Proceedings of the 9th International Blaise Users Conference

IBUC 2004

Gatineau, Québec
September 22 – 24, 2004

Recueil de la 9e conférence internationale des utilisateurs de BLAISE

IBUC 2004

Gatineau (Québec)
du 22 au 24 septembre 2004

Preface

This document contains the papers presented at the 9th International Blaise Users Conference held in Gatineau, Québec, from September 22nd – 24th, 2004. The conference included three days of technical papers and presentations on the use of Blaise and related topics.

The conference programme was planned and organized by the Scientific Committee of the International Blaise Users Group. The members of the committee included:

- Vesa Kuusela, Statistics Finland (chair)
- Leif Bochis, Statistics Denmark
- Bill Connett, University of Michigan, USA
- Sal Gara, Statistics Canada
- Lon Hofman, Statistics Netherlands
- Tony Manners, Office of National Statistics, UK
- Jim O'Reilly, Westat, USA
- Mark Pierzchala, Mathematica Policy Research, USA
- Pascal Riviére, INSEE, France
- Robert Wallace, U.S. Bureau of the Census

As the host organization, Statistics Canada has collected and printed these proceedings.

Préface

Ce document contient les communications présentées à la 9e conférence internationale des utilisateurs de Blaise qui s'est tenue à Gatineau (Québec), du 22 au 24 septembre 2004. La conférence de trois jours était axée sur des documents et des présentations techniques relatives à l'utilisation de Blaise et à des sujets connexes.

Le programme de la conférence a été préparé et monté par le Comité scientifique du groupe international des utilisateurs de BLAISE. Les membres du comité sont :

- Vesa Kuusela, Statistics Finland (président)
- Leif Bochis, Statistics Denmark
- Bill Connett, University of Michigan, É.-U.
- Sal Gara, Statistique Canada
- Lon Hofman, Statistics Netherlands
- Tony Manners, Office of National Statistics, RU
- Jim O'Reilly, Westat, É.-U.
- Mark Pierzchala, Mathematica Policy Research, É.-U.
- Pascal Rivière, INSEE, France
- Robert Wallace, U.S. Bureau of the Census

À titre d'organisme d'accueil, Statistique Canada a préparé et imprimé le compte rendu.

Table of Contents / Table des matières

Management of Cases and Interviewer Assignments

Using Blaise for Coding Listing Instruments 5

Utilisation de Blaise dans la répartition automatique des fiches adresse entre enquêteurs des enquêtes ménages 26

Contact History Instrument (CHI) 35

Michigan CATI Sample Management System (SMS) – Blaise CATI Enhanced 62

Application Development - Tools and Techniques

The Progamma Library of Blaise Routines 74

Understanding the Blaise Selective Checking Mechanism and the Appropriate Use of KEEP Statements 86

Instrument assembly, documentation and release 92

Using Arabic in Blaise 109

Automatic Coding and Code Generation

Codification automatique en ligne 120

Blaise Instrument Design for Automated Food Coding 129

Generating Blaise with Blaise 137

CodeBuilder2 – An Initial Coding Tool 145

Instrument Validation and Survey Integration

Generic System Interface Instrument 159

Randomly Generating Interviews to Validate a Complex Survey 181

Moving from 43 Generated Unique Instruments to One National Instrument 192

Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey 197

Audit Trail Analysis

Replaying Blaise Audit Trail Files for Data Verification 212

Replaying ADK Files for Testing Blaise Applications 221

Blaise AT Report System 240

Poster Sessions

Converting Translated Blaise System Texts To A New Version 251

USDA's Experience Using Blaise for Collecting Food Intake Data 258

Handling Post Processing of a Large Complex Blaise Instrument in a Continuing National Survey 259

Automatic Customization of Datasets Using Cameleon 260

Organisations Using Blaise

Error Reporting and User Interfaces for Post-collection Data Editing 269

The new German Microcensus starting in 2005, organisational and technological aspects, the use of Blaise 275

Multi Mode Surveys

Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey 283

System Implementation for a Blaise Multimode Web, CATI and Paper Survey 310

Mode effects and maintenance of data quality in a CAPI/CATI survey 321

Using CAWI in Household Surveys – Findings from Official German Statistics 331

Blaise and the Internet

Some Ideas On Securing And Processing A Blaise IS Survey 342

Exploring the capabilities of Blaise IS 347

Developing the On-line Milk Production Survey using Blaise IS 357

Documentation Tools

Programming Guidelines for Good Data Documentation 362

Blaise Documentation System 371

An evaluation of Delta, a documentation tool 389

New Directions

Recording Interview Sound Bites Through Blaise Instruments 411

Deploying Blaise to Tablet PCs for Mobile Use 424

Contacts 444

Using Blaise for Coding Listing Instruments

Malcolm Robert Wallace
Technologies Management Office
Census Bureau

1. Overview

The Census Bureau conducts many listing operations (surveys), using Computer Assisted Interviewing (CAI) instruments (listing instruments), as a means of obtaining sample for many of the surveys that it conducts. Conducting listing surveys is not flashy or exciting. It's a necessity of the survey industry to make sure that a complete sample is taken for the various surveys being conducted. It's a "behind the scenes" operation that doesn't get much appreciation. For the most part, there are no published results from the listing part of a survey. The listing instrument is the "tool" used to gather and, at times, select the sample person or unit. It then passes this information on to other systems or other instruments that will collect the interview data and collect all the "glory" associated with published results.

