all data centrally, due to the existence of a persistent Internet connection for the duration of each interview. Measures such as background checks, interviewer training, and interview monitoring to reduce the risks of inappropriate interviewer behavior can be employed. If Blaise Internet is chosen as the data collection platform, similar techniques can be adopted as for other web-based surveys. Alternatively, you can use appropriately-secured platforms such as Citrix.

An integrity consideration for even centralized CATI surveys (and CAPI surveys as well) is making sure that only one interviewer "owns" a case at any given time. The possible risks here range from respondent annoyance (if she is called a second time after already completing an interview) to data loss (if completed data are overwritten by blank data from an interviewer who was incorrectly assigned the same case).

Nor does data collection always operate on a single platform: multimode survey efforts are becoming increasingly common. As Hart and others discuss in their 2004 paper (Hart et al., 2004), such efforts involve a number of challenges for IT security, as well as for other aspects of the project.

Finally, the need for IT security does not end once an interview has been completed and transmitted to the home office. As long as survey data are being stored, processed, and analyzed, they need to be protected by a variety of human, technical, and physical controls. Encryption may be employed or even required for data storage on the home office network (Mamer, Hart, and Rozen, 2007). Nor can you assume that you are done with security once the data are no longer stored online; you may want to consider encrypting your backup media, and you certainly need to pay attention to retention periods and destruction dates.

8. Conclusion

IT Security in survey projects addresses a very broad set of concerns intended to reduce the risks to the confidentiality and integrity of survey data and the availability of survey support systems. In addition to the many actual vulnerabilities and threats that must be identified and addressed, there may also be a significant amount of supporting documentation and related activities necessary to meet the evolving regulatory requirements.

FISMA and the supporting NIST guidelines and publications are examples of a comprehensive and robust approach to IT Security management. Although FISMA is intended to regulate U.S. Government systems development, the concepts and processes address common security issues and can be used in a variety of situations.

The different survey modes and platforms share many common security issues but also present unique problems reflecting the specific configuration of devices, users, and data. Various security controls and techniques can be configured and combined to address these unique issues.

Blaise is a central IT component to many survey operations. Blaise is a mature and proven product that provides a number of security-related features and can be integrated and operated as part of a secure end-to-end solution.

9. References

Hart, Leonard, Amanda Foster-Sardenberg, and Yuki Okada. "System Implementation for a Blaise Multimode Web, CATI and Paper Survey." *Proceedings of the 9th International Blaise Users Conference.* September 2004.
< http://www.blaiseusers.org/2004/papers/25.pdf> (August 31, 2010)

Mamer, John, Leonard Hart, and Josh Rozen. "Encrypting Blaise Data on Network Servers." *Proceedings of the 11th International Blaise Users Conference.* September 2007.
< http://www.blaiseusers.org/2007/papers/A1%20-%20Blaise%20Encryption.pdf> (August 31, 2010)

Computer Assisted Recorded Interviewing (CARI): Experience Implementing the New Blaise Capability

Rick Dulaney and Wendy Hicks, Westat

I. CARI: What and Why

Computer-Assisted Recorded Interviewing refers to the collection, management, coding and other uses of sound recordings taken during data collection. Sophisticated CAPI and CATI software packages have combined with widespread availability of recording capabilities on laptop computers to produce a wealth of supporting survey data in the form of sound files. In particular, Blaise version 4.8.2 contains full support for CARI in the Blaise system.

The advent of CARI over the last several years allows field managers, methodologists, coders and others to be present at the moment of data collection in a way never before possible. Listening to the interaction between interviewer and respondent provides important new insight – into the instrument design, into training needs, and into the quality of the data collected. Listening to that interaction while seeing the data entered by the interviewer increases the value to the listener, and directly affects several survey processes:

- CARI supplements or replaces direct observation. In many CAPI surveys, a percentage of field interviews are observed by supervisory staff, normally as part of ongoing quality control efforts and possibly as mandated in contracts. Direct observation carries significant costs, and may have an observation effect on the cognitive interaction; that is, an observed interview may be a different experience for both interviewer and respondent from an unobserved interview. CARI capabilities allow observers to hear the interactions – at their own pace and with the ability to re-listen as needed – without interfering in the data collection. This becomes more effective if the recording technology includes no signal to the interviewer or respondent that the recording is active.

- CARI supplements or replaces interviewer validation. Most surveys use some form of validation to determine whether field staff have falsified interviews. In many cases this validation involves re-contacting the respondent and verifying that the interviewer was there, asking a few of the same questions again, recording the answers on a laptop, etc. CARI data allow validators to perform this function without additional burden to the respondents.

- CARI offers significant potential to assess data quality. There are many possible factors that may affect whether the data collected are of the highest possible quality. Recording the data collection allows survey designers to assess whether specific question design facilitates or hinders the collection of accurate data. Methodologists can also gain new insight into how interviewers work and how the interaction of the interviewers and the instrument affect the overall quality of the data.

- CARI provides an additional source for post-data collection processing, such as editing or coding. Often when editing inconsistent data or coding text strings back to a set of categories, the home office staff rely on text entered by the interviewer. This text may be in the form of remarks, or perhaps a response to an other-specify or similar text string. The data, however, are entered by the interviewer in the middle of an interaction that is focused on retaining respondent cooperation and moving through the questionnaire, and may not contain everything the respondent said.

Hansen et al (2005) demonstrated that text entered by interviewers during an interview is often less complete and even inaccurate as compared to the text data captured as a result of listening to audio-recordings of the survey interaction.

- CARI can assist with long open-ended responses. Many items that involve a longer answer are difficult for interviewers to record accurately while maintaining the momentum of the conversation. Having these items recorded can yield accurate and complete data while allowing the interviewer to concentrate on the quality of the interaction.

- CARI may augment or replace consent for subsequent data collection. Many organizations that require consent to release records, allow student participation, etc. will accept recorded consent. This recorded authorization may be easier, faster and more economical than other methods relying on paper forms, either mailed or collected in-person.

- CARI can also be used as a training tool. Interviewers in training can record their administration of a survey (or components of a survey), listen to the recordings and complete self-evaluations on their skills. Training staff can then use the recordings and the self-evaluation data to guide additional training and coaching.

Methodologists in particular may appreciate the value of CARI data, as these data are a rich source for understanding the specifics of data collection. The relative ease of use encourages experimentation, such as recording a large number of items or recording items only under certain circumstances. And one need not navigate the ocean of recorded data made possible by CARI technologies at one time; items can be "banked" for further analysis or comparison against future studies.

II. CARI Implementation Requirements

Over the course of several years, a set of implementation requirements has emerged for CARI:

- The technology should to the extent possible leverage existing hardware. This is made easier because virtually all laptops available in the last few years include built-in audio recording. On at least one occasion Westat has used a peripheral recording device to record both sides of a telephone conversation, but in general the recording capabilities of the laptop computer are sufficient for CARI. The emergence of CARI capabilities now influences laptop acquisition criteria and decisions.

- The recording function itself should be completely unobtrusive. Interviewers should not be "signaled" (overtly or through subtle cues) when the system is recording, and there should be zero effect on the respondent or the data collection.

- The CARI capability should be designed and implemented to provide the maximum flexibility for survey designers, reviewers, coders and other users. This includes the ability to designate certain items for recording but not other items and the ability to record items only for a certain percentage of the time they are encountered. This also includes the ability to record continuously for a period of time or for a set of items.

- CARI should work seamlessly with the data collection system, and should require minimal additional custom coding – and any custom work should be forward-compatible with future versions if possible. The system should also allow for "informed consent" in which respondents

are informed that portions of the interview may be recorded for quality control purposes, and should turn off the recording capability should the respondent wish not to be recorded.

- The CARI data should be highly usable for coders and other reviewers: they should be able to stop, start and restart the recordings; they should be able to listen to all recordings together for a single item (multiple recordings may be generated when an interviewer backs up and re-asks an item); and they should be able to see the data entered and the paradata associated with the recording. In addition reviewers might be anywhere, and should be able to review the recordings from wherever they might be.

- Because recorded data may contain virtually anything, all CARI data must be considered and treated as confidential.

Our core CARI implementation at Westat falls into three areas:

1. Recording items during the CAPI interview
2. Integration into field management systems
3. Reviewing and coding the CARI data

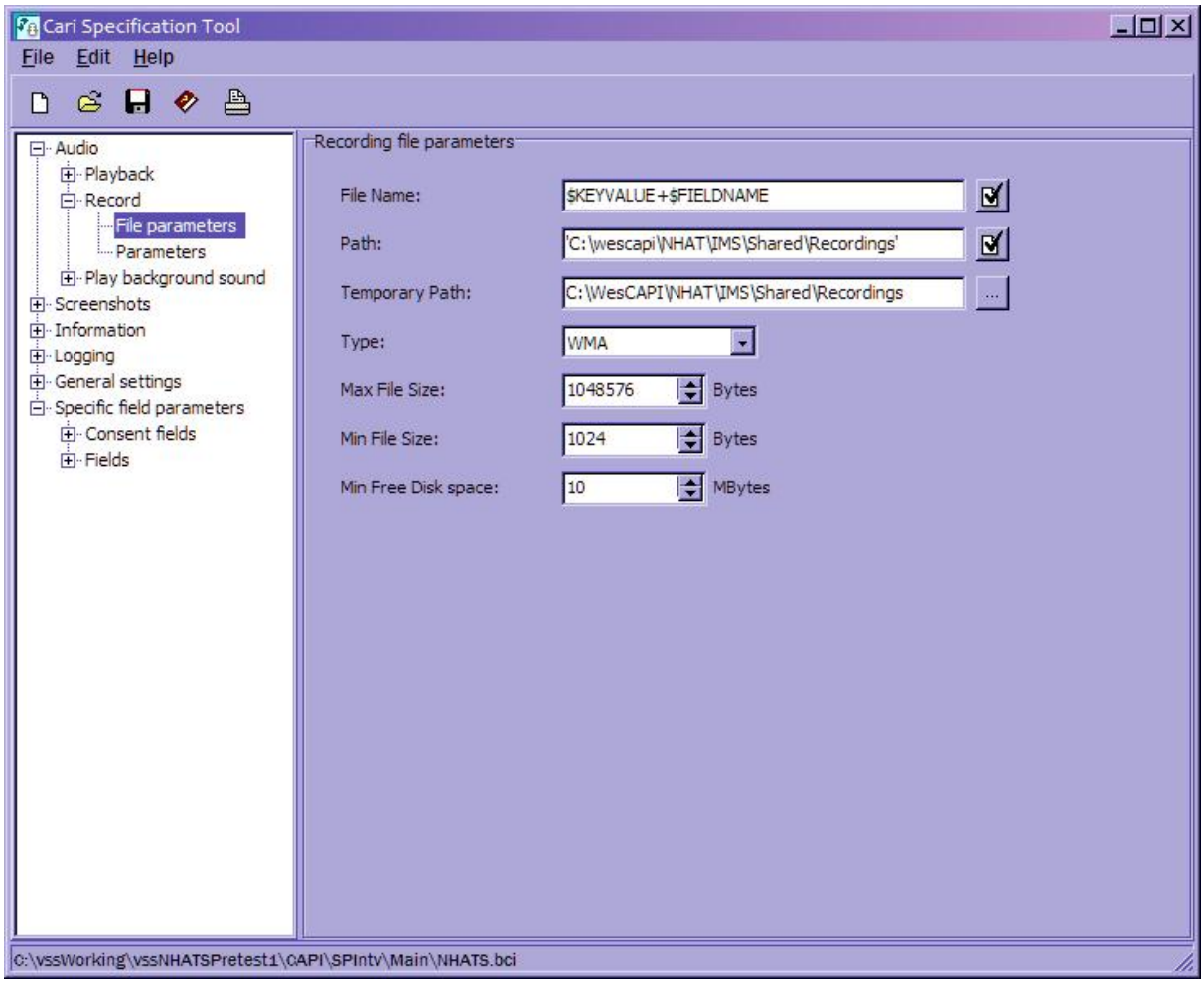
The remainder of this paper takes these up in more detail.

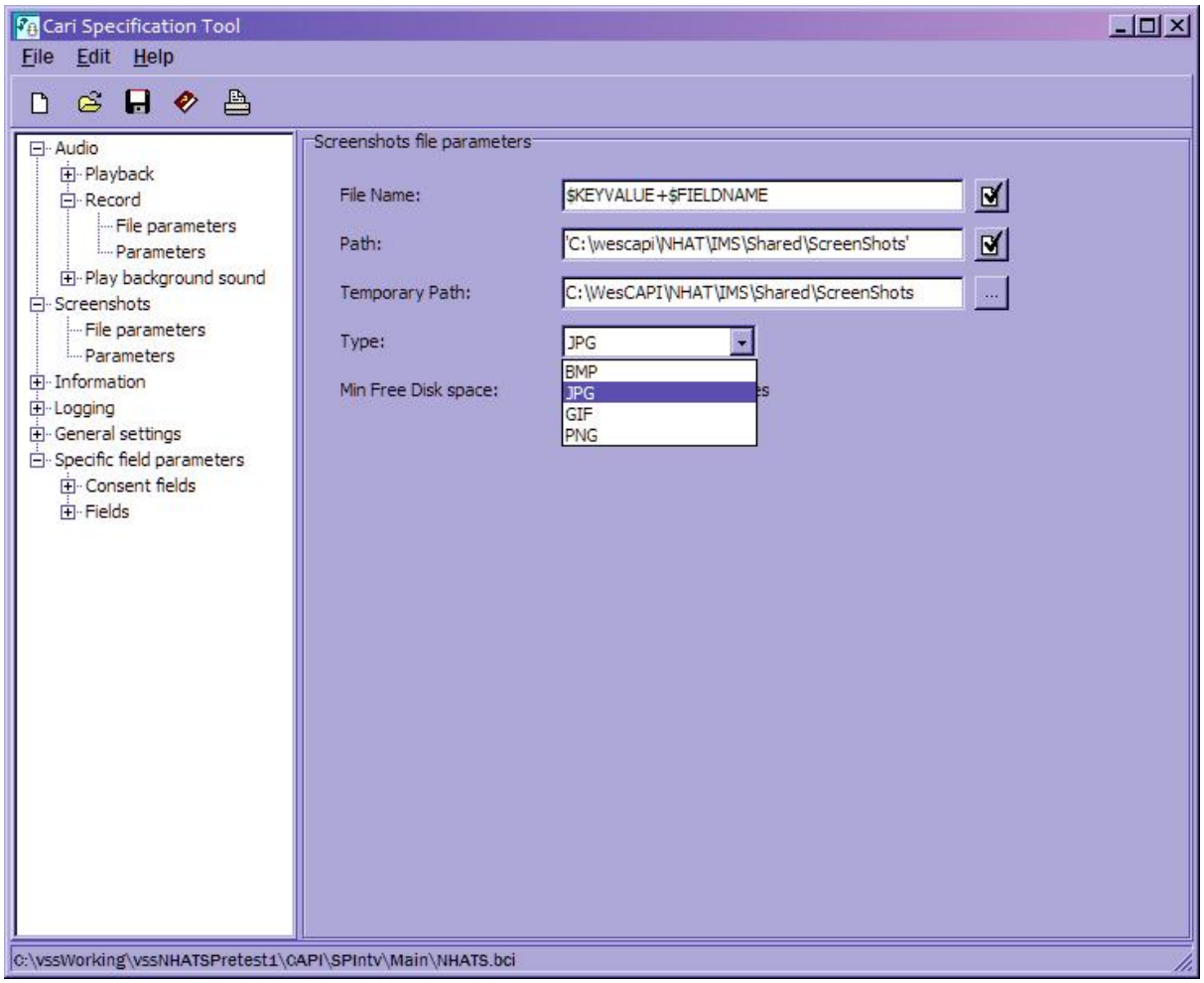
III. Recording Questionnaire Items

For some years, Westat has implemented a Westat-developed recording capability built into the Blaise product. This capability, known as WesCARI, leveraged the audit.dll shipped with Blaise. Programmers specified in an .INI file which items were to be recorded (using the full Blaise path) and a percentage between 1 and 100, where 100% meant that we always recorded the item. Since the audit.dll program always knew the current item, we evaluated that against the items in the .INI file and generated a random number to determine whether to record the item. The audit.dll program typically writes records to the audit trail log as each field is entered and exited, and so we turned the recording capability on and off at those times. The .INI file also controlled the specifics of the recording, including the format and granularity of the sound files, and the maximum length of the recording.

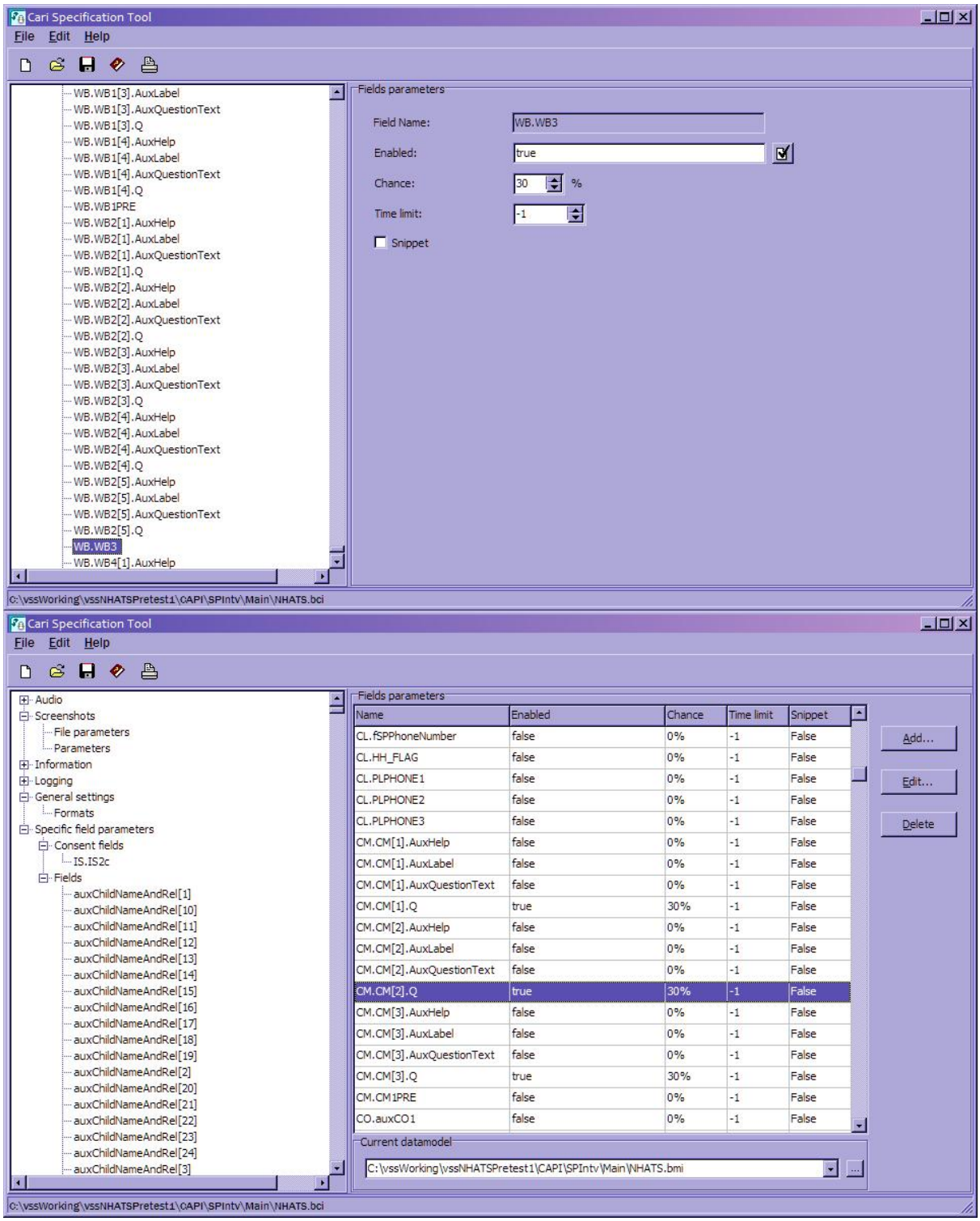
This approach yielded the sound files we wanted, but had some drawbacks. First among these was the need to re-engineer the recording capability into the audit.dll file whenever a Blaise release changed the audit.dll. Although these system changes do not occur all that often, we still needed to evaluate each release and update which versions of WesCARI were appropriate for that release. Second, our review applications could display the questionnaire specification for a particular item and the data stored for that item, but we could not easily display the item as it was displayed to the interviewer and the data as entered by the interviewer at the time of the recording. In most cases, this latter shortcoming was not a problem, but when interviewers backed up and changed the data – thus generating two or more recordings – it became more difficult to evaluate the interviewer's performance and the quality of the data. In addition, the mismatch between a questionnaire specification version of a screen hindered coders' ability to accurately assess interviewers' skill at reading scripted questions correctly since the appropriate fill logic was easily lost among all the fill alternatives.