1.1 Paper Objectives

This paper will explain the differences between a listing "survey" instrument and a household/respondent (interview) survey instrument, review the different listing surveys conducted by the Census Bureau, go over unique functionality required for our listing surveys, discuss some of the challenges the Census Bureau has encountered with using Blaise for coding listing instruments, and review future plans for using Blaise for some of our existing listing applications.

1.2 What is a Listing Instrument and How is it Different from an Interviewing Instrument?

A listing instrument is a CAI instrument that is used as a tool by the Field Representatives (FRs) to collect and sample lists of data in order to generate sample cases for interviewing. These lists of data can be various things, such as lists of permits, addresses, or names.

In the past, a listing instrument was an instrument used by the FRs to only collect "lists" of data. A listing instrument was strictly designed to guide the FR through the listing process, to collect lists of data, and to sample these lists when the listing was complete. The FR did not "interview" anyone with these instruments. Listing instruments were designed this way partly because that's what was required and partly because of the limitations with the software used to create these instruments (CA Clipper).

However, with the movement to use Blaise for coding our listing instruments, the difference between a listing instrument and an interviewing instrument is no longer as obvious. Some of the newer listing surveys now require a respondent to answer some interviewing questions. These surveys still have a listing component that the FR must complete on their own. Regardless, there are some other differences between a listing survey and interviewing survey. Listing instruments (surveys) are typically used as an intermediate step in the overall survey process. The data collected by the listing instrument is used as input into an interviewing instrument or control systems either right after the listing is completed or in the future after headquarters has sampled it.

Another distinction of a listing instrument, is that the FR determines when the case (listing) is complete. With interviewing instruments, the respondent controls whether a case is complete by answering the questions and reaching the end of the interview. Once all questions are answered, the case is considered complete. Since the FR is conducting the listing without a respondent, they must determine when the listing is complete. Lastly, listing instruments tend to contain some unique functionality that is usually not contained in interviewing instruments.

1.3 Listing Instruments in the Past

In the past, the Census Bureau's current surveys used CA Clipper to code listing instruments. This was an excellent application to use for listing operations since it used an underlying database and so allowed for huge listings without pre-defining a maximum limit for the number of lines listed. It also allowed for complex sampling of the data that required sorting the records in the listing (database). Using Object Oriented Programming techniques, the same look and feel was applied to the various listing applications produced at the Census Bureau. The main downside to using CA Clipper was that it did not handle the asking of questions sufficiently. It also didn't produce any type of audit trail file that could be used to troubleshoot challenging field problems.

1.4 Current and Future Listing Instruments

About the same time the Census Bureau started to use Blaise for coding new CAI surveys, two new listing projects were started. It was decided to use Blaise for these listing projects as well. We have since coded a third listing instrument in Blaise and plan to convert three other listing applications in the near future. Although the differences between a listing "survey" instrument and a regular household/respondent survey instrument can be subtle, they do exist.

2. Listing Operations Conducted by the Census Bureau

As mentioned earlier, the Census Bureau conducts many listing operations as a means of obtaining sample for many of the current surveys it conducts. These listing operations vary greatly. Some require the FRs to key in ALL of the listing data, some only require the FRs to key in the sample lines from a paper list, some even require less. The Census Bureau's current surveys area has seven different listing applications (instruments) that are in production or will be in production shortly.

Some of the listing instruments contain interviewing and listing portions, while others are strictly used for listing information.

2.1 Blaise Listing Instrument Surveys

The Census Bureau has conducted 2 production listing surveys with a third going into production in 2005. Each of the Blaise Listing instruments have there own unique functionality. However, where possible, the same "look and feel" is used to give the FRs some consistency.

2.1.1 American Community Survey – Group Quarters (ACS-GQ)

This instrument is scheduled to go into production in 2005. The ACS-GQ instrument is meant to be more of a "tool" for the FRs to use to conduct sampling of the number of units found at a GQ and then allow for the keying of the sample GQ units. This instrument has three main goals – 1. to sample the number of units found at the GQ, 2. to collect the GQ unit information for the sample units, and 3. to update the control systems with the result of the sampling and listing. This is strictly a listing instrument since it doesn't contain any questions that need to be answered by a respondent.

For this instrument, the FRs only need to key GQ information for the sampled GQ units. This is not a complete listing of the GQ. Once the listing is complete, the sampled information will be "handed off" off to the control systems and the FRs will conduct PAPI interviewing of the sample GQ units.