With the advent of Blaise 4.8.2, the CARI capability is included directly in the Blaise system. From the control centre, you can invoke the CARI Specification Tool and specify sound recordings and/or screenshots:


Cari Specification Tool
File Edit Help
Audio
Playback
Record
File parameters
Parameters
Play background sound
Screenshots
Information
Logging
General settings
Specific field parameters
Consent fields
Fields
Recording file parameters
File Name:
$KEYVALUE+$FIELDNAME
Path:
'C:\wescapi\NHAT\IMS\Shared\Recordings'
Temporary Path:
C:\WesCAPI\NHAT\IMS\Shared\Recordings
Type:
WMA
Max File Size:
1048576
Bytes
Min File Size:
1024
Bytes
Min Free Disk space:
10
MBytes
C:\vssWorking\vssNHATSPretest1\CAPI\SPIntv\Main\NHATS.bci

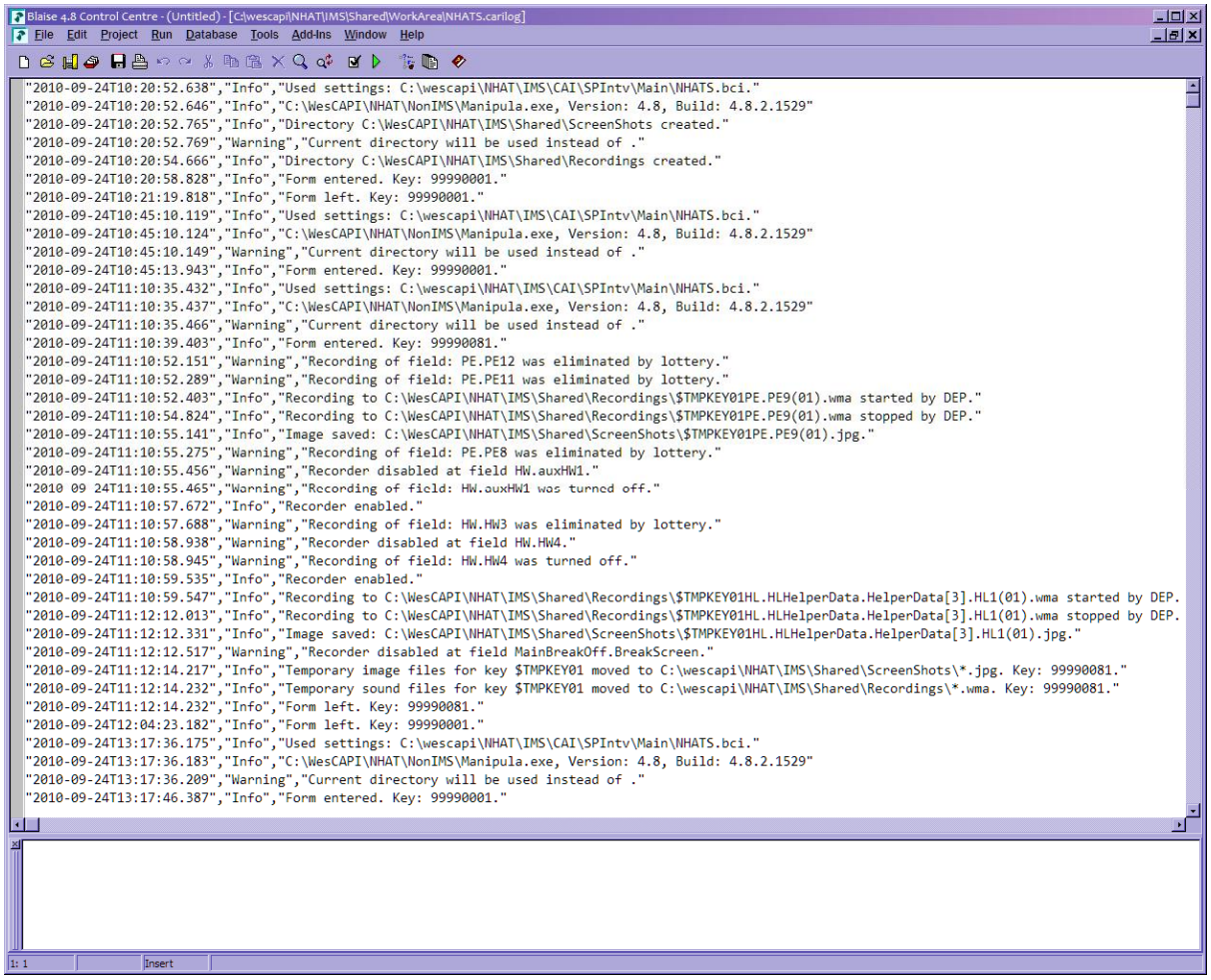


Note that you may specify the file name; in the example above, we include the case ID (KEYVALUE) and the field name. In the example below, note that you may designate the "chance" that the question will be recorded; below on the left we would like the item recorded 30% of the times it is encountered, and on the right we can review all the items and the probabilities of recording.



In addition to the probability feature, the CARI Specification Tool includes the ability to specify maximum recording length, granularity, etc. In addition, Specification Tool users can specify which field indicates that the respondent has consented to be recorded; in the event this field is set to No – whether initially or if the interviewer backs up and changes it to No – then no recordings are made.

The CARI Specification tool also allows us to designate a CARI log file that contains an audit trail of recording activities. We may specify that the CARI log contain only fatal errors, or contain additional messages useful for reconstructing what occurred during data collection. Here is an example of the most extended messages:



Finally, the ability to take screen shots showing the response and all fills is a very useful feature of the Blaise CARI capability:

Note that this screen shot – this is a file stored with the other CAPI data on the laptop – shows the data as entered by the interviewer. This is particularly useful in the event the interviewer backs up and changes the response: recordings can be keyed to the exact screen rather than to the final data values. Additionally, the actual screen shot shows only the appropriate fills for the question, exactly as displayed to the interviewer, facilitating assessment of interviewer skills.

We are using the Blaise CARI for several reasons. First, it is tightly integrated with Blaise, which leads to straightforward and robust implementation in our Blaise applications, and provides an easily-understood structure for identifying the items and percentages to record as part of the questionnaire specification process. Second, this integration eliminates the maintenance burden of having versions of the CARI software that are sensitive to the version of Blaise. We expect that CARI will be forward-compatible with subsequent versions of Blaise. And finally, the screen shot capability allows for more accurate and detailed coding.

IV. Integration into Field Systems

Sound files present several challenges to survey field management systems. First, they can contain any data at all, and therefore may contain confidential information. As such, the sound files need the highest possible level of protection within the overall data and communications flow. Second, the files need to be named and stored in a way that makes them easy to process and to identify with the Blaise survey data,

stored separately. Third, the system needs to accommodate significantly larger data files, which need to be transmitted as necessary throughout the system.

With regard to the first challenge, our response is to treat the sound files like any other case data. Westat's Baseline Field Operations System stores all survey data associated with a case in a single directory location, and keeps them all encrypted until they are needed for data collection. At survey time, when the interviewer has selected the case from a list in the management system and has started the survey, then BFOS decrypts the data files and starts the Blaise application. When the survey is finished, BFOS re-encrypts all the data files, including the sound files, and stores them in their case-specific location. The individual sound files themselves are named with the case ID, item or question number, date/time, and sequence number (if an item is recorded more than once).

The large data files generated by CAPI are a major challenge. First, the size cannot be accurately predicted, since the recording may be for just a second or two or may be minutes in length, particularly if the interviewer needs to probe or clarify a response. Second, one can estimate but not predict how many recordings there will be if the survey records only a percentage of the items. Third, many items scheduled for recording depend on prior skip patterns and therefore may not be reached at all (though the skip patterns are typically considered in setting the percentage for recording an item). Fourth, interviewers may back up an unlimited amount of times to the same item, thus generating many extra sound files.

It therefore seems prudent to plan defensively for a large amount of CARI data. On a recent small study at Westat, the Blaise data for an interview took 12-24 KB on average, while the complete case data including screen shots and sound files was in excess of 4 MB. This will have an effect on transmission times and on data storage and retrieval times. Data transmission is normally acceptable over broadband, even when several cases need to be transmitted, but if field staff are using phone lines they should transmit frequently and expect much longer transmission times than usual. Westat normally instructs field staff to transmit as soon as possible after performing any field work, but this is especially important for those staff who cannot or do not have a broadband connection and who need to use phone lines. On the laptop itself, Westat has historically not deleted completed cases after transmission until a natural break in data collection (such as the end of a round). We are now looking carefully at disk space usage and are ready to delete complete cases earlier if we suspect that the larger CARI files are having a negative effect on disk fragmentation and performance.

V. Coding Sound Files: CARICode

For some years projects at Westat have used networked applications to code CARI data. Recently, however, Westat has developed a broader capability available for distributed use across projects and known on projects as CARICode. The CARICode system allows authorized users to log in and securely perform specific coding functions using specific roles. CARICode functions include:

- Interviewer validation. Coders – generally field supervisors or staff dedicated to quality assurance – listen to recordings and fill out a coding form describing how well they heard the interviewer and respondent as well as other sounds they may have heard (or not heard) on the recording that can help provide clues as to the likelihood of falsifying data. By default, the coders listen to all the items designated for validation for a single case coding a specified set of variables describing what they heard on the recording and then complete a case summary. The summary data are then ultimately used to categorize a case as potentially falsified and needing further review. CARICode is loaded with the recordings, screen shots and relevant paradata for the cases to be validated (e.g., CAPI timing data, start time/date of the interview, previous result codes for the case, etc) and the recordings are presented on the screen with the screen shots and

paradata . The coders use all of this information – the recordings and the paradata – to assess the likelihood of falsification.

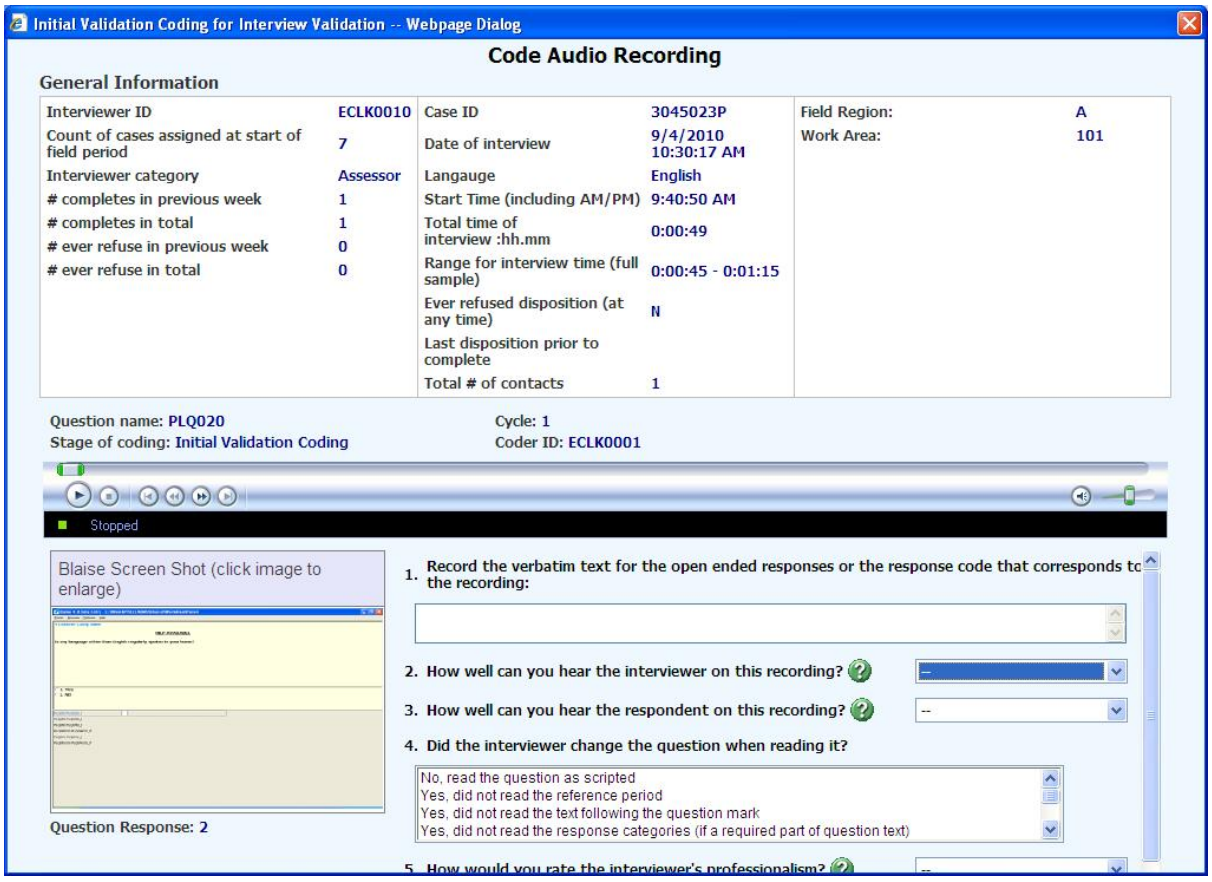
- Question assessment. Coders – normally design staff and/or methodologists – sign in and listen to specific questions and code how well they are working in the field. By default, the coders will listen to a specific question across cases rather than all the questions recorded within a case. Additionally, the system allows for a sample of recordings to be double-coded or verified, and differences adjudicated as needed.

- Interviewer performance. Coders – generally field supervisors or their managers – log in and listen to how interviewers asked specific items and how they responded to and recorded different responses. Any quality issues are addressed directly with the field interviewers as part of our ongoing feedback to their development.

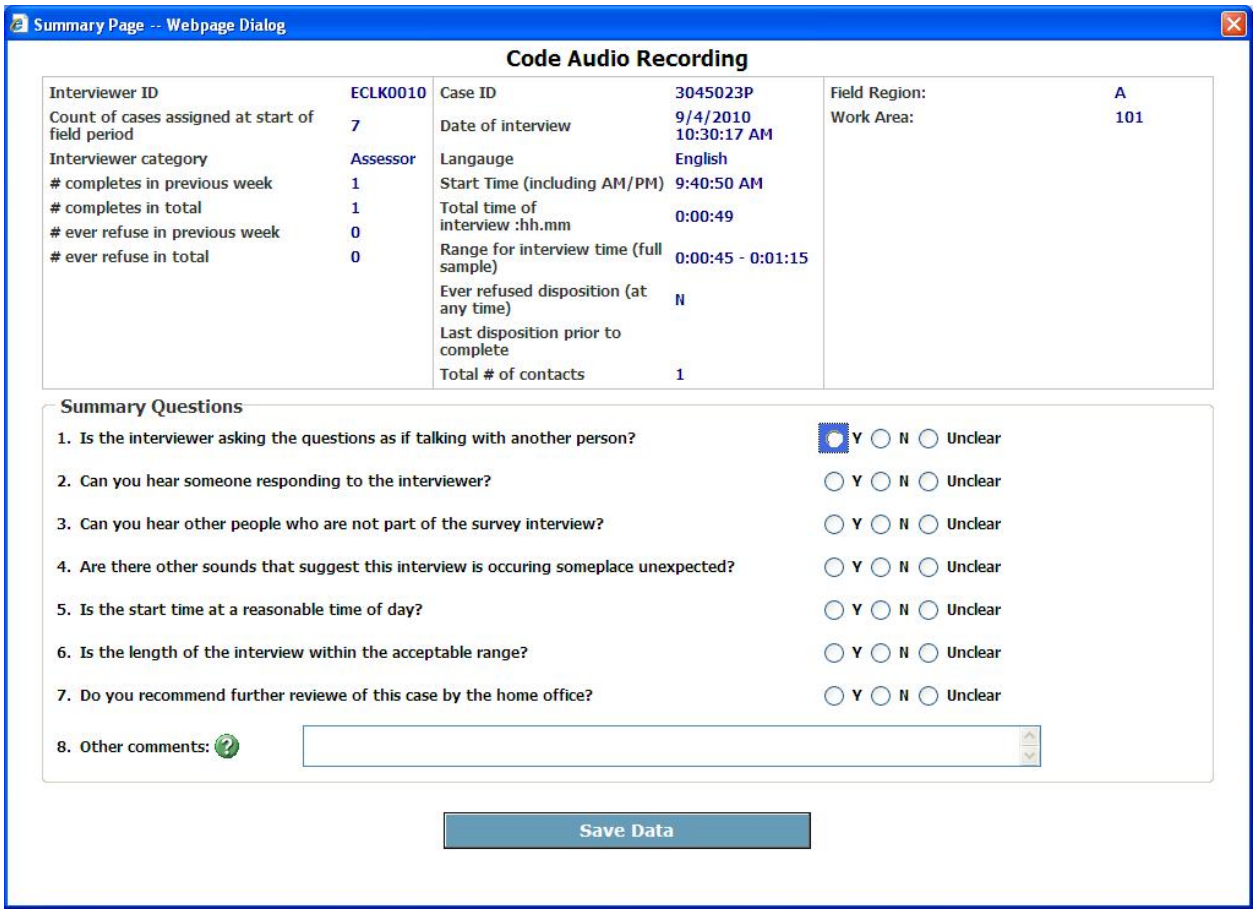
- Survey data capture. Coders – generally coding staff – log in and listen to the audio recordings of open-ended questions and capture the response verbatim, or use the recording to code the response into specified categories, such as with Industry and Occupation coding or some other classification scheme. This can be done as part of a field pretest to help develop defined response categories for full sample data collection, as part of a post-collecting editing operation, or as the actual data capture for a question.

For all of these functions, reports and other outcome materials are available for managers, methodologists and field staff as necessary.

An example of the screen used to code CARI data for interviewer validation follows. The screen is divided into four parts: The top section contains paradata from the field management system, the middle of the screen allows the coder to play the recording, the bottom left shows the Blaise screen shot (may be enlarged if necessary) and the bottom right contains the questions for the coder to answer.



The coder codes all recordings for a given case using this format, and then summarizes the recorded data for the case using the following screen:



One benefit of a distributed system is the ability to direct users who are best positioned for specific tasks to that work. CARICode is role-based, meaning that user tables are configured so that individual users may be a coder for one function, a verifier or adjudicator for another function, and have no role at all on a third function. Other configuration tables allow individual projects to designate specific CARI items and associate them with individual functions.

VI. Implementation Overview

An overview of the CARI implementation on a specific project is:

- During the questionnaire design and specification stage, items are targeted for recording by function or purpose of recording, along with the probability of recording. . For instance, a set of questions will be recorded and used for interviewer validation, another set for interviewer performance, and some for multiple purposes such as interviewer performance and question assessment. A pretest may place particular emphasis on question assessment, while another project may not use that function at all.
- These items and functions are tested during instrument development.
- Systems integration testing – i.e., testing that the instruments, field management system, and all data communication components work together – includes CARI sound files.
- CARICode items and coding/verification rates defined.
- Supporting data are defined and specified and loaded into the CARICode system. Paradata are at the heart of this specification, and may include:

    - Timings data – either the whole interview, or modules/sections within the interview
    - Interview date and start time
    - Previous result codes for the case, such as an pre-close out refusal
    - GPS coordinates
- Coding items and response schemes are configured for each coding function in CARICode.
- CARICode usernames, passwords, and access rights are defined and implemented for this project.
- Case data and sound files are loaded from the data collection environment into CARICode and tested, and coding begins.
- Coded data are extracted into various reports, including those designed to monitor and manage the coding effort, and more analytical reports geared toward data quality assessment and outcomes.

VII. Conclusion

Advances in CARI capabilities have expanded the utility of the recording data, making it easier to create the recordings and to subsequently process the recordings for many different purposes. CARI data and the improved coding capabilities significantly enhance the usefulness of the recorded data in terms of improving survey data quality, interviewer performance, and even reducing total data collection costs with the reduced need for direct field observations and validation reinterviews.

Development of an integrated CARI Interactive Data Access System for the US Census Bureau

Mai Nguyen, M. Rita Thissen, B.Christopher Siege, Sandhya Bikmal, RTI International

Abstract

In preparation for the American Community Survey Content Test, a team from the US Census Bureau and RTI International has been collaborating in the development of the CARI Interactive Data Access System, an integrated web-based system for reviewing of audio and image recordings which are made with Blaise 4.8.2 during field and telephone interviewing.

The use of computer audio-recorded interviewing (CARI) techniques is expanding and can be expected to grow rapidly with the release of CARI capabilities in Blaise software. Capture of these sound and image files is only the beginning of the process, however; to be useful, those recordings must be reviewed and analyzed in light of the study goals. CARI can have many values: confirmation of data quality and authenticity, examination of the effectiveness of question wording, study of interviewer and respondent behavior, managing performance of the interviewing staff, and other purposes. In this paper we discuss the design and implementation of the integrated CARI Interactive Data Access System with emphasis on data integration to bring audio and image recordings created from Blaise 4.8.2 during data collection into the system for quality review, authenticity evaluation and behavior coding of interview administration, respondent reactions and questionnaire design

Introduction

Blaise software offers many built-in capabilities for survey data collection, and with the release of version 4.8.2, the language now has the capability to record audio and capture snapshots of screen images during the course of an interview, storing them on the computer's local drive along with the response data. Whether used for in-person interviewing on laptops or computer-assisted telephone interviewing with voice-capture, the CARI-equipped Blaise instrument appears and behaves just as it would without audio or image capture.

By programming a survey questionnaire to take advantage of the audio and image recordings, researchers can obtain insight into the interviewing environment in detail through later review of the audio recordings and their matching screenshots. Many uses have been found for computer audio-recorded interviewing (CARI), ranging from quality control <*Thissen et al, 2010*> <*Mitchell et al, 2008*> <*Thissen and Nguyen, 2008*> <*Basson, 2005*> to coaching <*Biemer et al, 2000*> <*Watt, 2009*><*Thissen et al, 2008*> to data capture <*Edwards et al, 2010*> .

Over the past several years, the US Census Bureau has evaluated the use of CARI, with the intent of using it for production survey data collection after establishing its feasibility <*Biemer et al, 2000*><*Wrenn-Yorker and Thissen, 2005*>, acceptance by field staff <*Arceneux, 2007*> and cost-effectiveness <*Biemer et al, 2002*>. In 2008, a prototype CARI System was created at the Census Bureau <*Thissen et al, 2008*>. Stemming from that effort and other methodological and technical studies, a decision was made to develop a full-fledged system for use on the upcoming American Community Survey Content Test (ACS Content Test).

The ACS Content Test is programmed in Blaise 4.8.2, collecting audio and screen images as described in a separate paper at this conference, *Implementing Computer-Audio Recorded Interviewing (CARI) Using Blaise 4.8.2, William E. Dyer, Jr. and Malcolm Robert Wallace, U.S. Census Bureau*. To effectively use CARI recordings, the organization requires a secure system for receipt of the incoming audio and image files, an efficient interface for playback and coding their contents, and especially for a large organization

such as the Census Bureau, a facile mechanism for managing operations which will ultimately need to handle many thousands of cases passing through it on a flow basis.

The CARI Interactive Data Access System (CARI System) was collaboratively developed by RTI International and the Census Bureau for this purpose. The system was designed to reside centrally at the Suitland, MD, headquarters of the Census Bureau while being accessible to Census Bureau telephone centers in Hagerstown, MD, Tucson, AZ, and Jeffersonville, IN, as well as to the twelve regional offices across the country. To enable such broad access without distributed storage, the system provides a web based interface, requiring only a standard browser and network access credentials for usage.

The ACS Content Test is relying on CARI recordings from the Blaise instrument to aid with decisions on questionnaire design through behavior coding of the captured audio and image files. Later surveys expect to use the system for quality assurance, management and coaching of field staff, and potentially other purposes. In the pages that follow the CARI System's design, functionality, usage and implementation details are presented.

Functionality

The CARI System presents a web based interface currently consisting of 4 major components. Plans exist to expand the system to enable additional functionality. Two of the components provide direct access to the CARI recordings from Blaise, while two others support daily operations.

- **Behavior Coding Component (BCC).** The BCC is designed to allow researchers at the US Census Bureau to capture the interactions between interviewer and respondent by listening to the CARI audio files and viewing images captured during data collection. Coding interviewers' behaviors as part of the survey interaction not only allows research managers to monitor whether interviewers consistently adhere to the required (standardized) interviewing practices, but it also alerts them about possible problems with the questionnaire. For example, deviations from the interviewing script may reflect weaknesses in the questionnaire design or skip logic. On the other hand, the respondents' behaviors may also reveal if respondents adequately understand and respond to a particular question. The request for explanation, providing inadequate response, or other non-scripted discussions may indicate a weakness in the comprehensibility of the respondent task or survey question.

- **Quality Assurance Component (QAC).** Quality assurance activities ensure that the data collected in a survey have as little error as possible. Survey error can arise from numerous sources, but generally fall into two main categories: authenticity problems and errors introduced by individual differences in presentation of the questionnaire by survey interviewers.

  The first error source, relating to authenticity, arises when interviewers do not collect data directly from a respondent. This may occur in an innocent way, as in the case when interviewers feel they already know the respondent's answer and do not wish to burden them by asking the question, or it may arise as a result of malfeasance, in which the interviewer does not collect data at all or fabricates it. In any of these cases, the data cannot be considered authentic.

  The second source of error, resulting from differences in presentation of questions to the respondent, can be seen as a lack of precision. Most survey data collection is predicated on the notion of consistent delivery of the questionnaire to minimize individual differences. Though computerization forces some degree of uniformity, the interviewer may still cause variation through failure to follow protocol (such as non-verbatim reading), emotional loading (such as through tone of voice), biasing the response by suggestion or leading comments, or other

violations of best interviewing practices. These faults weaken the reliability and validity of the data set as a whole, though individual data points may be correct.

The QAC is designed to allow monitors to evaluate the quality of collected data by listening to the audio files and viewing images captured during data collection.

- **Automatic Assessment Tool for User Feedback (AAT).** The AAT module allows the Census Bureau staff to collect user feedback to evaluate ease of use and effectiveness of the CARI system from the user's perspective. RTI and Census Bureau staff will use the feedback to identify future improvements to the CARI system.

- **eTraining.** The eTraining module provides integrated, online training materials to introduce new coders to the system and to train them on how to manage their workload and to conduct the behavior coding and/or quality assurance coding properly.

Coding Workflow

The behavior and quality assurance coding processes are managed by Census Bureau staff at various divisions following specific workflows associated with each component. In general, the overall division of labor among the Census Bureau divisions is:

- Site supervisors and staff at the 3 telephone sites will primarily use the website to conduct the behavior and quality assurance coding activities.
- The survey configuration, case assignment to sites, workload monitor, and data extraction for analysis are done by research staff at the HQ.
- Website administration and its infrastructure are maintained and supported by Census Bureau staff.

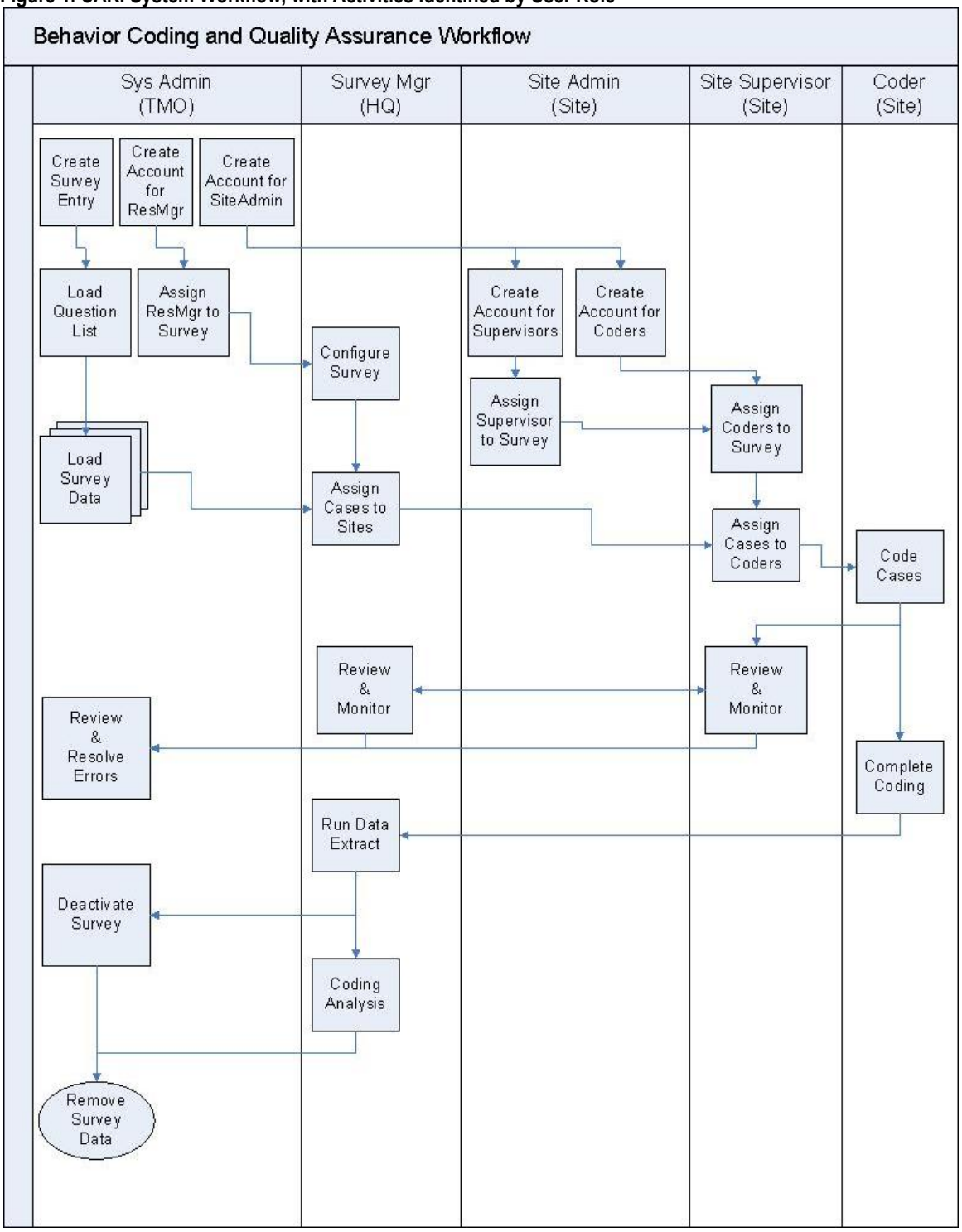
The CARI System is designed to support this division of labor through the use of user roles as shown in **Figure 1**. The coding workflow includes all tasks from the creation of a new survey entry through coding completion and finally survey data removal, in chronological order, identifying the responsible party at each step. The major tasks in this work flow are:

- Create new survey entry and user accounts
- Configure and assign staff to the survey
- Load data into the system
- Assign cases to sites and coders
- Conduct coding
- Monitor coding progress
- Data extraction and close-out

Each user role has different responsibilities within the website, providing a clear division of labor from setup through closeout. The original setup for any survey takes several steps, after which operations become more routine. To begin with, survey entries are created in the system by the System Administrator. Subsequently, a list of the questionnaire items which were recorded is uploaded through an interface provided as part of the system. Once a survey entry is ready, the case data can be loaded into the system, but it cannot be coded yet. The System Administrator at this time may create accounts for users, such as a Survey Manager and Site Administrator, giving them access to the newly created survey entry. Responsibility then shifts to the Survey Manager, who configures the coding interface with survey-specific header items, coding categories, coding groups and code definitions. The Research Manager also designates groups of cases for any sites at which coders will work, after which the Site Administrators can assign those cases to specific Coders, who listen to audio, view screen images, select appropriate codes and write notes with their observations. The majority of work is done by the Coders, who may review

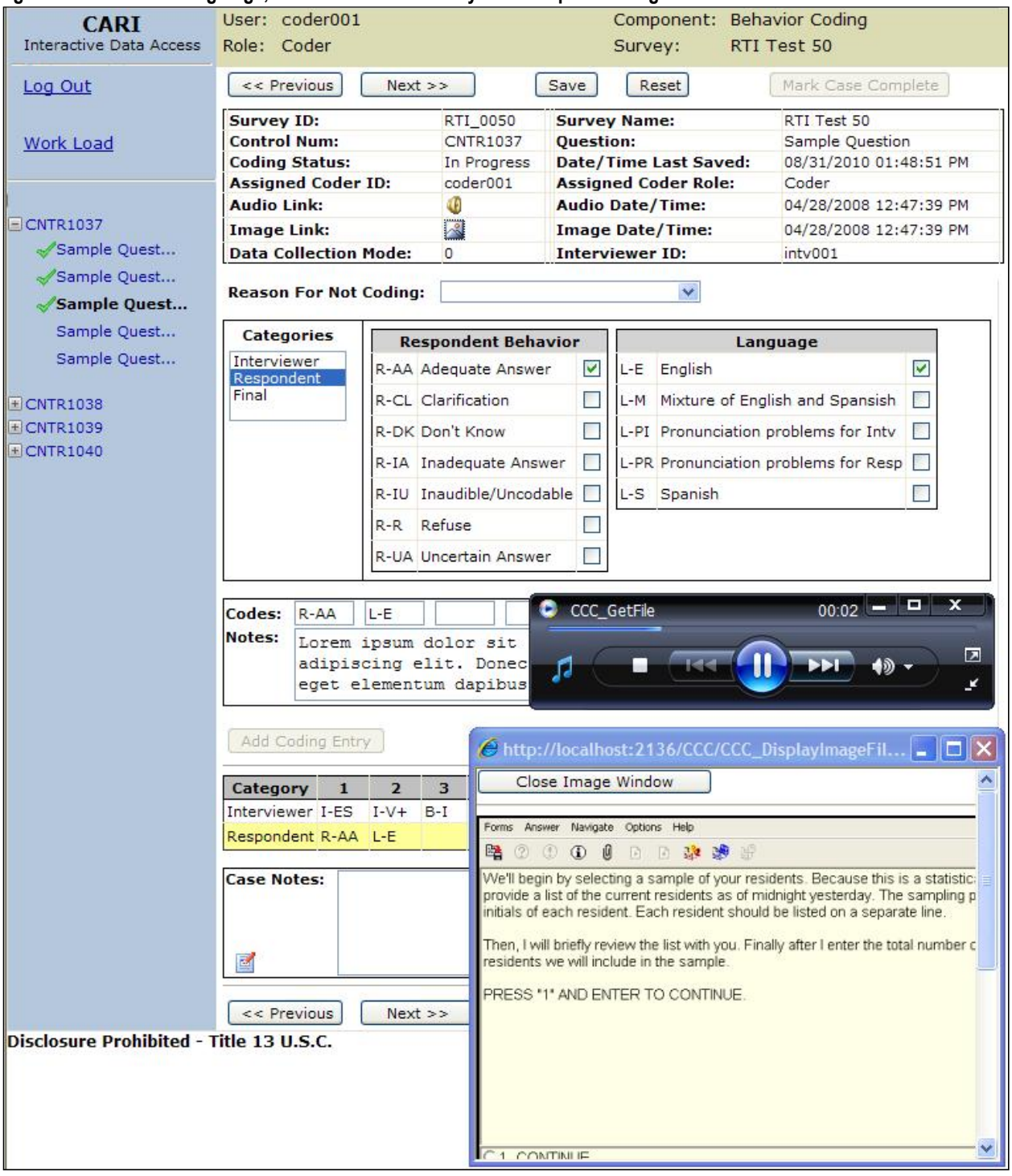
thousands of cases for a single survey, and Supervisors, who oversee the work of the Coders. When all coding is complete, the Research Manager and System Administrator again take responsibility, determining when to close out the survey entry. These steps are very similar for both behavior coding and for quality assurance coding, providing a workflow which is fully managed through the CARI website interface.

Figure 1. CARI System Workflow, with Activities Identified by User Role



The Behavior Coding page, together with the Window Media Player for playing audio file and the captured images, is shown in **Figure 2** below. Through this page, the coder can access the audio recording by clicking on the Audio link and view the image by clicking on the Image link. The page also provides the capability for flagging the case if further review is needed. Further, there is provision for additional notes on the case or for each utterance to be captured by the coders.

Figure 2. Behavior Coding Page, with Window Media Player and Captured Image

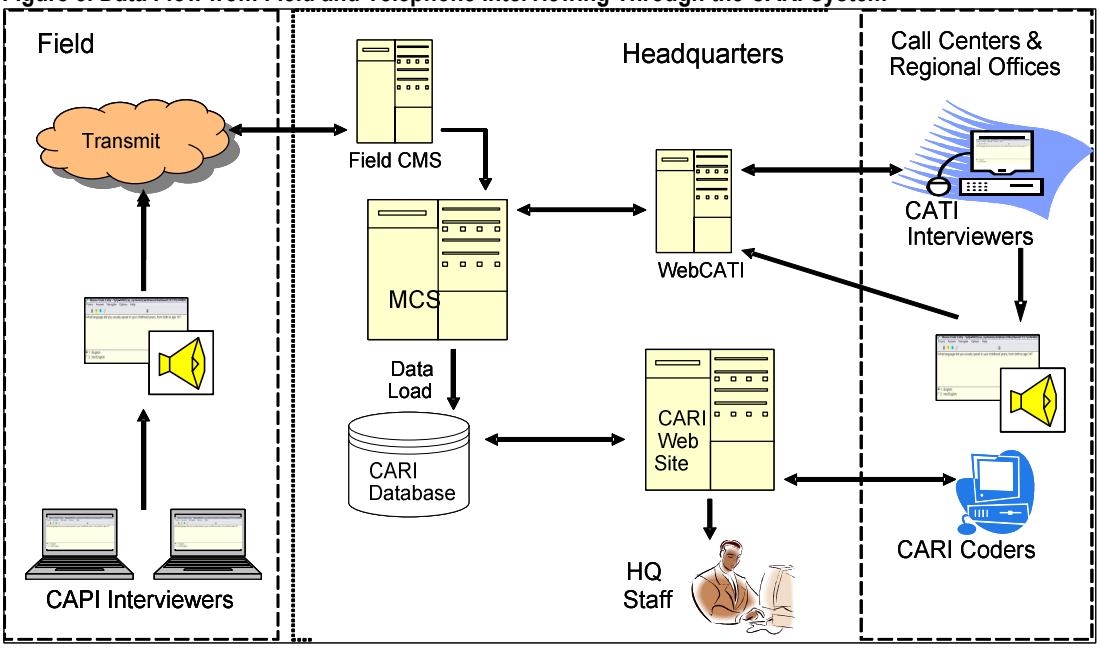


Data Flow

The purpose of the CARI System is to make Blaise CARI files available for study by staff who are located around the country, potentially quite distant from the interviewing sites or even from the Census Bureau's campus.

Response data files, including CARI files, originate at the point of interviewing, such as on the laptop of a field interviewer or the desktop of a telephone interviewer. Regardless of where the data start, the files are transmitted to a centrally located master control system (MCS) for loading into the CARI System. A simplified diagram of the overall data flow is shown in **Figure 3**. CARI coders may be located at any of the Census Bureau's locations, depending on the needs of the survey and the type of analysis for which the CARI files are being coded. Ultimately, the coded results become available to headquarters (HQ) staff, both as summary statistics and as exported datasets.

Figure 3. Data Flow from Field and Telephone Interviewing Through the CARI System



The CARI System utilizes an Oracle database as its data repository, which users of the CARI System access through the CARI website. CARI audio files and screen images, selected interview data and paradata flow into the database from the MCS through a pre-defined programmatic data-loading interface. The database also accumulates administrative data such as user accounts, roles, survey configuration parameters, code definitions and other operational information.

Once loaded, case information and CARI recordings become available to coding staff through the website. Codes are assigned, comments written and operational flags are set by the coding and supervisory staff through the web pages, which validate and store this data alongside the original information. CARI System reports and data review pages provide details and summaries to interactive users, and an extraction function allows authorized research managers to export datasets to secure locations for further analysis. Upon completion of the work, when there is no further need for storage, system tools enable removal of the unnecessary data.

System Design Overview

The CARI System has been developed using the Microsoft ASP.NET Framework 3.5 with an Oracle database. The Visual Basic.NET programming language is used for server-side development tasks in conjunction with the JavaScript language for handling client-side activities. AJAX technology is used in highly interactive pages to provide better user experience.

The Microsoft ASP.NET Framework provides a modern platform for developing enterprise-scale web applications. It provides the following development benefits:

- **Improve productivity** by using the form-based, drag-and-drop development model familiar to many developers. Microsoft Visual Studio 2008, being one of the best development tools, provides a further boost to productivity.
- **Leverage large infrastructure and prebuilt class libraries** available via the Microsoft .NET 3.5 framework, including ADO.NET for data access, server-side web UI controls, AJAX programming support, etc.
- **Support for enterprise relational database management system (RDBMS)**, including Microsoft SQL Server, Oracle database and IBM DB2.
- **Ease of deployment**, especially to Windows servers running the IIS web server.

The CARI System is hosted on the Census Bureau's Windows server using the Microsoft IIS web server. It is accessible within the Census Bureau secured intranet network. It has achieved the security certification and accreditation specified by NIST SP 800-37. The CARI System website is designed to be viewed with the Microsoft Internet Explorer (IE) web browser version 6.x or later.

In addition to network security, the CARI System also employs a number of security measures to control user access and protect its data:

- SSL protocol implementation.
- Integrated LDAP user authentication using the Census Bureau's eDirectory.
- Role-based authorization to control user access to both web pages and data within the system.
- Protections against SQL injection and cross site scripting attacks using best practices.
- Validation of all data input
- Use of parameterized queries and stored procedures for all database access
- Encryption of configuration data

The CARI System controls data access using two types of roles, system roles and survey-specific roles. System roles include:

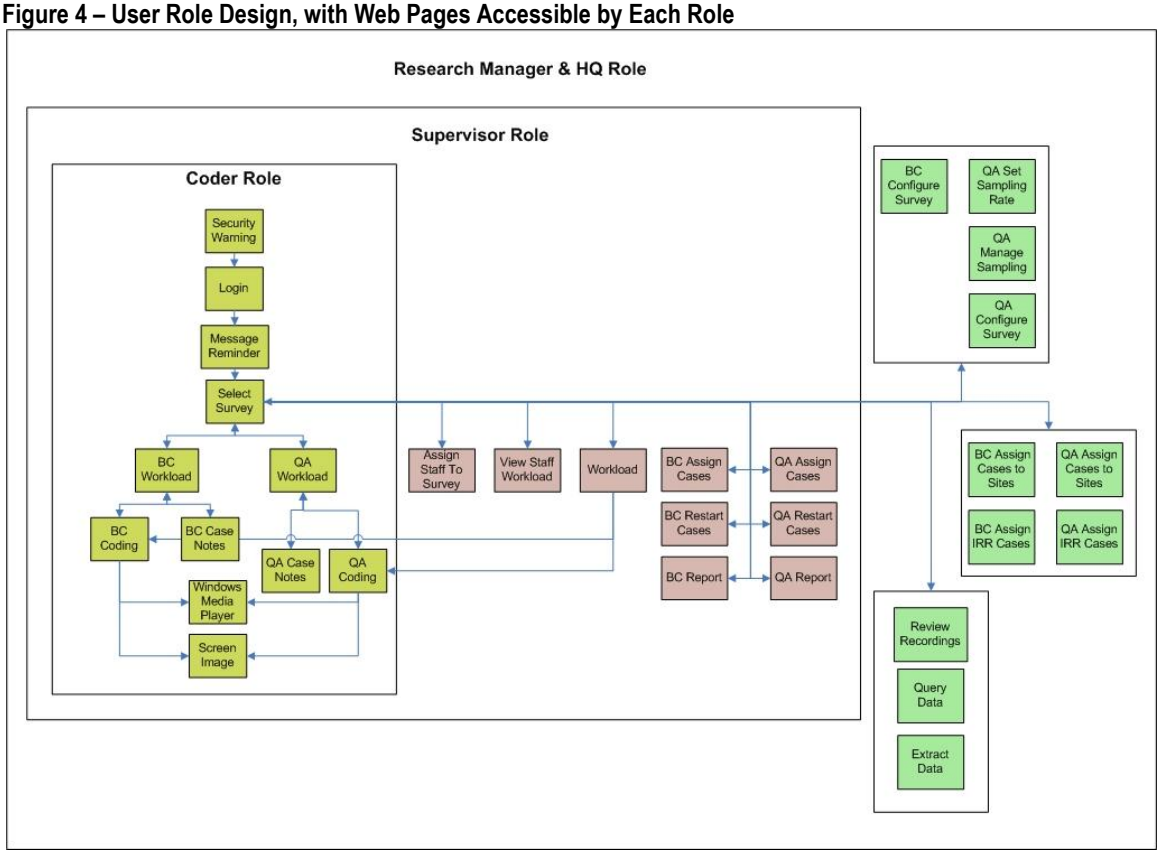
- **System Administrator**. Members of this role will be able to access all web pages and view and modify all data for all surveys.
- **Security Administrator**. Members of this role will be able to access select web pages for security auditing purpose only. They will not be able to see any data within the system.
- **Site Administrator**. Members of this role will have access to all web pages, but can only view and modify user accounts for their site (user accounts) and cases assigned to their site.

Survey-specific roles include:

- **Coder**. Members of this role can only access coding-related web pages and work on cases assigned to them. They can see neither cases from other users nor cases from surveys they're not assigned to.
- **Supervisor**. Members of this role can access coding-related web pages, workload management pages and reports, but they can only view and modify cases assigned to their site.
- **Research manager**. Members of this role not only can access all pages as a supervisor can but also can view and modify cases across sites for any survey assigned to them. They are able to access pages to configure these surveys and set up the QA sampling rate parameters.
- **HQ**. Members of this role have the same access level as research managers with the exception that they cannot modify any data.

This role-based design can be seen in **Figure 4** where each colored box represents a web page accessible by the specified role.

Figure 4 – User Role Design, with Web Pages Accessible by Each Role

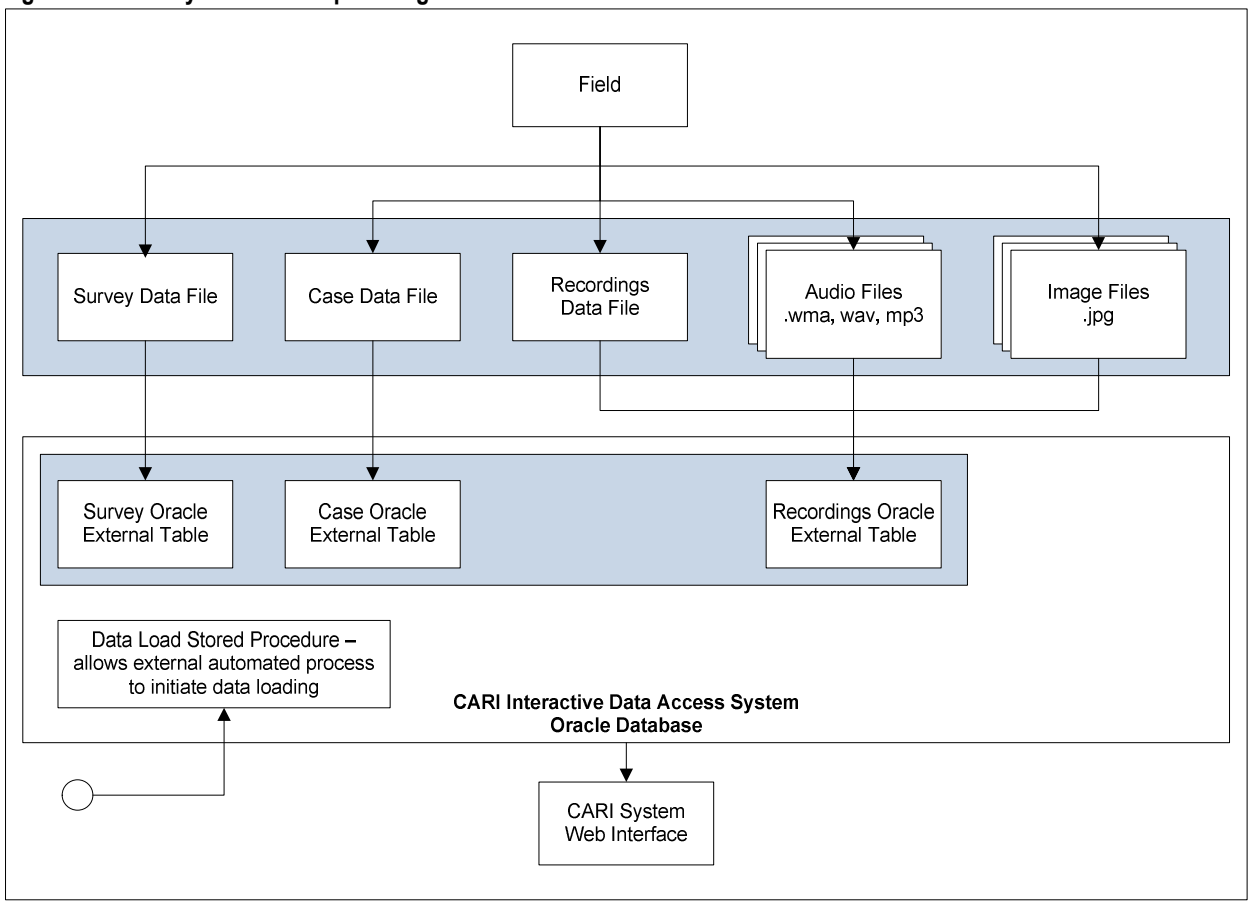


System Integration

The CARI System's primary point of integration is with the Census Bureau's master control system (MCS) through a data loading process. This process transfers selected paradata along with audio and images files from the interview case records and is capable of quickly loading thousands of recordings into the CARI System database. Within the CARI System web interface an initial survey configuration is defined and the instrument questions are imported. Once complete the CARI System can support an automated batch process of loading the survey case data either as a single complete set or on a continual flow basis.

Integration between the systems is facilitated through the use of Oracle external tables. These external tables allow flat files to be exposed within the Oracle database and provide a convenient way to process external data for importing. They provide an abstracted interface between the systems and allow details of the exporting process from the MCS to be decoupled from the importing process of the CARI System. The definition of the flat files provides an interface contract between the systems as illustrated in **Figure 5**.

Figure 5 – CARI System Data Import Diagram



Once the data load process is complete the user can then access interview cases within the web based interface for either Behavior or Quality Assurance Coding. Within these primary areas of functionality, the user can access case and question level attributes while reviewing both the Blaise screen capture and audio recordings.

After case and question level review has been completed within the CARI System it is possible to perform a number of defined data extracts. Associated with most of the data extracts is a SAS program to allow for external analysis.

Conclusion

The new Blaise functionality of audio and image recording opens a wide range of potential opportunities for survey research and operations. As mentioned above, the technology enables behavior coding, quality management and coaching activities. Additional possibilities present themselves:

- Conducting "cognitive interviews" through silent observation of production interviewing, by use of CARI files.
- Data collection, for open-ended questions or for semi-structured (conversational) interviewing
- Enhanced training for interviewing staff, allowing them to practice interviewing and review their own performance

With further technological advances, speech analytics and transcription may become feasible for post-processing the recorded audio, and pen-based computers might be able to record signatures as part of the

screen image collected for consent questions. These and other activities will make good use of Blaise's newly introduced capabilities.

References

Arceneaux, T. "Evaluating the Computer Audio-Recorded Interviewing (CARI) Household Wellness Study (HWS) Field Test." 2007 Proceedings of the American Statistical Association. Statistical Computing Section [CD-ROM]. Alexandria, VA: American Statistical Association, 2007. 2811-2818

Basson, Danna. 2005. "The Effects of Digital Recording of Telephone Interviews on Survey Data Quality." Pp. 3778-85 in the American Statistical Association 2005 Proceedings of the Section on Survey Research Methods. Washington, DC: American Statistical Association

Biemer, P.P., D. Hergert, J. Morton and W. G. Willis (2000), "The Feasibility of Monitoring Field Interview Performance Using Computer Audio Recorded Interviewing (CARI)", Proceedings of the American Statistical Association's Section on Survey Research Methods, pp. 1068-1073

Biemer, P.P., F. Mierzwa, M.R.Thissen (2002), "Comparison of the Steady-State Costs for CARI vs. Reinterview Approaches for Interview Verification", RTI Technical Report to the US Census Bureau

Edwards, Brad, W. Hicks and Michelle Carlson (2010, June). Old Dogs New Tricks: CARI and CASI Innovations to Reduce Measurement Error and Nonresponse, Presented at the International Total Survey Error Workshop, Stowe, VT.

Mitchell, S.B., K.M. Fahrney, M.M. Strobl, M.R. Thissen, M.T. Nguyen, B.S. Bibb, and W.I. Stephenson (2008). "Using Computer Audio-Recorded Interviewing to Assess Interviewer Coding Error," Presented at the 63rd Conference of the American Association for Public Opinion Research (AAPOR). New Orleans, LA USA

Thissen, M.R., Fisher, C., Barber. L., Sattaluri, S. (2008). "Computer Audio-Recorded Interviewing (CARI), A Tool for Monitoring Field Interviewers and Improving Field Data Collection", Proceedings of the International Methodology Symposium 2008, Statistics Canada, Gatineau, Canada

Thissen, M.R., & Nguyen, M.T. (2008, June). Using Audio Recording to Detect Data Collection Error. Presented at The 2nd International Total Survey Error Workshop, Research Triangle Park, NC.

Thissen, M.R., Thorpe, S., Peytcheva, E., Barnes, C., Park, H., & Fisher, C. (2010, June). Improving Data Quality Through Audio and Image Recording. Poster presented at International Total Survey Error Workshop, Stowe, VT

Wrenn-Yorker, C and Thissen, M.R. (2005), "Computer Audio Recorded Interviewing (CARI) Technology", Presented at the Federal Computer Assisted Survey Information Collection (FedCASIC) Conference.

Watt, Barbara (2009, May), "Accountability for Data Quality Excellence", Presented at the International Field Directors and Technologies Conference, Delray Beach, FL

Acknowledgements

We would like to thank the U.S. Census Bureau for funding the development of the CARI System. We would also like to thank the Census Bureau staff for their contribution and collaborative effort in this development, as well as a number of RTI staff including Carl Fisher, Neelima Kunta, Brett Anderson, Erica Saleska and Emily McFarlane Geisen. The views expressed in this paper are those of the authors and not necessarily those of the U.S. Census Bureau.

Implementing Computer-Audio Recorded Interviewing (CARI) Using Blaise 4.8.2

William E. Dyer, Jr. and Malcolm Robert Wallace, U.S. Census Bureau

1. Introduction

Our mission was to integrate Computer Audio Recorded Interviewing (CARI) with the American Community Survey (ACS) Content Test instrument for use by a Behavior Coding and Quality Assurance system being developed for the U.S. Census Bureau by Research Triangle Institute (RTI) International. Specifically, we needed to create sound and image files for the CARI questions being used for this new system.

Circumstances dictated that we implement the recording component rapidly so that we might provide input to this new system as it was being designed, developed, and tested. In the fall of 2009 we were tasked with creating what we now refer to as a "homegrown CARI." Aside from an abbreviated schedule we were faced with sponsor requirements for CARI we considered diverse and complex.

As requirements started to unfold and the number of questions the sponsors wanted to record increased, the complexities with coding our homegrown version also grew dramatically. We were pleased when we were able to capture a screen image along with a recording for a CARI question the first time the field was visited. However, the code to implement this was bulky and tricky and our homegrown version was not meeting the needs of the system or the sponsors. At this point we decided to investigate using the Beta version of Blaise 4.8.2 CARI. Once we were able to figure out how to use the CARI we quickly realized that Blaise 4.8.2 was what we needed to move forward. While using the Beta version of Blaise CARI we were able to identify some issues that Statistics Netherlands quickly addressed for us. We were also able to identify and request some additional CARI functionality that was very useful for us and hopefully will be useful for others. This paper will discuss how we implemented CARI into our Blaise ACS Content Test instrument using Blaise 4.8.2, results of our testing, and how Blaise 4.8.2 CARI addressed our CARI needs.

2. CARI Requirements

As the project started underway, the requirements kept increasing. The sponsors required a significant amount of detail (e.g., recording individual questions, series of questions, bounding questions, irregular navigation, multi-person households, top quality recordings, CARI transparent to the interviewer, sampling, screen shots, and more). Along with the requirements from the sponsors we needed to define and determine the requirements for the system. How do we implement CARI consent? What should the maximum recording length be? How do we transmit the files? How big will these files be? What settings do we need to use? What information do the back-end systems need?

For discussion purposes of this paper we will limit "CARI Requirements" to those requested for the survey instrument. Here is a summary of some of the requirements we were working with:

- Record up to 238 unique questions (18 household level and 44 (times 5) person level)
- Only record the first 5 people (over 12 years old) in roster
- Create two different content test paths – with some containing unique CARI questions and some CARI questions overlapping each path

- Use a CARI Sample flag to control the CARI recordings for 5 different paths. Different questions are recorded depending on the value of CARI sample flag (0-4)
- In certain situations record all CARI eligible questions (ignore the CARI sample flag)
- Build in a system "emergency exit" in case CARI bogs down the control systems. That is, allow interviews to be conducted without creating any sound/image files by using a system override CARI flag
- Expect to always have a 1 to 1 relationship between sound and image files – each sound file should have a corresponding image file
- Image/sound files should be created when backing up and going forward, however only save the ones that have a discussion (something of substance)
- Stop recording after 60 seconds (this was more of a system requirement than a sponsor one)
- Consent could only be recorded if the respondent agreed to be recorded.
- If interviewer changes the respondent then the instrument must "automatically" re-ask the consent question (instrument must force this)
- CARI must integrate easily into our existing CAI system
- CARI must work for CAPI and CATI

These requirements were fairly significant and did not include much of the behind the scenes type of requirements that we needed to research and define ourselves. We had major concerns regarding file sizes and potential number of files. Requirements also had to be defined for the sound/image file naming and what was needed to help integrate this data into the CARI Behavior Coding system being developed simultaneously by RTI International. The Behavior Coding and Quality Assurance system is described in a separate paper at the 2010 IBUC conference, *Development of an integrated CARI Interactive Data Access System for the U.S. Census Bureau, Mai Nguyen, Rita Thissen, B. Christopher Siege, Sandhya Bikmal, RTI International*.

3. Homegrown Version of CARI

It was pretty clear from the start that we would not be able to program all of this - in the time we had, with limited staff - and still create a reliable, easy to maintain product. We lobbied for the sponsors to record as small a number of questions as possible, recording one time the first time they were asked. We suggested that, if we were lucky, we might be able to get a screen capture of the question text and answer as we left the field.

In a previous field test, we implemented a CARI instrument that recorded snippets from a Health Wellness Study so CARI concepts were not completely new to us. The initial foray into recording interviews consisted of a few snippets from the interview which were recorded in a .WAV format, converted to .MP3, and zipped for transmission. Given the size of sound files and the limitations of the transmission system and recording software we used, there were concerns of putting a major burden on the system with recording and transmitting up to 30 questions. Unfortunately, limiting the number of questions to that few did not meet the needs of the sponsors and researchers. Our team had to make our best guess as to the method and formats for the system given that we were limited to Blaise version 4.8.1b1403 at the time.

3.1. Homegrown Approach

We discussed recording alternatives and after a couple of false starts decided our 'best solution' to integrate recording was twin alien procedures - a CARI Recording procedure and a Snap Shot procedure. An alien procedure is additional code outside of Blaise used to perform specialized functions. In our application we used the alien procedure method to control turning on and off the recorder and to call the program to capture screen shots. Blaise allows us to do some very versatile work but we needed to pay close attention to the suggestions and limitations – see message from Blaise Help:

- **WARNING!**
  "There are no restrictions for procedures and methods. You may develop any method for graphics, complex algorithms, sophisticated sound techniques or other advanced applications. Take notice that DLL and COM procedures and methods are executed outside the Blaise system and any malfunction of using them is at your own risk. **We strongly advise you to test methods and procedures thoroughly before using them.**"

We combined the requirements, early sample questions, a base lined ACS Blaise instrument and sample data to establish a "proof-of-concept." The twin alien procedures, though bound very tightly to the data structure, were able to produce sound (.WAV) and image (.JPG) files. We proved early on that we could produce very good quality files and lots of them. However, these files were only created one time for each CARI question - the first time the question was entered. Running the homegrown version we were able to confirm that the sound and image files would be very large and possibly challenging to transmit through the existing control systems for both CATI and CAPI.

3.2. Homegrown Issues

Besides the fact that this approach only produced one set of files per CARI question, it also required a significant amount of code to implement. We had to add numerous variables for each question to track whether the image or sound was to be recorded, start and stop flags, hard coded variable name fields to meet output requirements, and a lot of other tracking information. Each field eligible for recording had to be checked during execution of the rules. We had to know where the interview was coming from to turn on and off the recorder and all of these fields had to be kept on route all of the time. The procedure for recording kept growing longer and longer as CARI variables were added to the list. We far exceeded the length of a block and had to subdivide the task. Attempting to add the other requirements (different paths, different CARI flags, etc.) to the CARI code made the sheer size of this code massive and un-maintainable. As mentioned previously, there were 238 unique questions that needed to have CARI related code applied to them.

After much work we were able to get the homegrown CARI to run. However, this method left much to be desired and did not give our customers everything they really wanted. Irregular navigation was another challenge that this method could not handle. But, we did learn a lot about what we could not do or change and how hard it is to keep track of recording and picture taking.

4. Blaise 4.8.2 CARI

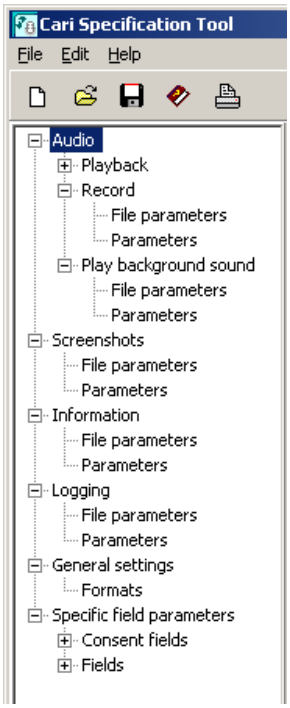
About this time a new Beta version of Blaise CARI was released. We decided that it would be worthwhile to research this to see if it could address some of our issues. It is said that technology sufficiently advanced enough looks to the primitives like magic. This is how it seemed to us, moving away from our homegrown version of CARI to the Statistics Netherlands implementation. When Blaise

4.8.2 build 1500 was released for beta testing, we took a giant step toward satisfying the sponsor's requirements for collecting sound and image files for a Behavior Coding System – and making it much easier for the programmers.

We quickly did a proof-of-concept using Blaise 4.8.2 CARI, using the same ACS Content Test instrument we had been working with for our homegrown version. It seemed promising enough that we decided to switch gears and go with the Blaise CARI implementation.

4.1. The BCI File

Blaise 4.8.2's crown jewel for us is the .bci file. This is a valuable addition to the Blaise suite of products. The CARI configuration file allows a great deal of control over settings used for CARI: sound/image quality; compression; the creation of images and sound recordings. What a great tool!



Developers may run the CARI Specification Tool from the Blaise Control Center. Not all of these options are used for every application. We feel that once the parameters are set many of these options can and should be shared by any survey in the agency needing a CARI component. Note any unused options were left at the default setting or disabled if possible.

This tool may also be accessed stand alone by launching the **BLCariSpec.exe** program. It is shipped with the Blaise software and can be found in the Bin directory where the software is installed. We use this program application with our testing application to give sponsors and managers the opportunity to review our configuration settings. This can also be used as a testing and development tool. This tool was very helpful when researching the various settings to use for CARI.
Let us review the relevant menu options and settings used in this specification.

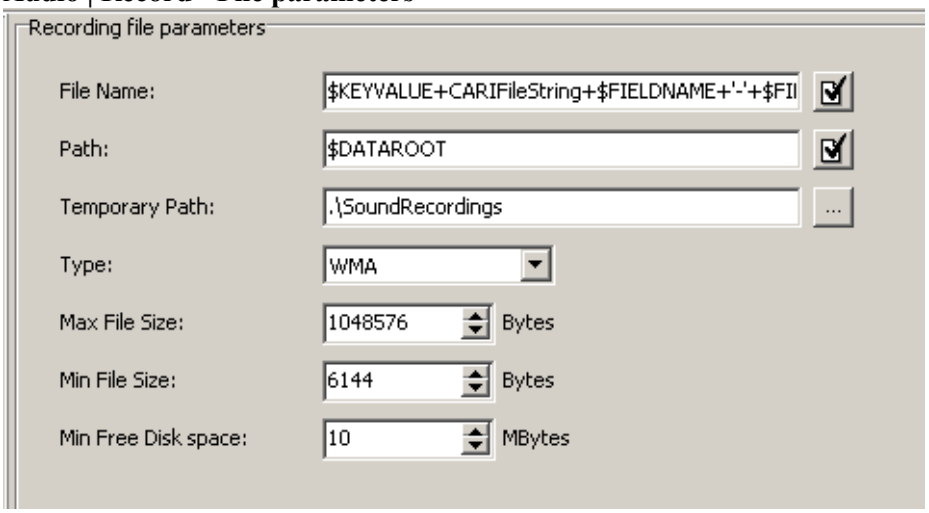
The challenge here is to figure what settings one should use. In an ideal world we want crystal clear screen shots, crisp and clear recordings, and very small file sizes so that transmissions and systems run smoothly and do not get bogged down with large files. In practice however, we decided to use recordings and screen shots of a lower quality to reduce file sizes and mitigate the burden on our transmission and control systems.

4.1.1. Setting the Sound/Image File Names

Determining the appropriate file names for the sound and image files is very important. We were somewhat restricted in file naming due to our earlier development work. You will obviously want to make sure that you have a unique name for each CARI file name. The latest version of Blaise 4.8.2 assists in this endeavor by adding a unique counter to the end of each sound/image file name. This helps differentiate between multiple visits into the same CARI field. Our file naming consisted of 4 major pieces: a unique case identifier, some case specific information, the full Blaise variable name, and the

date/time the recording was created. Setting up your sound/image file name in the BCI can be a little tricky. Here is how we did it.

Audio | Record - File parameters



File Name = $KEYVALUE+CARIFileString+$FIELDNAME+'-'+$FILEDATE+$FILETIME

The filename is derived from the case primary key value + our CARI specific file string maintained in the code + Blaise dot notated fieldname + File date + File time as described in the General Settings | Format. We use the exact same naming convention for the Image files so the names would match until the time/date stamp.

Example: 2CSS00401abcde3272200Sect03.Housing.COMPUTERC-20100603131428(01).wma
2CSS00401abcde3272200Sect03.Housing.COMPUTERC-20100603131434(01).jpg

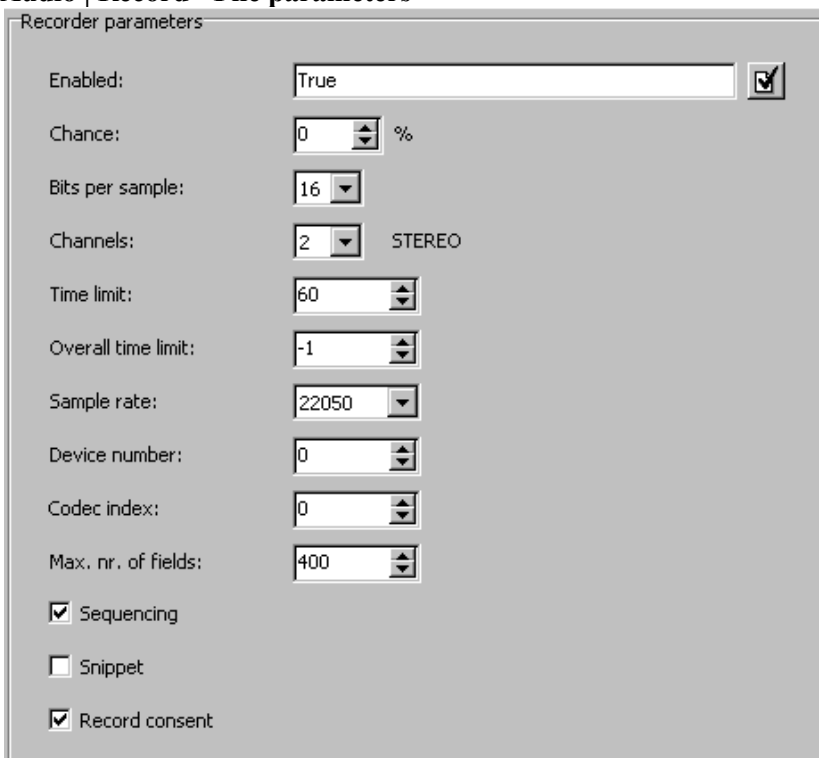
Note that the "(01)" is the counter (sequencing number) set by Blaise and will increment by 1 each time a recording/image is created for the same question.

4.2. Setting the Sound Files

There are 3 different sound file formats to select from - WAV, WMA, and MP3. We selected the Windows Media Audio (WMA) format because it gives us small files of acceptable quality. Another important consideration is the "**Min (minimum) file size**" (in bytes). We decided to set this to 6kb (6,144 bytes) which, based on our research and settings, should be less than 1 second of recording. If this setting is not set, Blaise generates sound and image files each time the interviewer moves through the CARI question. We decided that recordings less than 1 second in length were not useful and would most likely be created due to interviewers quickly moving backward or forward through the instrument. This could end up creating a lot of extra files that would need to be transmitted from the laptops. These temporary files are stored in the "Temporary Path" (we left it as: \SoundRecordings) and are discarded when the instrument is exited. However, the Blaise CARI Log file will track these so you will know they were created.

4.2.1. Global Sound Recording Parameters

Audio | Record - File parameters



These are the global record settings. **Enabled** = "True" and **Chance** = "0%" means that the recorder will use the field specific settings to determine whether or not to record a question. We went with a "**Time limit**" of 60 seconds. That means that the recorder will automatically turn off after 60 seconds of recording the same question. Otherwise, the recorder shuts off as soon as the interviewer exits the CARI field.

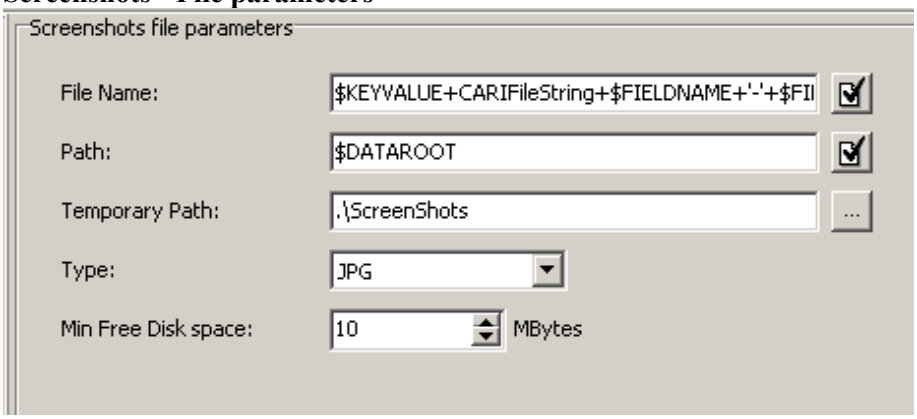
There is a wide range of settings allowed for recording quality. The general rule of thumb is to match the quality of the recording with the equipment being used: the quality of microphones for recording and playback speakers or headsets will contribute to the decision. Why record in concert quality and playback on $1.99 speakers? In addition, the quality of the files both for sound and images determines the file size.

The "**Sequencing**" item turns on the Blaise counter so that 01, 02, etc. is assigned to multiple recordings of the same CARI question. Checking the "**Record consent**" item tells the recorder that Consent must be "true" in order to record any CARI question.

4.3. Screenshot File Settings

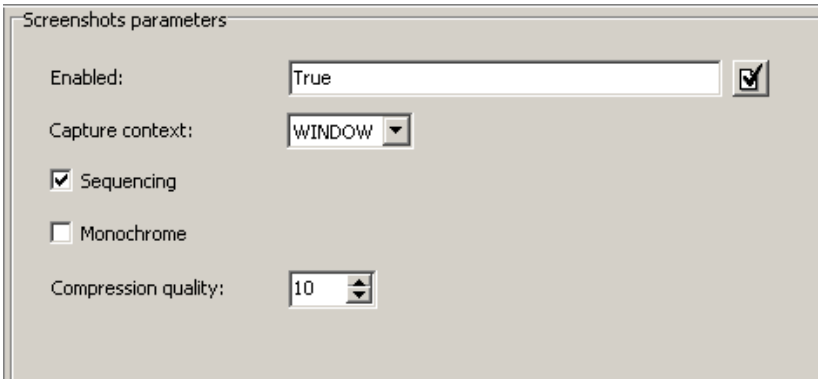
There are 4 different image file formats to select from - .BMP, .JPG, GIF, and PNG. The primary concern has been compatibility with the systems used to process output. We selected JPG initially because bitmaps are very large. It was also selected because JPG was the original file format that was picked when the homegrown CARI solution was being developed and our CARI processing systems were designed to look for and process JPG images.

Screenshots - File parameters



The same file naming convention is being used for the image files (screen shots) as is defined for sound files. In this way it is easier to process the files so that the Behavior Coding system is able to match the sound/image pairs together during review.

Screenshots - Parameters



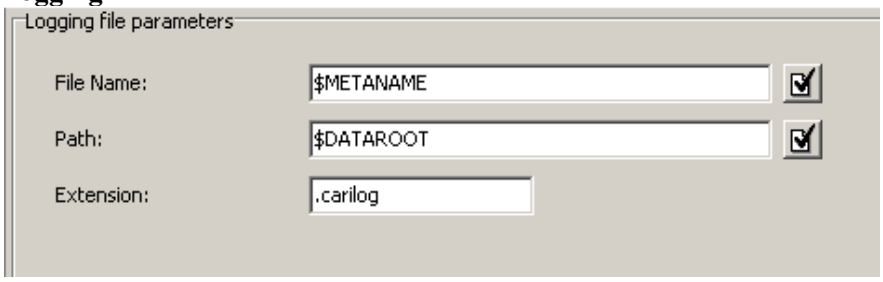
There are three different "**Capture Context**" settings; Form, Window, and Screen.
These set the image perimeter of the image capture. We desire to have the status bar, survey name, and version on each image so we chose the "WINDOW" (full Blaise screen) option.
Again, "**Sequencing**" is checked to add the counter to the end of the image file names when more than one image is taken for a CARI question.

"**Compression quality**" allows the setting for the level of detail (quality) for the JPG file. The larger the level the better the picture and the larger the image file. We went with a very low setting here (10%) in order to minimize the size of the JPG file as much as possible.

4.4. The CARI Log File

The CARI Log file keeps track of all of the image/sound files created during the interview, even the ones that are deleted once the case is closed. This is similar to the audit trail file. We discovered it helpful to have some ability to audit and analyze recording activity. The .carilog file and .adt audit trail have been very successful thus far in being able to recreate and account for any unusual issues observed in testing. Sometimes the behavior in the instrument is not what the tester expected, but we were able to explain what happened after stepping through the audit trail and CARI log files.

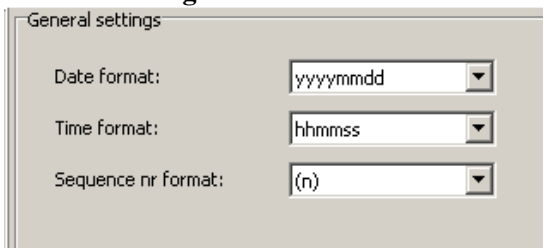
Logging



4.5. General Settings

The Beta versions of CARI did not contain these options. When we created our filename we used the $DATE and $TIME variables in the configuration file for file naming. This did not work as expected since all files created for a session had the exact same date and time. This happened because the temporary files are renamed when you exit a session. Using the "general settings" options eliminated this issue, and now date and time are correctly set to the date and time the sound/image is captured in the CARI file names created.

General Settings



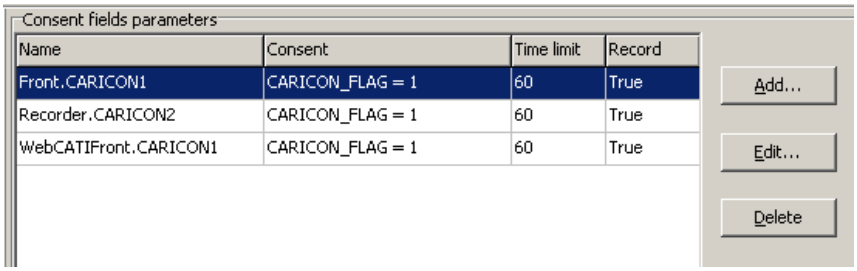
The "**Sequence nr format**" option gives a number of alternatives for bracketing the sequence number. We use "(##)", but other systems may prefer other bracketing (e.g., $, %, ~, }, etc.).

4.6. Specific field parameters – Setting the Consent Fields

Consent is an interesting topic that deserves a more detailed discussion. But, as far as the BCI file settings are concerned, if you use consent for CARI then you will want to set at least one consent field in your BCI file. If the "Lent" condition for your consent field evaluates to "True" then the CARI fields defined in the "Fields" section will be recorded if their conditions evaluate to "True". If the "Lent" condition in your consent field(s) does not evaluate to "True", then no recordings will be made regardless of the settings in your "Fields" section.

For our project we needed the ability to collect consent from 3 different locations – CAPI front, CATI front, and from a parallel tab that allowed the respondent to change his/her mind mid-interview. Therefore we added 3 consent fields to the BCI file:

Consent fields



The trick here is that the consent "Lent" condition had to evaluate to "True" for all of our consent fields so we had to be careful on how we set this up. Unfortunately for us, we could not take full advantage of Blaise and had to add extra code in order to address our requirement of asking consent twice – the first time we could not record it (we called this verbal consent) and if the respondent agreed, then we had to ask a second time (the actual CARI consent question) while recording it. The challenge here was, if the respondent changed his/her mind and said no at the verbal question, we did not ask the CARI question so we could not control the recorder easily. Instead we created a separate variable – CARICON_Flag – and had to do all kinds of code gymnastics to keep this flag up-to-date based on all the possible ways consent could be asked/changed. This approach appears to work fine for us, but it is not ideal. Newer versions of Blaise 4.8.2 have a setting that allows us to not save the recordings for "consent" if the respondent says "no". We are hoping to take advantage of this feature for future projects using CARI, but we would need approval from the U.S. Census Bureau's legal staff first.

4.6.1. Considerations with Handling Consent in the Instrument

If you conduct CARI and require the respondent to give his/her consent to be recorded before recording any questions, most likely you will add a consent question near the front of the interview. This works alright unless the following happens:

1. The respondent changes his/her mind
2. The interviewer changes respondents

4.6.2. What if the Respondent Changes His/Her Mind?

Here, the expectation would be as soon as the respondent says "no", the instrument must stop recording. Obviously, the Field Representative (FR) would need to modify an answer in the instrument in some way so that the recording stops. Expecting the FR to back up to the front and change the Consent field is not very realistic. Therefore, we added a parallel tab to the instrument that the FR could access at any time. We made the question in this tab a "consent" question in the BCI file and controlled turning on/off the recorder in this method. Technically the respondent could change their mind from No to Yes, Yes to No, or some crazy multiple combination of each.

4.6.3. What if the Interviewer Changes Respondents?

In some of our survey instruments we allow the interviewers to change respondents in the middle of the interview. One could instruct the FR to simply visit the parallel consent tab and re-ask consent. However, our sponsors did not think this was the best approach. Their feeling was the interviewers may not remember to re-ask consent in this situation. To address this we added code to the Respondent tab that "forces" the instrument into the consent tab if the respondent is changed during the interview.

4.7 Specific field parameters – Setting the CARI Fields

Fields Parameters are the heart of the BCI CARI configuration! Being able to select CARI fields using this BCI file setting eliminated thousands of lines of code for us.

Fields parameters

Name	Enabled	Chance	Time limit	Snippet
Sect03.HousingA.ACCESST	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False
Sect03.HousingA.BROADC	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False
Sect03.HousingA.BROADT	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False
Sect03.HousingA.COMPUTERC	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False
Sect03.HousingA.COMPUTERT	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False
Sect03.HousingA.COMPUTYPC	((RT1002.CARISAMPLE = '1') OR (SPANFLAG = 1))	100%	60	False

Add... Edit... Delete

Here you simply add the fields you want to record, specify the conditions for recording the field, indicate the chance you want it to record if your conditions are met, add the upper time limit (if you have one), and indicate if it is a snippet or not. **Snippets** will record until the **time limit** is reached (starting from the CARI question). Snippets set to "False" indicate that the recorder will turn off as soon as you exit the field or after the upper time limit is reached.

For our implementation the sponsor pre-determined the sampling for the possible CARI questions for each case. So we used the value of the CARISAMPLE variable sent in on input to determine whether to record or not. This gave the sponsor the ability to subdivide their sample and collect recordings on different topics by case.

Name	Enabled
Sect04.Person4A[1].ANCW	((RT1002.CARISAMPLE = '2') OR (SPANFLAG = 1))
Sect04.Person4A[2].ANCW	((RT1002.CARISAMPLE = '2') OR (SPANFLAG = 1))
Sect04.Person4A[3].ANCW	((RT1002.CARISAMPLE = '2') OR (SPANFLAG = 1))
Sect04.Person4A[4].ANCW	((RT1002.CARISAMPLE = '2') OR (SPANFLAG = 1))
Sect04.Person4A[5].ANCW	((RT1002.CARISAMPLE = '2') OR (SPANFLAG = 1))

The image above is an excellent example of how to specify recording arrayed variables. We agreed with the sponsor to allow up to five people in the household to be recorded. Sect04.Person4A[1].ANCW in this example is the first household roster person ancestry question. We then had to include each pertinent subscripted field in the BCI file. This is very easy to set up.

5. CARI Testing Considerations

In addition to the usual instrument testing that takes place, CARI requires additional testing. One of the main objectives of testing CARI is to verify that the instrument is creating sound and image files when you expect it to and that it does not create sound/image files when you do not expect it to. As noted earlier, the requirements for the content test were fairly challenging. The instrument contained 2 unique paths for content test questions. Some of the CARI questions overlapped and some were unique in each path. Combine that with the 5 different CARI sample flags, the "Record All", and the "Record None" options and you have 12+ unique scenarios with many more potential CARI paths through the instrument. The authors and the testers certainly had their work cut out for testing. Besides verifying the sound/image files, you must verify the file names, quality of sound/image files, that consent works as expected, that new system interfaces work correctly, and so on. We also needed to spend time advising the sponsors on ways to test CARI since this was new for them as well.

5.1. Helping the Sponsors Test CARI

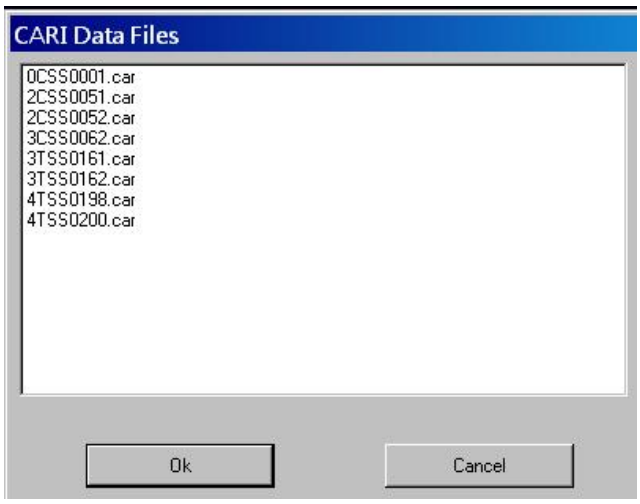
Early on we realized it would be a challenge to test CARI and realized that it would be important to identify which fields should be recorded based on the input and scenario. So one of the things we did was to add a fill to the question text for CARI variables which displays in red on the top line of the infopane:

CARICARI***CARI***

This fill is assigned in the main driver program for testing and used the same conditions as the CARI settings. This really makes a CARI variable stand out for the testing team and developers. Of course this must be "turned off" for production.

The idea behind this approach was to indicate to the testers that the question was being recorded so they could verify the scenario was working and verify that sound/image files were correctly generated for the case. The challenge was making sure the conditions for filling the red CARI text exactly matched the conditions set for recording these fields.

The next step for the sponsors was to verify all sound/image files were correctly created, verify the sound matches the image, and verify the image matches what was entered into the question. Since we zip up all sound/image files into a <case ID>.car file we created a program in our test environment to display all cases that were tested for that instrument:



Once the sponsor selected the case they wanted to review, all image files were displayed. The sponsor was then able to review the file names or click on the first JPG where the program automatically played back all sound files while cycling through the image files. The program also indicated the size of the zip file and estimated the transmission time for the file. This was done to stress the size of these files and potential transmission issues with the volume of recordings and images being created for a case. Once we moved to a newer version of 4.8.2, we were able to scale down the quality (size) of the image files so this became less of a concern.

5.2. Testing CARI Over the Network

There were some interesting issues which developed when testing CARI over the network. One difficulty was that the various divisions had different microphone settings and default sound and image display software. Across divisions and even within branches computers were different due to age, memory, software, and so on. We had to work with the testers to setup the correct settings on their PCs.

We also requested that sponsors test CARI on the FR laptops since those are running a very different software and hardware configuration than our office PCs. The FR laptops run Windows Vista with wide screen display at a higher resolution.

5.3. CARI Testing Tips

Some things to keep in mind for testing CARI:

1. Verify that sound/image files are created when you expect them to be. You also want to verify that you create "pairs" of sound/image files and that no unpaired (extra or unexpected) ones are created.

2. Verify that consent is working correctly. In other words, verify that sound/image files are not being created when consent is "No" and verify they are being created when consent is "Yes". All possible ways to answer and change the value of consent should be tested and verified. This was one of the most crucial things for us to test.

3. Verify every potential CARI field will create a recording if needed. One good way is to print out your BCI file settings using the "File – Print" option from the CARI Spec Tool. Example:

```
Field Name   Sect03.HousingA.COMPUTERT
Enabled:     ((RT1002.CARISAMPLE = '1')  OR (SPANFLAG = 1))
Chance:      100%
Time limit:  60 seconds
Snippet:     False
```

   This will list the conditions of every CARI item defined in your BCI file. One can quickly scan through this to verify the settings match the requirements. We initially had a few typos in our settings file and this list helped us identify those. Another method for documenting the CARI file settings is to use **BRegEdit.exe** to create what we now refer to as the "barf" file. This tool will list (barf out) the CARI specification in a file suitable for documenting and printing

4. It is beneficial to have two people testing – one reading the question and another responding. This will help give more realistic sound file sizes and will give people an opportunity to review the sound quality.

5. For CATI, special devices were purchased that connect the phones and CATI workstations which allow the CATI machines to record the phone conversation. For testing CATI, you should make sure testing is conducted using the phone and another respondent.

6. Verify that the settings in your BCI file are working as expected. For example, we set our minimum sound file size to 6kb. We then purposely created small sound files by moving back and forward through the instrument quickly and verified these files did not get saved after exiting the instrument. The CARI log file is a good way to verify this as well. We also

reviewed the size of all the sound files that were transmitted in from the systems testing and verified all were greater than 6kb.

7. The CARI Team conducted an end-to-end full systems test to verify each step of the process created and passed along the expected files. They verified that the correct sound/image files were created on the laptops, those same files were packaged correctly in the zip (CAR) file, the files made it back to HQ, and finally the correct files were processed through our Master Control system into the CARI Behavior Coding system.

8. Verify that instrument lag is not introduced when modifying BCI file settings.

6. CARI System Testing Results

We conducted our final systems testing using Blaise 4.8.2b1529. This was the latest version we were able to use given our timing. Results documented will reflect this version of Blaise unless noted otherwise.

6.1. Sound File Results

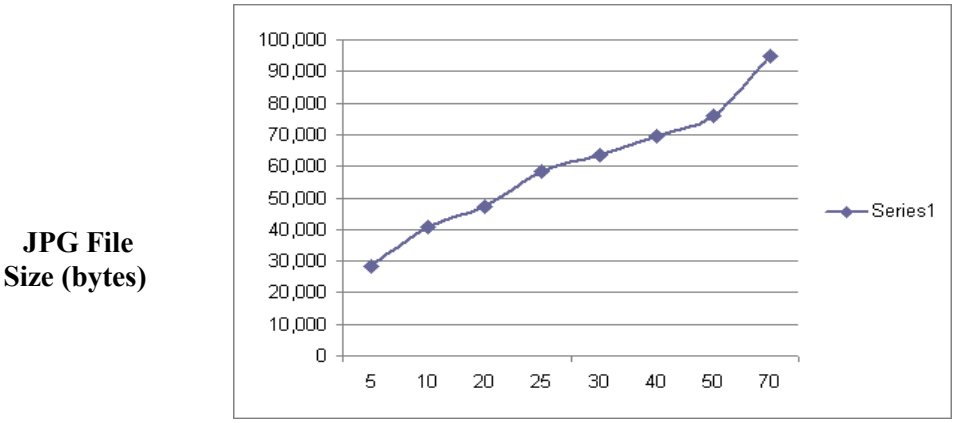
Since the MP3 files generated by Blaise were very large, we decided to use WMA files. WMA files are much smaller than the MP3 files. The sound quality was not as crisp, but the CARI Team agreed the quality was acceptable for use in the behavior coding system. It is difficult to get exact comparisons since the size of the sound file has so many factors – length, outside noises, computer settings, etc. Our research showed that the Blaise WMA files were about 12 times smaller than the MP3 files. In addition, WMA files compress better than MP3 files giving us even more file size savings. (Note that some of this testing was conducted with an early version of 4.8.2).

The size of our WMA files ranged from 6kb (a little over a second) to 68kb (about 60 seconds). We found it interesting that the average size of the CATI sound files (using the recording devices connected to the phone and PCs) was larger than the CAPI sound files (from laptop). CAPI WMA file sizes ranged from 6kb to 40kb while CATI ranged from 6kb to 68kb. The median file sizes were around 10kb for CAPI and 14kb for CATI giving CATI about a 40% larger size. The 60 second sound file was about 60% larger for CATI (68kb vs 40kb). We have not been able to determine the reason for this difference in file size. One thought is that the CATI recording devices used in the phone centers may pick up more background noise. Also, our CATI testing may have had more conversation than our CAPI testing.

6.2. Image File Results

In our initial testing of the Beta version of Blaise 4.8.2 we noticed that the sizes of the JPG files were very large. Many JPG file sizes were well over 100kb. Given the number of CARI questions to be recorded and the potential for multiple JPGs per question we investigated other image formats. The BMP files were even larger so we tested the GIF format. The GIFs looked very promising - the image quality was adequate and the size was very small. As we began to investigate using this format we discovered an issue with it. Using the GIF image format caused a significant lag in our instrument. It added a 2-6 second lag when exiting a CARI question. This lag was not acceptable since it would disclose to our interviewers which questions are being recorded. After discussing concerns about image file sizes with Statistics Netherlands, a new feature was added to the JPG image format called “compression quality”. This feature allows us to scale down the quality of the image in order to decrease image file sizes.

Compression Quality was a very useful enhancement that allowed us to dramatically reduce the size of the JPG files. The question with this new compression quality (CQ) setting was to determine "how low do you go?" To help us answer this question the CARI Team compared the image quality and size for the same screen shot with various CQ settings. The following graph shows how the size of the JPG file changed with the various CQ settings:



Compression Quality Setting (testing with Blaise 1520)

The CARI Team decided on the 10% CQ setting to take advantage of the file size savings. There is definitely image quality degradation with this setting, but the CARI Team agreed that the quality was adequate to read the question text and make out the answer entered. During this testing, we also noticed that lower quality JPGs compress better which translates into even more file size savings.

Our systems test findings indicated, as expected, that the images taken from the phone centers (4x3 monitors with lower resolution) were smaller than the images taken on the laptops (widescreen, higher resolution). Image file sizes ranged from 32kb-52kb for CAPI and 27kb-40kb for CATI (CAPI was about 20-30% larger).

6.3. Comparing Sound Files to Image Files

Even with the savings in the JPG file sizes, the images were much larger than the sound files in our testing. For CAPI, the JPG file sizes averaged about 3.4 times the size of the WMA files. CATI had a smaller difference with JPG file sizes averaging about 2.1 times the size of the WMA files. The median WMA file was about 10kb CAPI/14kb CATI while the median JPG file was about 38kb CAPI/30kb CATI.

6.4. Overall File Sizes

From our system testing we are expecting the average compressed case to increase about .25 megabytes in file size. This average includes cases that did not contain any sound/image files (which is what we expect in our production), so CARI only cases would have a larger increase. Production may yield different results since many of the system test files were scenario based. We are using a 6kb minimum sound file size to try to reduce the amount of "extra" files that would be produced by FRs moving back and forward throughout the instrument. We are hopeful that eliminating these files will save a lot of space from extra sound/image files being created that really are not needed. It is hard to determine exactly where to set this number since file sizes vary and the sponsors do not want to lose any

conversation they may contain. We will have a better idea of these numbers after our production test later this year.

6.5. Outstanding Issues

Since we conducted all of our systems testing with Blaise 4.8.2 build 1529 we are moving forward into production in this version. During our verifications test, the CARI Team identified two issues with this version of Blaise. Both of these issues worked correctly in prior versions of Blaise, and we believe both have been resolved in the latest version of Blaise 4.8.2.

1. The sequencing of the Blaise sound/image file names was not working as expected. All files created for a CARI question were assigned a sequence number of "01" regardless of how many times the question was visited.

2. When the max recording time is reached for a CARI question (60 seconds in our instrument) – meaning the CARI question was not exited before the max time was reached – the image file is not created. The sound file is correctly generated, but there is no corresponding image file.

7. Conclusion

When implementing CARI into a Blaise instrument, we are looking for something that creates small, useable sound and image files without noticeable instrument lag. We also want to be able to control when these files are created and make sure that irregular movement in the instrument can handle the creation of these files. More importantly, we are looking for something that is fairly simple to implement into an existing Blaise instrument.

With Blaise 4.8.2 CARI we now have the ability to do just that. This software made our lives so much easier. The most challenging part for us was dealing with the consent issues because of our agency specific requirements. Newer versions of Blaise 4.8.2 give us more flexibility with how consent can be handled which will hopefully make things easier for us to implement in the future. Another challenging part of implementing CARI is determining the ideal settings for the sound and image files. Once the settings that work best for you are determined, you are well on your way to implementing CARI. The BCI file and editor (BlCariSpec.exe) are useful tools that simplify the selection of CARI fields. The editor also allows you to easily try out and test various sound and image file settings. If you desire, Blaise CARI also provides the flexibility to use more than one BCI file with your survey.

We wish to thank Statistics Netherlands for adding CARI into Blaise. After attempting to implement fairly complex CARI ourselves, we can appreciate this new component to the Blaise software.

8. Acknowledgements

The authors would like to acknowledge the American Community Survey Office (ACSO) CARI team for their contributions and collaborative efforts. They were a huge help with identifying requirements, testing, and identifying issues with our various versions of CARI. The views expressed in this paper are those of the authors and not necessarily those of the U.S. Census Bureau.

Experiences with Blaise for Data Editing in the Housing Rental Survey

Harm Melief and Mark Kroon, Statistics Netherlands

1. Introduction

This paper reports on a data editing system designed and built at Statistics Netherlands for use in the housing rental survey. The survey collects data on housing rental, which is used to calculate price indices f or housing rental costs and which is also supplied to the Dutch Government. The data is collected partly from paper forms and partly using electronic sources, all of which have to be corrected manually. Chapter 2 describes the survey in more detail and will explain what features a system should contain to support the survey.

The resulting system is described in Chapter 3. The system supports data import into a relational database, scanning it for mistakes, correcting them, calculating the price indices and exporting the results. All these properties were implemented in the Blaise Suite, using Manipula, Maniplus and Blaise. The resulting system is fairly complicated, the implementation of which presented us with a number of unforeseen problems. In chapter 4 our experiences with using Blaise for data editing are described. Finally, these experiences are discussed in chapter 5, where our final conclusions are given.

2. The housing rental survey

2.1 Description of the housing rental survey

The housing rental survey (huurenquête) is conducted by Statistics Netherlands (CBS) and (partly) commissioned by the Dutch Government. The main output of this survey is the housing rental price index, which is computed for both the normal and harmonized rents. Secondary outputs are the garage rental and service charges price indices.

The housing rental survey is a stratified fixed panel survey of some 14000 (rented) houses from approximately 1300 respondents, i.e. rental agencies. Each year a few hundred houses drop from the panel, usually because they are either sold or demolished. Therefore, the panel is yearly replenished from a group of new rental houses. Moreover, every few years the panel is updated to remove any bias and fill out the strata if needed.

Of the roughly 14000 houses 9000 rentals are collected using paper forms, 4500 are obtained from the administrations of large rental organizations and some 500 are provided as electronic forms (Excel). The paper forms are sent out and received by a special department within Statistics Netherlands. They are sent out in May and usually received over a period of one to two months, starting from the beginning of July. The filled-out paper forms are manually entered into an Microsoft Access database, which is updated daily during the response period.

Approximately 50 variables are measured, most of which are used to determine the housing rental price indices. Some variables, however, are specifically collected for the Ministry of Housing, Spatial Planning and the Environment (VROM), which uses this information for setting housing policy.

All forms are inspected for inconsistencies, omissions and mistakes, after which they are manually corrected. Small and obvious mistakes, or mistakes resulting from the typing out process, are immediately corrected, but for most problems the respondents need to be contacted. Much of this work is done by temporary workers. For optimum efficiency all imperfect records of one respondent are collected and treated in one phone call. Thus only one person is responsible for a given respondent, which guarantees that respondents are not needlessly contacted multiple times. Distributing the work amongst the workers is something of a challenge, as will be explained in chapter 3. Preferably all errors are corrected, but to remain cost effective, some are ignored, especially if the involved variables are not used during the calculation of the price index.

Based on 4 categories, type of housing (2), type of rental agency (2), year of building (4) and geographic region (4), the housing rental price index is stratified into $2 \times 2 \times 4 \times 4 = 64$ strata. For the rental prices the following method is used to calculate the price index. Within each of the cells, the sample mean of all observed and corrected records is calculated. The results of the cells are combined using a weighted average to give the average rent of the country, from which the index is calculated as the ratio of the current and the last period. The weights for this calculation are supplied by VROM, which has information about the complete housing rental sector. For the garage rents and service charges the price indices are calculated through an unweighted average of all respondent and corrected records.

2.2 A supporting system

Discussions with the supervisors of the process helped us define the main features of a support system for the survey. As described above the main purpose of the process is the editing of the respondent data and the calculation of the price index. The task of supporting the data editing consists of three parts. Firstly, a large set of rules is needed to detect any imperfections in the forms. Secondly, an incorrect form needs to be conveniently presented for correction. This means that as much information as possible should be presented in one screen and that related variables should be grouped together. Thirdly, due to the large number of forms that need to be corrected, an efficient system for distributing the work amongst the temps was desired. Calculating the price index is a relatively straightforward set of aggregations, which can be supported by many different tools.

The system of course needs to support the import of the survey forms, but there are also a number of supporting data sources that need to be imported. At the end of the process the computed results, but also some of the micro-data needs to be exported for use within our organization. Just as for the calculation the I/O can be performed in many different tools. This means that the most distinctive feature of the desired system is support of the data editing. It was deemed that the Data entry Program of Blaise could support this feature, while Manipula should be able to support the I/O and the price index calculation.

3. Support system in Blaise

We have built a system, which supports the process described above, in the Blaise Suite version 4.8.1, build 1447. In this chapter the main parts of the system are described: data I/O (section 3.2), data editing (section 3.3), calculation (section 3.4) and classification management (section 3.5). The basic architecture of the system and the coordinating main menu are described in section 3.1

3.1 General architecture

The system needs to support a large number of different functionalities. To organise the program, a home screen was built in the form of a Maniplus menu. This menu delivers a menu bar, from which the functionalities may be called up. These functionalities are grouped into three main groups: editing,

management and miscellaneous purposes. This last group contains both data I/O and the price index calculation. The management section contains both classification management and user rights.

As the most important function of the system is to support the editing (correction) of the survey forms, this forms the basis around which the system has been build. Consequently, the core of the system is determined by three data models:

1. Panel data: This contains the fixed data of the panel houses. Fields include the address and a link to the rental agency.
2. Respondent data: This contains the information of the rental agencies. Variables include the contact information and identifying properties of the agencies.
3. Variable data: This contains all the variables from the survey. This includes all the rents and service charges, as well as the properties of the houses, such as number of rooms and parking facilities.

These three data models are linked through the panel data model, as this model defines the house through its primary identifier, its address. Besides these main data models, a large number of supporting data models were needed for the different parts of the program. In the next sections these different parts of the program are described at which point the supporting data models are described.

Most data is stored in a SQL database. The data models were connected to corresponding database tables through .boi files. Since Blaise only allows joining one record at a time, we chose to combine data from the different data models by creating views in the database. For instance for the calculation information from all three database tables is needed, which was obtained by making a view in which these three tables are joined. These views can again be accessed through a boi file.

3.2 Data I/O

As mentioned in the previous chapter, the survey is collected either as paper forms, from the administration of a large rental agency organisation or from electronic sources. The paper forms come in two formats, introduction forms for new respondents and panel forms for recurring respondents. Including the two types of electronic data, this means that four statistical sources are used. Moreover, some 8 secondary sources are needed to supply supporting data. These typically include a ZIP-code register, information about the values of the properties and others. The sources have different formats, mdb (Access), xls (Excel) and text (csv and fixed columns).

The output takes two forms. A group of output is derived from the panel and respondent tables. These are used for the setting up of the paper forms at the start of a new survey. These outputs are all text files. In addition to this group there are some 12 different outputs required of different aggregates of the three tables. These outputs are delivered to different customers within and outside the statistics bureau. The most important customer is the government (VROM). These files are also all text files.

All these data are imported or exported using Manipula, which supports all of the required file types. The required data models for the input or output files were mostly programmed by hand, but for all mdb or xls files, which are accessed through a boi file, the Oledb workshop could be used.

3.3 Editing the data

The editing of the data consists of three parts, work assignment, main editing window and record based editing (in a DEP). These three approaches work together to form an interactive environment, which supports a group of data editors. Each of these components is treated in a different subsection.

3.3.1 Work assignment

Much of the actual editing work is done by temporary workers. As explained in the previous chapter mistakes or omissions are corrected by calling the respondents. For optimum efficiency all imperfect records of one respondent are collected and treated in one phone call, which means that the work assignment is grouped by respondent. To facilitate this process a list of all respondents with imperfect records is made. This list is presented in a separate window in a lookup, see figure 1. The list contains the name of the respondent as well as the number of houses, number of mistakes and types of mistakes. From this list entries can be selected which can be assigned and selected for editing (see next section) by using the buttons at the top of the screen. A number of filter options are also supplied, in the form of the controls at the bottom of the screen, which allow the temps to select only their own respondents or to prioritize unassigned respondents.

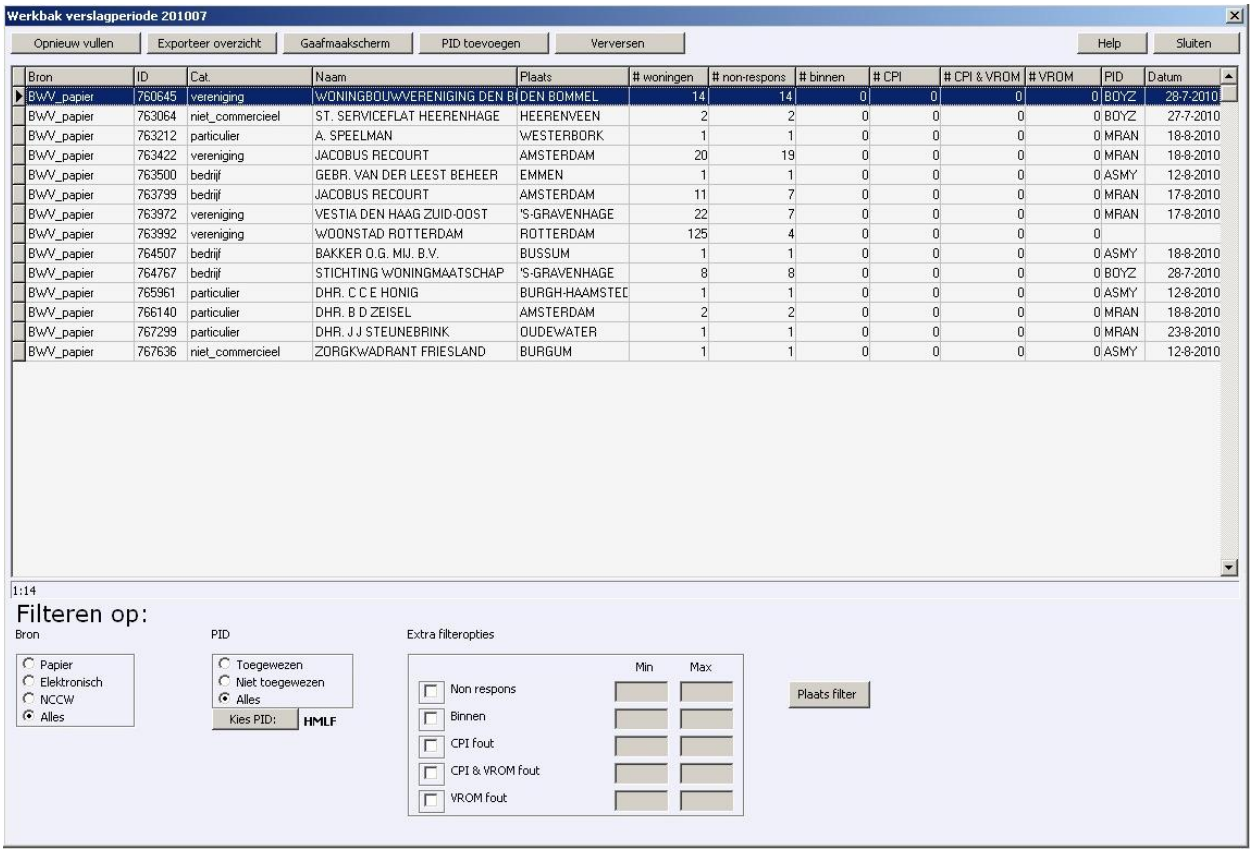


Figure 2: This figure shows the work assignment screen. The buttons at the top of the screen govern refreshing, refilling the list and allow the user to assign or open an entry in the main editing window. The controls at the bottom of the screen govern the different filtering options.

This screen was built in Maniplus. The lookup was found to be an efficient method of presenting large list of data. Moreover it could be integrated with other modules such as the main editing window. This allowed the building of an interactive system. The disadvantage of lookups is that they are difficult to sort, especially on different categories, which meant that the different filtering options had to be implemented. Moreover, we had some problems with refreshing of the lookups, see next chapter.

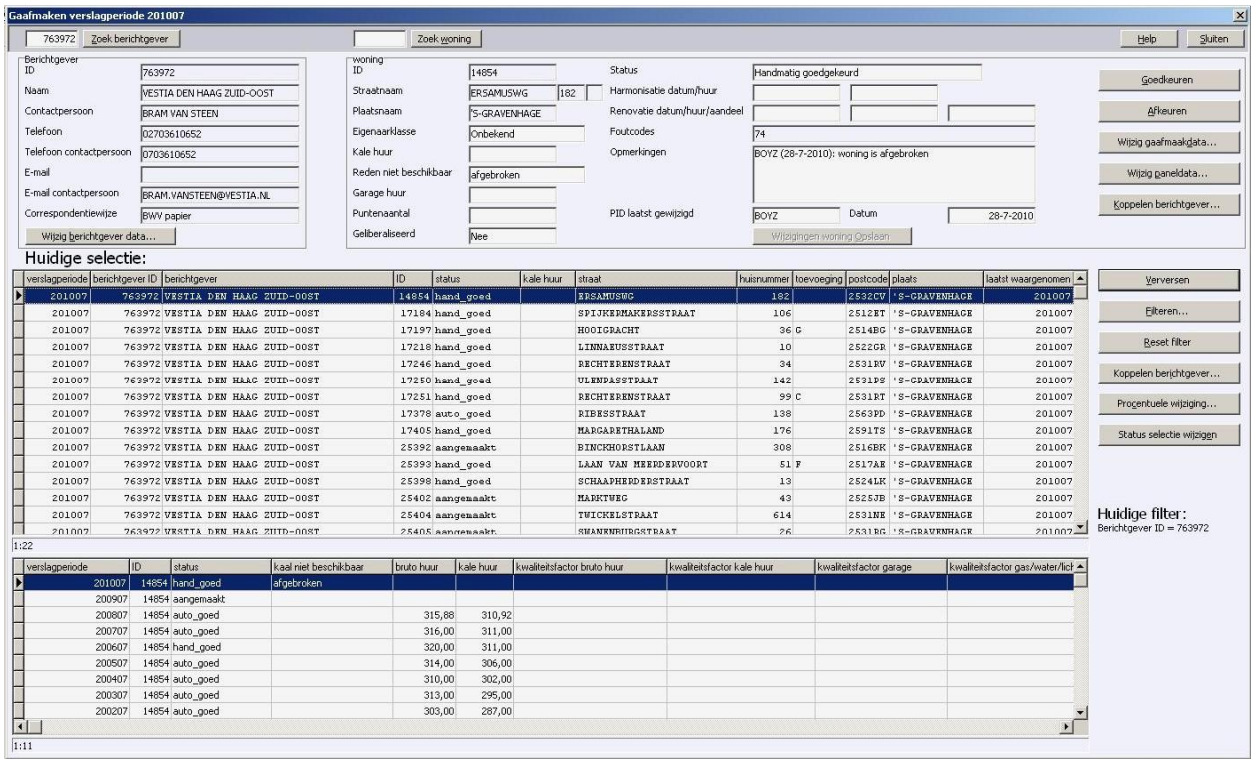


Figure 3: This figure shows the main editing window. This window was set up to edit groups of houses, typically all houses of one respondent. For privacy reasons the rental prices have been removed.

3.3.2 Main editing window

As explained above, the editing is simultaneously performed on all houses of one respondent. Therefore a screen was build which supports the editing of a group of houses, see figure 2. The upper lookup shows the selected group of houses, one of which can be selected. The lower lookup shows a time series of the currently selected house. The buttons to the right of the lookup govern all group-based actions (the first two bullets below). The controls at the top of the screen show detailed information on the currently selected house and the associated respondent. The buttons allow the user to edit some of the detailed information and open a DEP for the panel data, variable data and respondent data for the currently selected house.

In principle the group can be selected based on different criteria. Usually, all the houses of one respondent are selected. A number of actions can be performed in this screen:

- Changing variables of the entire group. Certain variables, such as the rental costs and also the status of a record (whether it is correct or not) may be changed for an entire group. This is governed by the buttons to the right of the lookups.
- Changing the filters to change the group size. As mentioned above, the typical group contains all houses of one respondent, but for larger respondents it often is convenient to further refine the filter, for instance to show only the records with mistakes in it. This is also governed by the buttons to the right of the lookups.
- Viewing and editing currently selected house: The details of the currently selected house are presented in the upper lookup. Double clicking one record of the list triggers a refreshing of the lower lookup, giving a time series of that house over the last ten surveys. This also updates the controls at the top of the screen. The block on the left shows information about the respondent of the house, which facilitates the contacting of this respondent. The button below the information opens a DEP, which allows the editor to change the information of the respondent. The controls on

the right give some of the panel and variable information of the house. The buttons allow the editor to change the status of the house and also to start the DEP's for these two data sources, which is treated in the next section.

3.3.3 Record based editing (DEP)

From the main editing window data entry programs (DEP's) may be launched to edit records in the three main database tables. Each of these DEP's is constructed from the corresponding data models. The layout of the DEP's has been defined in the layout sections which use modelib and menu files.

The panel and respondent data entry programs are relatively simple. They both contain only one double column with the names and values of the variables. Both sources use only one external data link, to the ZIP code register to check addresses which are entered. This check is also the only rule in these data models.

The variable data has a much more complicated layout, see figure 3. This data source contains most yearly changing variables and as such most editing is done in this screen. The DEP contains some 150 variables and another 50 auxiliary variables. It is spread out over two pages. To check for mistakes and inconsistencies approximately 150 rules have been defined.

The screen is set up in three parts. The buttons govern a number of functions, which are explained later on. The information blocks at the top of the screen contain external data from the other two main data sources. Below this information block the variable data is presented. The third and fourth columns are conditionally interchangeable, depending on whether the record represents a new (introduction) or a panel housing unit. Violations of the rules, which corresponds to errors or omissions in the data, are denoted in the usual manner using a red sign with a white cross in it.

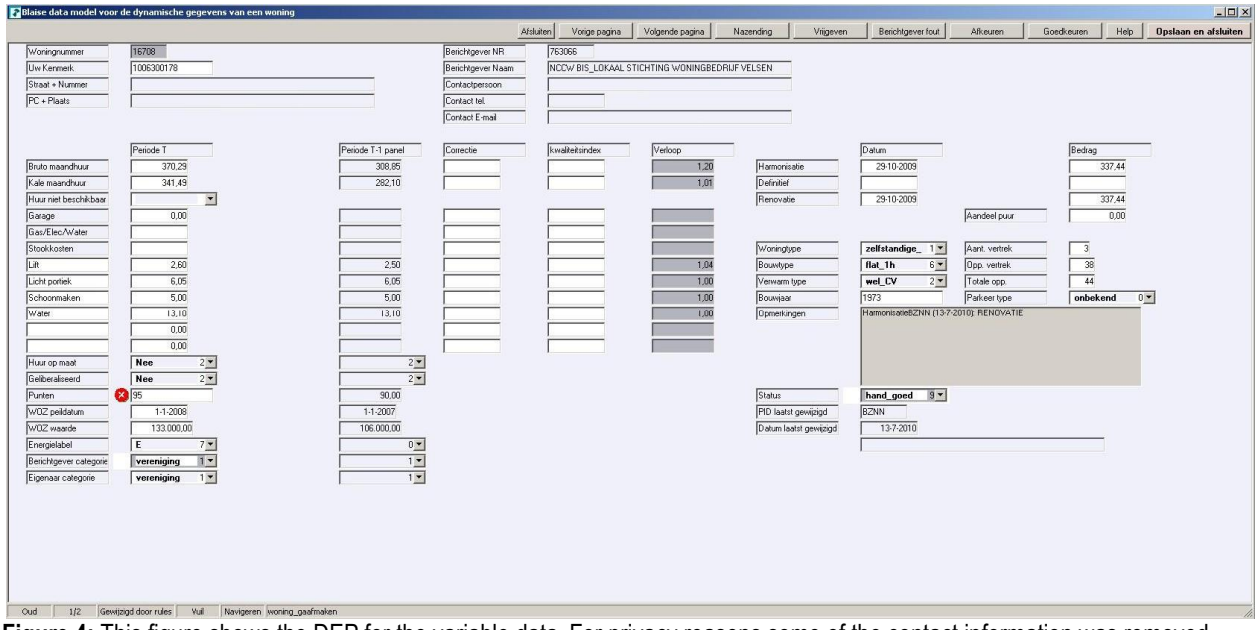


Figure 4: This figure shows the DEP for the variable data. For privacy reasons some of the contact information was removed.

All three DEP's contain a number of buttons defined in the menu file. The variable data has most buttons, which are mainly used for navigation and changing the status of the record (validate/reject). All three DEP's have a close button which starts a Maniplus script, which forces the user to place a comment if the

record has been altered. In that case it also records who the user was and at what time the record was changed. This information is used to supervise the temps.

3.4 Calculation

The calculation was performed in Manipula. As will be explained in section 4.2, a view was made, which joined the three main data sources. This view is used as the input for the Manipula script. The script uses a sort section for aggregation, in combination with two manipulate sections. In the first section each record is assigned to one of the cells mentioned in chapter 2. Also, all records that should not participate in the calculation, either because they were manually rejected or because essential variables are blank, are removed from the calculation. Subsequently, all valid records are aggregated in the sort section, after which the average rental prices for each cell are computed in the second manipulate section.

The results of this calculation are joined to the weights per cell that are supplied by the Ministry. In additional scripts the data are then summed to different aggregates. These include aggregates over provinces, municipalities and a total aggregate for the entire country. For all aggregates the average prices are calculated both for the current and former year. These averages are divided to obtain the price indices for the aggregates. The aggregate for the whole country is called the housing rental price index and is part of the consumer price index. These results are also used for some of the export products, mentioned in section 3.2.

For the service charges a different approach is used. Firstly, some 700 categories of service charges have been recorded, which are grouped into 13 categories (12 real and 1 residual groups). Secondly, unweighted (arithmetic) means for the entire country are used to calculate the average values for the former and current year. The averages are then divided to obtain the prince index for the 13 categories.

3.5 Classification management

Within the survey two types of classifications are in use.

- A large number of categorical variables are used, half of which are yes/no variables. The other half usually have between five and ten categories. These variables are defined using Blaise types, the advantage of which is that Blaise consequently enforces these types in the DEP and in Manipula and Maniplus scripts. A disadvantage is that these types are hard-coded and changing them usually means changing all involved data models. That in turn means that all Manipula/Maniplus scripts and boi files that use these data models have to be recompiled too.
- Lists are also used extensively in the survey. For instance the 700 different service charges mentioned in the last section have to be categorised into 13 groups. These lists have to be maintained which is typically done in the following manner. The list is put in a table, which is accessed through a boi file. A screen is built in Maniplus in which a lookup presents the list. A user can select an item from the list and subsequently edit this item, either directly through a control and buttons or by opening a DEP to edit the entry.

4. Using the Blaise Suite

In the previous chapter we have presented a large system build in the Blaise Suite (Blaise, Manipula and Maniplus). To build this system we have used many parts of the Suite extensively and in some parts reached the boundaries of the Suite. In this chapter our experiences with the main components of the Suite are described.

4.1 Experiences using Blaise

Blaise was used to generate the DEP's for the main data sources as well as a few DEP's for the editing of classifications. In general our experiences with Blaise were positive. Defining data models is rather straightforward and self defined types are a very powerful tool. In particular the property of Blaise (and Maniplus) that values outside the type definition are automatically blocked is very convenient. For the DEP many automatic properties are set and many settings may be refined in the mode library (bml). Finally, the rules, which are integrated in the forms, are a powerful tool of checking the validity of the input. All these properties make Blaise a very powerful tool for building survey forms.

A less attractive property of Blaise is that the layout of the DEP is divided over two sections, the order of appearance is defined in the Rules, while their aspects are defined in the layout section. Moreover, four files are used to define the DEP, the bla, which contains the variables and the aforementioned sections, the modelib, which allows customization of the variable and page makeup, the menu file (BMF), which allows customization of the menu bar and the datamodelproperties file, which is used to define some of the properties of categorical variables. This separation of the code makes the development of a complex DEP somewhat of a struggle. In fact for highly customized and interactive designs Blaise can be as time-consuming to use as any normal programming language. This is especially the case if a reasonably small number of variables is used in a complicated design, as was the case in the variable data DEP.

Another problem with Blaise is that including external files into the data model can be rather complicated, a fact that is made more troublesome by the fact that Blaise has no debugger, as opposed to Manipula and Maniplus. Moreover, external tables, which are imported through a boi file into a DEP are completely put into memory even though only one of the external records is needed. For our project this meant that by opening one record in the variable data required opening the full panel and respondent data. This resulted in an unacceptable performance loss. The solution was to gather into a separate table all relevant external data for a record, data which is already opened in the main editing window. This table was then used as an external within the DEP, which improved the performance to an acceptable level.

Finally, if the data model is used in combination with a boi file, as we did, changing rules may invalidate the boi link. For this problem, however, there is a solution: reservechecks may be used as 'false' rules, but they require a strict bookkeeping.

4.2 Experiences using Manipula

Manipula was used in our project for calculating the index, for the data I/O and also for various smaller functions, mostly in cooperation with Maniplus. We found Manipula to be a versatile tool, which can access a wide range of file types, especially through the boi files. Moreover, the property that Manipula automatically links input and output variables of the same name, reduces the number of lines of code. The aggregation (sort section) is unusual as it combines ordering and aggregation of data, but it works and is rather fast.

However, we found that it had three main limitations, all associated with boi files. Firstly, it was found that joining data from two tables, either bdb or boi files, is very slow. For text files this problem doesn't exist, but for the sake of data integrity databases were preferred. In our system data joining was often needed, which made this a big problem. The solution was that we generated a number of views, which performed the necessary joining within the database and which we could access through new boi files. The resulting performance was much better.

The second limitation was that while reading data through a boi file has a decent speed, writing data to a database table is very slow. This problem was acute for the importing of data, which is performed daily and could take up to two hours. We haven't found a workaround to improve the performance. The import is now performed at night through a batch job.

The third problem was that, while the boi files allow version control on records, this capability makes database interaction even slower. This also affected the main editing window and the DEP's, all of which need to interact with the database. In the end, to maintain a decent performance for our system, version control was dropped. We decided to use daily version control by making a backup of the main data tables each night before the new import.

4.3 Experiences using Maniplus

Maniplus was used for defining the main menu and the different editing screens. A Manipula menu defines a menu bar onto the main window. It may adopt different forms, depending on the operational rights of the user. The advantages of using Maniplus are that it can communicate with Manipula and Blaise. Moreover it is relatively simple to set up. The main disadvantages are that it is somewhat limited and doesn't allow a full integration with other screens, like office applications, where the menu bar is always shown in all opened windows..

All screens, except for the DEP screens are in effect graphical user interfaces, which in Maniplus are called Dialogboxes. Four types of objects may be defined within a dialogbox, buttons, texts, controls and lookups. Buttons and texts are functional, but not special when compared to other GUI builders. Controls are ways of presenting and editing variables and as such may take different forms (radiobuttons, dropdowns, editfields). They are a robust way of accessing data and actually force the user to use values within the type definition. The lookup is a good way of presenting list. They automatically align the data and return the primary key of the selected record.

The main problems with Maniplus screens are that most properties of the elements are hardcoded. No relative positioning may be used and no wildcards may be used for the other properties such as colouring. This can make setting up and changing a window rather time consuming. Also, refreshing Maniplus windows is difficult. At the beginning of the project the Evaluateonchange setting did not work. A new build was rolled out by the Blaise team, fixing this problem. However, this property remains somewhat difficult in use and might slow down an application. We found that a good way of refreshing a window is performing a (dummy) writing action on the underlying databases. This triggers a refresh of the values of that source and it can be implemented in a refresh button.

5. Conclusions

In this project we have build a complex data processing tool, which uses the Blaise Suite in an unusual manner. While Blaise has been designed to perform data collections for surveys, we have elected to only use it for the editing of separately collected data. Thus, we did not use one of the primary strengths of Blaise, interactively collecting data, while using rules to force only correct responses. We did find that it is certainly possible to perform data editing in Blaise. However, given the size of our data set and the fact that we had to use relational databases for storing our data. As a consequence the performance of the system was a major issue. Moreover, the complexity of the editing process, especially the work distribution and the classification management meant that we had to build much functionality in Maniplus. It did support these functions, but for this purpose it is not a particularly powerful tool.

Even though we did have some problems using the Blaise Suite, the support team was always very helpful. We usually contacted them through the normal Blaise support e-mail and they usually came back to us within a day. This aided significantly in making this project a success.

Concluding we can say that if most of the data collection is performed in Blaise, the last bit of data editing can very well be done in this tool too. It is a very good package for quickly building those types of tools. However, if the data collection is not done in Blaise and the complexity of the required solution becomes higher, Blaise is not the ideal tool for further data processing.

Post-Collection Processing with Blaise in a Distributed Environment

Mary Laidlaw, Mike Rhoads and Jane Shepherd, Westat

1.0. Introduction

The processing, quality review, and delivery of survey data for a recent longitudinal study was particularly challenging due to the need to support multiple users in a distributed processing environment with highly structured levels of data access. As on most longitudinal studies, the need to re-field data swiftly to drive the subsequent protocol for each participant competed against the need to deliver clean and consistent data for analyses. Our data processing solution had to meet both requirements simultaneously. In addition, the data that required post-processing was collected in multiple modes: Blaise CAPI, Blaise CATI, and hardcopy forms scanned through data capture software. Reviewing the data with an eye towards consistency between instruments was essential due to the need to determine the appropriate next steps in the specific participant's protocol.

In order to meet the needs of this and other Blaise projects, we enhanced and integrated a set of Blaise-based tools that could be used for a variety of data processing tasks at multiple locations. This paper describes our experience with the implementation of the Blaise Data Processing and Quality Assurance system (BDPQA) we developed. We discuss: the initial requirements of the projects; the system and database architecture; the flow of data to and through the system; and the view of the tool from the users' perspectives. We conclude with some lessons learned from our experience.

2.0. The Post-Processing Requirements

Westat has used Blaise post-processing tools for many years. In recent years, a number of projects have required more extensive post-collection processing. Some of the requirements addressed by BDPQA include:

- The division of collection and editing responsibilities between a central office and distributed sites, each of which was responsible for the quality of its collected data.

- Multiple views of the data so that staff at the distributed sites could access only the data for that site's cases. The staff at the central office could access all collected data.

- The sequencing and division of post-collection processing tasks to be performed by the sites and the central office. For example, while interviewer field comments were most appropriately reviewed and resolved by the sites which were responsible for that staff, coding of text responses was conducted by the central office in order to ensure project-wide data consistency.

- Two levels of review – the editor and the supervisor – with appropriate task authorization in order to ensure high data quality and the validity of updates.

- Documentation of all data decisions across data collection sites and the central office in a comprehensive Data Decision Log. Again, the site staff could view only the decisions that pertained to the site's cases. The central office staff would need to view the decisions across all centers.

- Integration of data processing for hardcopy form data. All editing of data collected via hardcopy would need to occur during post-data collection processing and would be documented in the data decision logs along with decisions about CAPI and CATI data.

- Easy update of the post-processing tool as new versions of hardcopy forms as well as the Blaise instruments and their data models were fielded. It was also required that the tool be adaptable to any new instruments added to the study protocol.

- An efficient flow through all post-processing steps at both the sites and the central office with ongoing support for all instruments. Because the progress of the protocol for each respondent depended on the likelihood and timing of medical events, the sequence of data collection would result in a flow of data from all instruments into the study's systems on an ongoing basis; there was no specific end date for the use of any instrument and the start up of the next phase of the study. Collected data was expected to complete processing by both the site and central office within three to four days of collection.

- Capability for a wide range of ad hoc and predefined reports to be used for multiple workflow and analytic purposes including evaluating the flow of the data through the system, the consistency of coding, the status of each case's data across instruments at any one time, etc.

- The processing approach needed to manage the storage of the original data file (data as collected and passed to any post-process) and all other data files generated as a result of BDPQA activities. All file versions needed to be immediately accessible for a period of at least one year in case of the need for additional processing.

- The process and flow of each instrument's data for each case (referred to as a production unit throughout this paper) must be independent of all others. To accommodate a variety of protocols for each household and respondent, the existence of one instrument in the BDPQA did not necessarily require the existence of another instrument in the system at the same time.

Given these requirements, Westat leveraged an in-house Blaise editing system as the base for a new distributed approach, BDPQA. The editing process within BDPQA centers on the review and resolution of decision log entries. Decision log entries are created for field comments, Blaise edits, and reported data issues. This paper addresses our experience with the initial version of the distributed processing BDPQA system.

3.0. BDPQA System Architecture

The BDPQA system consists of two primary components: a server-based batch process called Workflow Manager that manages the flow of data within the system, and a Windows desktop interactive application that is used by editors and verifiers. There are also two administrative tools. One of these (BES Setup) is used to generate an initial instance of the system and to manage the deployment of system updates, while the other (Manage Users) is used to control user access and roles within the system.

All system components are primarily written in VB.NET. The system makes use of numerous Blaise capabilities, including the Data Entry Program (DEP), Manipula, Blaise Datalink, and the Blaise Component Pack (BCP). We use Microsoft SQL Server 2005 for data storage, and utilize SQL Server triggers and stored procedures to implement such functions as change auditing. In addition to these primary technologies, BDPQA uses various COTS software to perform certain functions. These include Microsoft Access (generating reports), eDocPrinter PDF Pro (converting reports to PDF format), Adobe Reader (viewing PDF documents), and Microsoft Word Viewer (viewing audit trails).

Since the system was designed for access by widely distributed site-based staff, we thought about whether we should use Blaise Internet as the basis for the user interface component. We decided against this option for two primary reasons. First, as we have discussed in previous papers (Allan, et al., 2001; Dulaney and Allan 2001; Gowen and Clark 2007; and Frey and Rhoads 2009), Westat has evolved an in-house Blaise editing system over many years, and we wanted to take as much advantage as possible of that desktop-based user interface code. Second, while we have used Blaise Internet for several interviewing projects, we did not have any experience with it for data editing applications. Although we certainly did not rule out the possibility that Blaise Internet could be used as a suitable case review and updating tool for data editors, we did not have nearly enough time for research and development in this area given the tight time pressures of the initial project.

Since we decided to stay with a desktop client rather than implementing a Blaise Internet solution, we needed an alternate way to make the system available to site-based staff. Fortunately, Westat had already implemented a highly-secure Citrix platform for other project purposes. We were thus able to implement the user interface component as an application on the Citrix desktop, which was then available to authorized site-based and home office staff.

3.1 Workflow Manager Component

The Workflow Manager component of the BDPQA application controls the flow of all data into and within the system. It is a nightly batch process, although it can also be run on-demand if special needs arise. Workflow Manager performs the following tasks:

1. Loads all newly completed Blaise interview data into interview data tables in the site-level SQL Server database

2. For all new cases, correctly names (by instrument and ID) and loads all acceptable audit trails into the system.

3. Updates the site's decision log table by going through all non-closed out cases in the interview data site tables and adding an entry for:

    a. all new interviewer comments found by invoking the GetRemarks method from the Blaise API

    b. all new errors found by invoking the Blaise API's CheckRecord method

4. Loads the interview data from all newly closed out site office cases into interview data tables in the central office SQL Server database

5. Loads the decision log site office table entries for all newly closed out site office cases into the decision log table in the central office database

6. Updates the central office decision log table by going through all non-closed out cases in the interview data central office tables and adding an entry for all new errors found by invoking the Blaise API's CheckRecord method

All of the data loading steps listed above are performed by Manipula programs, which take advantage of Blaise capabilities for accessing and storing data in multiple formats—native Blaise, SQL Server, and XML (which is used to bring in records from the data capture system). Workflow Manager also uses

Blaise API routines to extract interviewer remarks and to identify any violations of Blaise editing rules in incoming cases.

3.2 User Interface Component

Section 5 below describes the interactive component of BDPQA from a user perspective in terms of its look and feel and its functionality. From a systems perspective, an essential element of this component is its integration with Blaise through the use of the Blaise Component Pack (BCP). The system invokes the Blaise DEP both for data updating and for data browsing. The use of DEP ensures that the user of the system cannot make any changes to the data that violate the rules of the data model.

The user interface component also interacts with the SQL Server database tables that Blaise uses to store its data. We set up an insert trigger on each of these tables, which is activated whenever a data change is made (causing a new row to be inserted into the table). The trigger then invokes a stored procedure that compares "before" and "after" values to identify which specific data items were changed. For each difference that is found, we insert a new row into a separate table that includes the case ID, variable name, and newly-assigned value.

3.3 Data Storage

All data within BDPQA is stored in Microsoft SQL Server. Typically three separate databases are used: one for each of the two phases of editing (site and central office), and a third database that contains various types of metadata that are used by the system. This is configurable at the time an instance of the system is created, so that all of this information could be stored within a single database if desired.

Within the site and central office databases, some of the tables are effectively "owned" by the Blaise Datalink component. Workflow Manager uses Blaise to initially load case data into the system, and the Data Entry Program that is invoked from the interactive component of the system uses the data in these tables to replay and update cases. These tables are read by non-Blaise modules of the system for report generation and other purposes, but they are written to only by Blaise. Blaise Datalink offers a number of data storage options, which it refers to as data partition types, and it also provides a capability it refers to as "generic" storage. For BDPQA, we use the Flat Blocks data partition type. We also selected the generic data storage option so that we could take advantage of versioning. This data storage structure is described in much more detail in an earlier paper (Frey and Rhoads, 2009).

In addition to the tables used by Blaise Datalink, the databases also contain numerous other tables that are managed by the system entirely outside of Blaise. These include tables that are used to store and manage decision log entries, maintain user information and log system access, and hold various items of metadata that are used by the BDPQA system.

3.4 Access Control and Security

The BDPQA implements a highly granular user access control model. Access is controlled by site, by instrument, and by function. For example, some users may be provided with read-only access so that they can view case data but not change any values, and only designated users may be allowed to view Blaise audit trails. User access information is stored in SQL Server tables, and there is an interactive interface module (Manage Users) that allows the system administrator to add and remove users and to modify access rights. System users can access the SQL Server database only through the interactive component of the system, which enforces the access constraints described above. As an additional precaution, the case data from each site is segregated into a separate schema and set of tables.

The BDPQA can operate within an overall environment that provides additional security measures. For instance, the system can be reached only through a Virtual Private Network (VPN), and access to it requires the use of two-factor authentication.

3.5 Configuration and Customization

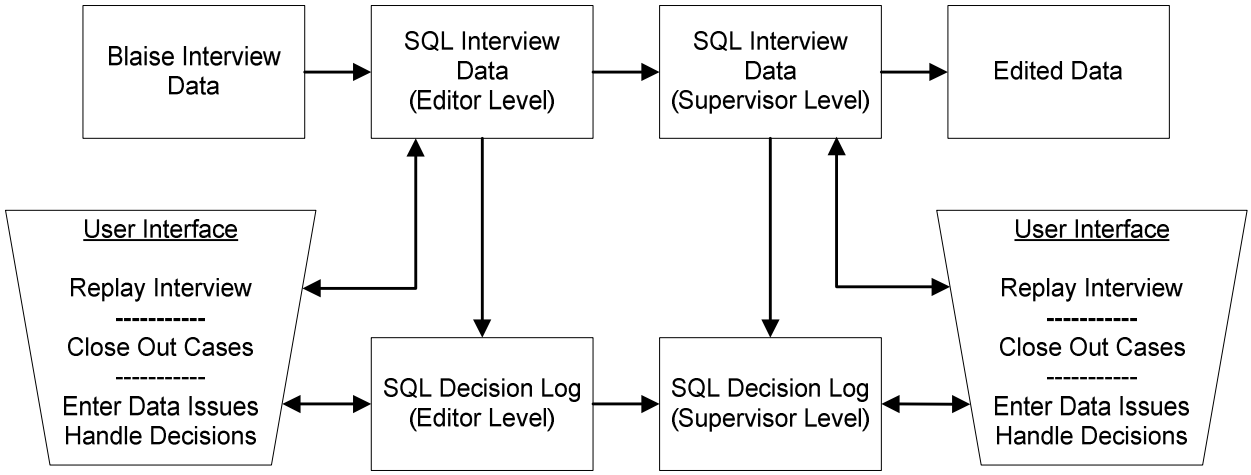
BDPQA includes a setup tool that is used to generate an initial instance of the system and to manage the deployment of system updates. This tool, combined with a set of initialization (INI) files, provides for identifying the necessary network data locations, SQL Server databases, Blaise data models, and sites. The primary function of this module is to create the necessary schemas, metadata, BOI files, instrument tables, and other objects within the databases.

Since the BDPQA is a corporate system intended for use on multiple projects, it was designed to be highly customizable. This is achieved through a combination of INI files and metadata tables within SQL Server; as much of the system logic as possible is implemented through table-driven programming. Among the aspects of the system that can be customized for each project are the following:

- Verifier Role – Whether the user who edits a decision log entry can also verify the same entry
- Edit Check Exclusion – Which edit checks, if any, should not produce decision log entries
- Critical Data Items – Which fields, if any, should be included in or excluded from the Critical Item report
- NonDeliverable Items – Which fields, if any, do not need to be included in deliverables
- Watch Window – Whether the Blaise watch window should appear on the screen during replay
- Field Audit Trails – Whether field audit trails will be provided to system users
- Name to be used for instrument (form) in UI screens and report headings
- Name to be used for study in UI screens and report headings
- Internal security – List of the login IDs and roles of all individual users

4.0. The Data Flow within BDPQA

As indicated above, the Workflow Manager module of the BDPQA ran daily to pick up the new production units transmitted from the field and made them accessible to the editor (site office) level of the BDPQA. Once the site had completed all work on the production unit, the Workflow Manager moved the closed production unit into the database accessible to the Supervisor (or central office) view. This movement is illustrated in the following diagram:



The diagram illustrates the centrality of the Decision Logs to the movement of the data through the system. Because the BDPQA centers on the decisions, a case cannot be closed out of the system – either at the site (editor) or central office (supervisor) level - until all decision log entries have been reviewed and resolved. This review and resolution process normally involves the following four steps for each decision log entry:

1. Review and resolution by an editor level editor
2. Review and resolution by an editor level verifier
3. Review and resolution by a supervisor level editor
4. Review and resolution by a supervisor level verifier

The resolution at any step overrides the resolution made at any previous step; this gives the supervisor level verifier the final say. There are some coding-related and other circumstances in which a decision log entry is not generated until the production unit reaches the supervisor level. For these entries, the review and resolution process is limited to only the last two steps.

5.0. The Users' Perspective: Two Views of the Tool

The overall approach for the initial project's data processing was to give each site office primary responsibility for the accuracy and completeness of its collected data and the home office responsibility for the consistency of the processing of the data across all sites. Therefore, in terms of post-processing division of labor, the site offices were to perform the following tasks:

- Documenting in the system any issues reported to the office that might affect case data;
- Reviewing comments entered into the CAPI and CATI instruments;
- Updating the case data, if necessary, in accordance with issues and comments;
- Running the edits programmed into the Blaise data models to check that the transmitted and edited data was recorded as expected by Blaise;

- Reviewing case data as needed to support inquiries from the field or to validate interviewers' work; and

- Releasing the data to the home office for finalization.

For the initial study, Westat served as the central office. Westat's responsibilities for post-processing included:

- Establishing and documenting a consistent body of editing rules (based on situations encountered) to be implemented by the site offices;

- Reviewing all data updates and data decisions made by the site offices to ensure standardization of data processing across the study;

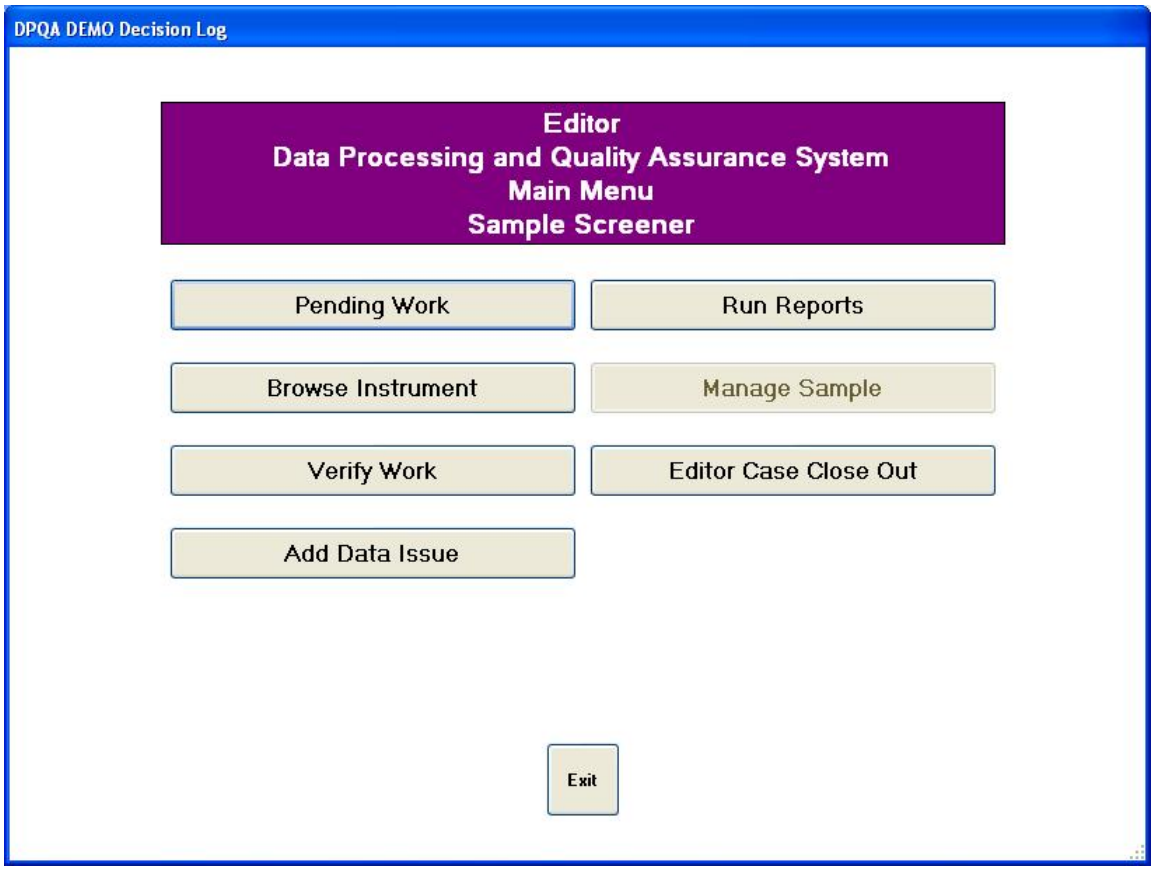
- Coding text responses as required by the study;

- Reviewing frequencies and cross tabulations to identify unusual data patterns; and

- Finalizing the data and corresponding documentation for delivery in preparation for analysis.

5.1 The Users' View: Editor Level

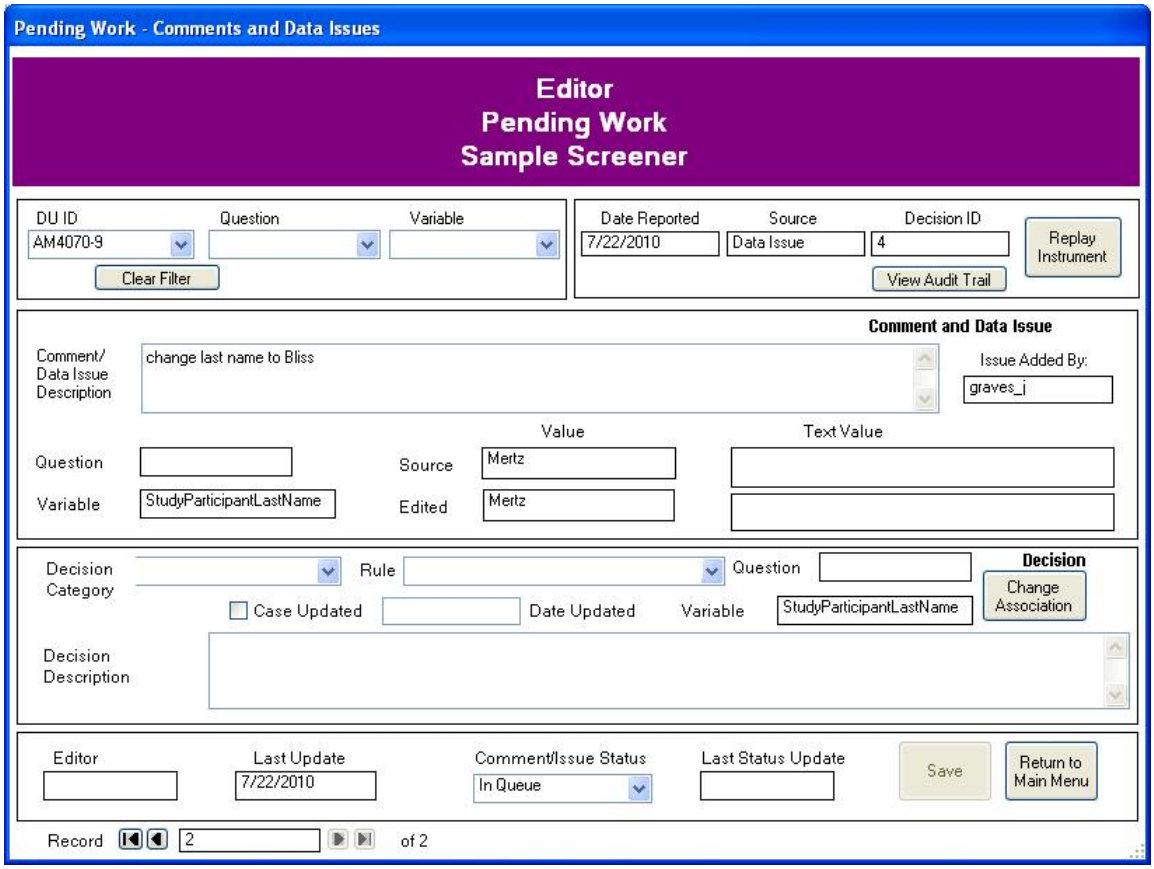
It was intended that the site offices would use the editor view of BDPQA. After the user signed in to BDPQA, and connection and loading were complete, the user selected the instrument to be worked. Regardless of the instrument selected, BDPQA users at each site office were limited to the data for their PSU and role only. We worked with the sites to ensure that only appropriate staff had only the access they needed. After the instrument was selected, the system displayed the BDPQA Editor Main Menu.



As you can see on the illustration, the BDPQA Editor Main Menu presented options in the order of expected workflow. (For this version of the tool, Manage Sample was not active, because this function was not required for the initial phases of the study.) Within the editor view, BDPQA site office users could select:

- **Pending Work** to review, resolve, and document in the Decision Log all field interviewer comments, post-data collection data issue reports, and edit checks;
- **Browse Instrument** to replay the instrument in read-only mode;
- **Verify Work** to review and verify data decisions at the 100% level, to run the reports that could assist in the verification of work, and to update the status of the data decisions to close them out individually;
- **Add Data Issue** to enter into the decision log any data issues as reported to field supervisors and data managers. The Add Data Issue screen functioned as an electronic template and all fields were required so that issue documentation would be consistent and interpretable;
- **Run Reports** to run the programmed monitoring reports to track and supervise workflow through data processing. The user had the option of viewing the report on the screen, saving the report in PDF format or printing the report; and
- **Editor Case Close Out** to release the production unit to the next level of processing – the supervisor level.

The core of the system is the Pending Work module, which allows users to review the details of a field comment, data issue, or edit check, and to document their resolution in the decision log.



The Pending Work screen has three main sections. The items displayed in the Heading section include:

- **Subject ID (here, a Dwelling Unit or DU ID)** with which the decision log entry is associated.
- **Question and Variable fields** which act as filters. Filters can identify all work related to a particular question or variable.
- The **Source** of the issue; this could be a Field Comment, an Added Data Issue or an Edit Check failure.
- The **Replay Instrument** button takes the user to the instrument in data collection mode to review and update data.

The next screen area is the Comment/Data Issue section and displays:

- **Question** and **Variable** with which the comment/issue is associated.
- **Source** - the original value for the variable.
- **Edited** - the current value. If the data has not been changed within BDPQA, the Source and Edited values will be the same.

The lower portion of the screen is the Decision section and includes:

- The **Change Association** button is used to change the Question and Variable with which the comment or issue is associated. Often in the field, an interviewer will need to enter a comment at a later point that does not directly relate to the relevant question. In order to include the most relevant associations in the data decision log for reporting and sorting purposes, this functionality was included in the processing options.

- The **Decision Category** is initially blank. It is used to indicate the level of action (or inaction) required to resolve the issue. The selections include Leave As Is, Action Required, or Supervisor Review. It should be noted that if Decision Category is set to Supervisor Review, the Comment/Issue Status of the issue will remain "In Queue" and cannot be changed. A verifier will have to look at the issue before it can be resolved and the status updated.

- **Rule** indicates the standard rule that was followed to resolve the issue and update the data, if required. It was intended that the set of rules would build throughout the course of the study as a variety of data issues were encountered.

- **Decision Description** is the detailed description of the action that was taken.

- The **Case Updated** box and the **Date Updated** field are related. If the box is checked by the user to indicate a data change, the system fills the date.

- The system populates **Editor** with the current user's ID when the user saves work and auto-fills the date in **Last Update.**

- **Comment/Issue Status** describes the point to which the issue has been worked. Status begins as "In Queue." Once the user completes work on the comment/issue/edit, he or she changes Status to "Ready for Verification" and the issue moves to the Verify Work queue. The system auto-populates **Last Status Update** whenever the Status is set.

- **Save** is activated, not grayed out, when the user has updated any of the information in the Decision section of the screen.

The **Verify Work** screen functions in the same way as the Pending Work screen; however, the screen includes a Verifier field (which is automatically filled in when the Verifier saves his/her work – this field functions like the Editor field in Pending Work) and the Comment/Issue Status selection is "Ready for Verification" and "Complete." When all comments/issues/edits (decision log entries for the production unit are "Complete," the unit is available in Case Close Out. Units are available for close out only when all associated decisions have a status of Complete. If a unit in BDPQA had no field comments or data issues and has passed all editing and QA/QC reporting, it is immediately available for close out. Once a unit is closed out at the editor level, it was no longer available for update within the individual site's version of BDPQA.

5.2 The Users' View: Supervisor Level

Once a case is Closed Out at the editor or site office level, it is available to the central office or supervisor level. The options at the supervisor level are similar to those at the editor level but are arranged a bit

differently based on typical workflow at the central office. Here, Run Reports appeared first, followed by **Add Data Issue**, **Pending Work**, **Edit Case**--which allows changes to the instrument data; in effect, what "Replay Instrument" was to the editor, **Verify Work** and central office **Case Close Out**. Running reports was particularly important to the central office because, in addition to supporting the same functions as the reports at the editor level, reports were also necessary for the coding function to be performed by the central office only. Due to the need for coding of text responses, the central office view of the BDPQA relied on an updated data model with additional fields for coded values.

6.0. Lessons Learned

It is advisable when designing a Blaise instrument that will be used for by BDPQA to avoid using within the data model:

- Time stamps that depend on today's date and/or time
- Sampling criteria that can be changed by data value changes
- Auxfields that can give the impression of a non-existent error during replay (this can happen with auxfields, as opposed to fields, because their values are not stored in the database)
- Walls within the instrument that prevent interviewers from returning to previously-answered questions
- Field names that conflict with reserved words in SQL Server

For existing instruments with any of the above problematic features, workarounds can usually be devised. Within the context of the decision log and various reports, Blaise fields are identified by a variable name (the actual Blaise field name) and a question name (the tag value associated with the field in the Blaise data model). For this reason, tags should be carefully chosen. You may want a unique clearly-named tag for each field and possibly shared tags for related fields.

In addition to paying attention to tags when programming the original Blaise data model, you are also advised to provide unambiguous wording with each potentially complicated edit check. When no explicit wording is programmed with an edit check, Blaise uses by default the relevant programming logic. This programming logic can be confusing and will appear by default in a decision log entry if no explicit wording has been programmed.

For field data collected via hardcopy forms and stored in a SQL table, an appropriate Blaise data model must be produced. This can be done using the Blaise OLE DB Toolbox to produce a data model from the SQL table and subsequently adding question text and tags.

7.0. Conclusion

The BDPQA met the challenges of this longitudinal, multi-mode, and distributed data collection effort. By leveraging the Blaise metadata available through the instrument authoring process and customizing our existing in-house tool to meet the requirements enumerated at the start of this paper, we were able to ensure that secure and high quality data processing ran right on the heels of data collection. The tool can be adapted for use on other projects in which raw field data are stored in either Blaise or SQL Server databases. The system edits and cleans the data using a two-level review process.

BDPQA offered a number of advantages in addition to those that accrue from using the superb editing capability offered by Blaise. These advantages include:

- Automatic versioning that can be used to ascertain for each production unit the date of any data changes and the values stored in the database at any given date.
- Ability to store and retrieve field audit trails for each production unit.
- Ability to generate a wide variety of reports
- Ability to provide internal security that identifies users and limits their roles
- Ability to use different data models at the editor and supervisor levels. This can support an approach whereby coding or other tasks can occur only at one of the two levels.
- Ability to provide decision log information either electronically or in hardcopy format to meet standard client requirements.

8.0. References

Allan, Boris, O'Reagan, Kathleen, and Lohr, Bryan. "Dynamic ACASI in the Field: Managing All the Pieces." *Proceedings of the 7th International Blaise Users Conference.* September 2001.

Dulaney, R. and Allan, B. "A Blaise Editing System at Westat." *Proceedings of the 7th International Blaise Users Conference.* September 2001.

Frey, R. and Rhoads, M. "Blaise Editing Rules + Relational Data Storage = Best of Both Worlds?" *Proceedings of the 12th International Blaise Users Conference.* June 2009.

Gowen, L. and Clark, P. "Lifecycle Processes to Insure Quality of Blaise Interview Data." *Proceedings of the 11th International Blaise Users Conference.* September 2007.

Validation of Survey Data in Xml

Leif Bochis Madsen, Statistics Denmark[6]

Abstract

In a number of surveys conducted by Statistics Denmark all or part of the data collection is carried out outside the office and delivered in some kind of electronic format.

Increasingly, xml is used as format of delivery and this offers some opportunities to push validation of the data towards the supplier. For example, an Xml Schema may be used by the data supplier in order to assure the correct format before submitting the data.

However, other xml technologies are available which provide different means of validating the data in even more sophisticated ways.

The paper will discuss some technologies available and the possible role of Blaise.

Introduction

In a paper written for the IBUC/2007[7] an outsourcing project for the Danish Labour Force Survey (LFS) is described. The requirements and architecture of this project is fully described in this paper and I shall only summarize briefly.

Due to legal causes Statistics Denmark could not require the use of specific software – i.e. Blaise – by the organizations offering tenders for this project. Therefore - among other aspects – the outsourcing project involved specification of a data exchange format and efforts made to check validity of the data transferred from the organization responsible for conducting interviews to Statistics Denmark. The format chosen was xml and the work carried out in Statistics Denmark comprised construction of software capable of validating the xml data and importing the data into Blaise[8].

The validation was split into two parts. The first part consisted of checking the conformance to an xml schema and was the responsibility of the supplier of data. The second part was implemented as a Blaise data editing instrument and carried out inside Statistics Denmark.

After three years of operation some problems in this setup have been identified and ideas of improving the data exchange have occurred.

The given conditions are:

1. Data must be exchanged in xml format.
2. It is not possible to require external data providers to use non-standardized software (i.e. Blaise)

[6] The opinions and assessments stated in this article are those of the author and do not necessarily reflect opinions and assessments of Statistics Denmark.

[7] See [LBM07]

[8] This project was carried out before the Blaise Xml format was published in Blaise 4.8. A lot of work could have been avoided if the project had been carried out one year later, but that's life!

Problems

Different survey software offers different sets of properties and possibilities. For example, in Blaise it is easy to describe rules comprising a computation of dates – e.g., the day six months earlier than today – and using this computed date in a condition – e.g., comparing this day with the respondent's start of job – and this way decide which questions to ask next. This kind of computation was not possible to carry out in the software used by the interviewing organization. Following this, some forms that were correctly filled in the interview software turned out to be faultý in the Blaise editing instrument.

Another issue is the treatment of warnings. It will not make sense in post-editing to examine – or re-examine – all warnings that occurred and may have been suppressed under interviewing. However, a statistical treatment of the remaining warning conditions may be relevant in order to identify, for example, weak parts of the questionnaire. This statistical treatment is not straight-forward to carry out using Blaise tools.

Also, during the three years of operating the LFS a number of misunderstandings concerning routing etc. has been discovered. It is an assumption that there is a potential for improved documentation of the survey and better communication between the supplier and receiver of interview data[9]. This requires a media that may be used for exchange of metadata.

Eventually, there is a wish to push more validation towards the supplier in order to discover possible discrepancies at an earlier stage.

Overall, the project has emphasized the need for platform independent tools for validation and formal description of routing, checks and warnings.

Tools for Validation

In a thesis[10] written at the IT University of Copenhagen, a study of different tools for validation of xml data has been carried out using the LFS outsourcing project as a case study.

The study examines a number of available tools like Xml Schema, RELAX NG[11] and Schematron[12], that all support some kind of 'co-occurrence constraints'.[13] The two most comprehensive of the tools are Xml Schema and Schematron and they will both be briefly discussed here.

Both are so-called schema languages, i.e. languages that describe an xml document type and can be used in order to test whether a particular xml document conforms to its definition.

Xml Schema is a grammar-based language which means that a grammar describes the structure and contents of a document. Validation implies that errors are reported if the document does not conform to

[9] [LBM09], p. 46.

[10] This paragraph is based entirely on [LBM09].

[11] RELAX NG home page, URL: http://www.relaxng.org

[12] Schematron. A language for making assertions on patterns found in xml documents, URL: http://www.schematron.com/

[13] The term 'co-occurrence constraints' here denotes the set of routing, check and warning constraints in Blaise terminology, i.e. the rules.

the grammar. From version 1.1 – released autumn 2009 – Xml Schema incorporates so-called assertions that may describe co-occurrence constraints. However, it lacks some features that are widely used in Blaise rules like conditional computations, modularity and the distinction between errors and warnings.[14]

Schematron is a rule-based language which means that it is used to describe the rules that a particular xml document must conform to. Schematron supports validation of almost all types of routing and integrity constraints. However, it is not possible to change the contents of a document so the possibility to define default values and/or computed values is not present. Also, using the rules-based approach it is possible but not practical to validate structure and content.

Both of the languages use Xml Path (XPath) as query language for validation of co-occurrence constraints which suggests that the possible validations are fairly equal. To a certain extent they are, but in Xml Schema there are a number of limitations that make it almost impossible to use a modular approach. For example, the XPath expressions are only allowed to query downwards in the hierarchy and it is not possible to declare local variables to hold intermediate results. In Schematron there is no limitation on the axes used in XPath expressions and it is possible to declare local variables that may be used to represent, for example, parameters and auxfields as they are known in Blaise.[15]

Compared to Blaise, the grammar-based approach (Xml Schema) is well suited for validating the structures, i.e. fields, types and blocks while the rules-based approach (Schematron) is better suited for validating the rules.[16]

Finally, in order to produce reports Schematron provides the best possibilities. The generation of human-readable error messages to be used in a production environment is fully supported.[17]

A possible setup

In [LBM09] it is recommended to divide the validation process into three levels:

1. Well-formedness (syntax) is the first level of validation and is equally determined for any xml document. Checking the well-formedness – and character set – of a document is a basic task and requires no knowledge on the purpose of the document and its contents.
2. Structure (semantics) is the next level of validation and verifies that a document conforms to a given schema. Violation of the structure is a sign of errors in the generation of the xml document.
3. Constraints (pragmatics) may be violated because of inconsistencies in the collected data. For example, caused by a routing error in an instrument used for interviewing or caused by an interviewer ignoring a warning from the instrument and carrying on interviewing, thus creating an inconsistent form. [LBM09, p. 47]

The three levels imply three different strategies – and may be supported by different tools. The first level is supported by any xml-aware system; the second may be supported by an Xml Schema definition and the third possibly by a Schematron definition.

[14] Ibid., p. 30-32.

[15] The languages are compared in ibid., p. 42-44. The conclusions about the use of XPath in Xml Schema 1.1 also refer to later studies, because there was no implementation available at the time of writing the thesis.

[16] [LBM09] suggests using RELAX NG as the language for validation of structure and content. Partly because RELAX NG and Schematron are standardized in the same suite as complementary tools, partly because RELAX NG provides a representational (human readable) language as well as its xml equivalent.

[17] Ibid., p. 43-44.

The next step is the question of how to achieve this scheme. As it is also concluded:

> "None of the languages, however, are particularly user-friendly. (..) (And) none of them compares to the expressiveness of the Blaise language. Schema languages in xml syntax are in their wordiness and unreadability userunfriendly by definition while xml is still crucial for exchanging the schemas across platforms." [Ibid., p.48]

Therefore an obvious solution would be to generate these definitions from a Blaise source.

From Blaise 4.8 generation of an Xml Schema definition is part of the Blaise package which makes this part simple, but even earlier versions made it possible to generate an Xml Schema definition with the help of Cameleon.[18]

Unfortunately, Cameleon is not capable of handling the rules of a data model why it is necessary to use the Blaise API in order to generate a Schematron definition comprising the rules of the data model.

Schematron examples

Looking at a few examples may clarify the job. The first example shows a Schematron definition that reports the number of records in a Blaise xml document:

```
<rule context="/Database">
  <let name="AntalRecords" value="count(./Datarecord)"/>
  <report test="true()">
    Validating <value-of select="$AntalRecords"/> Datarecord-elements.
  </report>
</rule>
```

The **rule** element contains a set of computations, assertions and reports limited by the context which could refer to a block in a Blaise data model, but in this case refers to the document element of a Blaise xml document.

The **let** element declares a local variable identified by its name and a value retrieved using an XPath expression. The let element may be used to represent parameters, auxfields and locals in a block.

The **report** element is used to write a message to an output channel (e.g. a file) if the test expression yields **true** (which it always does in the example above). There is a quite similar **assert** element that will output a message if the test yields **false**. Asserts should be well known to Blaise users as it is the normal behaviour of checks and signals.

Let us look at another example[19]:

```
<rule context="//Job">
  <let name="currAge"
    value="ancestor::Datarecord/Person[1]/PersonAge"/>
  <assert test=
    "(year-from-date(current-date()) - (number(./YearOfJobStart))
```

[18] Examples are mentioned in [LBM07] and [LBM09], appendix B.

[19] From [LBM09], p. 28-29

```
      lt number($currAge)">
    You cannot have started your job before you were born!
  </assert>
</rule>
```

This rule applies to all blocks named Job (i.e. the block Job in all the datarecords in the dataset) , it defines a local variable which retrieves its value from a list of persons in the household. An assert element is used to check that the respondent has not been longer in his current job than his actual age allows.

The validation above may be retrieved from the similar rules in a Blaise data model:

```
FIELDS
   YearOfJobStart "When did you start in your current job?" : 1900..2010
AUXFIELDS
   currAge : 0..120
RULES
   currAge := Person[1].PersonAge
   YearOfJobStart.ASK
   CHECK
      (YEAR(SYSDATE) - YearOfJobStart) < currAge
         "You cannot have started your job before you were born!"
```

Future work

As the small examples above suggest the logic expressed in Schematron is not very far from the logic expressed in Blaise. Therefore, it appears practical to convert Blaise rules into a Schematron definition. Because Schematron is a standardized language, it should be possible to use this solution in order to push further validation towards the supplier of data leading to cleaner data when transferred. Also, Schematron may be used to produce reports on the number of suppressed warnings and thus fulfils most of requirements from the introduction.

Important is the ability to automatically generate Schematron definitions from the Blaise metadata. There still remains some work in this field.

References:

[LBM07]
Leif Bochis Madsen: Blaise and Xml: Experiences Of An Outsourcing Project, in: IBUC 2007 11th International Blaise Users Conference, URL: http://www.blaiseusers.org/2007/papers/Z4%20-%20Blaise%20and%20XML.pdf

[LBM09]
Leif Bochis Madsen: Definition of Validation Rules for Xml Data for Electronic Questionnaires, Unpublished thesis from the IT University of Copenhagen, 2009, temporarily available at URL: http://www.itu.dk/people/lbm/Validation_Rules_Thesis.pdf