The FRs and Regional Offices (ROs) will then use the control systems to monitor the progress of the paper questionnaires that are handed out to the sample GQ units. The instrument is responsible for the setting of the initial outcome of the paper questionnaires that are needed for each GQ. Non-sample questionnaires are coded appropriately so they are removed from the control systems. By having the automated instrument and control systems, the Regions know exactly how many questionnaires to expect and how many listings are outstanding.

2.1.2 GQ Automated Instrument for Listing (GAIL)

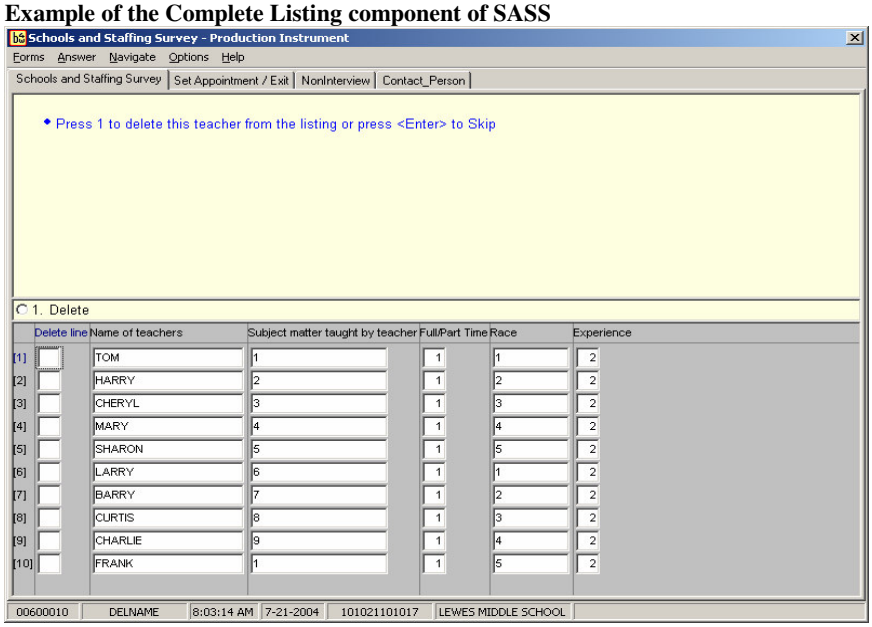
This instrument is currently in production. This is another instrument that collects information about GQs. Although this instrument doesn't do any sampling, it's much more complex than the ACS-GQ instrument since it is used to conduct complete listings at the GQ as well as updating information from a previous listing. The instrument also allows for many listing options.

The GAIL contains both an interviewing piece, where the FRs ask questions of the GQ contact person, and a listing piece - where the FR collects and lists information about the GQ. The information collected by the GAIL instrument is sent back to the Census Bureau and is used to update the GQ sampling frame for the current surveys.

2.1.3 Schools and Staffing Survey (SASS)

This instrument was used in production during the 2003-2004 school year. The instrument contains 3 main components. The first component is a screener portion where the FR makes contact with the school (or address) and interviews a respondent to determine if it's in scope for the survey. The second component of the instrument is a complete listing component where the FR keys in the names of all teachers in the school along with other information about the teacher.

Example of the Complete Listing component of SASS



The third component is a complex sampling component that takes place once the listing is complete.

The instrument then supplies the control systems with the sample teacher name and other listed information for the teachers that fell into sample.

Once the listing is complete, the FRs conduct PAPI interviewing of the sample teachers. They hand out paper questionnaires to the teachers that have fallen into sample. The control systems are used to monitor the progress of the paper questionnaires, as well as the listings.

2.2 CA Clipper Listing Instrument Surveys

The Census Bureau still has a few CA Clipper listing surveys that are in production. These surveys have been in production for years and have had very few problems. All of these surveys will be converted over to Blaise in the near future.

Obviously CA Clipper is out-dated software and these instruments must be converted to a GUI software package. However, using Clipper for coding listing instruments does have some advantages – some of these include:

- Clipper allows you to conduct complex sampling while in the instrument
- Clipper allows you to generate new CAPI cases "on the fly" while still in the listing
- Clipper doesn't require a pre-defined upper limit of listing lines needed for a survey. You can add as many lines as you need for any given case.

Blaise also has advantages for coding listing instruments - some of these include:

- Blaise allows you to easily code questions in the listing instrument. This advantage will allow us to eliminate one of the interview instruments required in SOC by combining it with the listing instrument.
- Blaise has some required functionality (such as the copy from line above) already built-in.
- Blaise generates Audit trail files for help in trouble shooting

2.2.1 Permit Address Listing (PAL)

This is a monthly survey that has been in production since 1996. This instrument collects building permit information for new residential construction. This is a "true" listing instrument. The FRs visit permit offices each month and key in the permit numbers, address information, and geocode information for every residential permit issued that month. This instrument contains no "interviewing" questions. The FR simply enters the listing instrument and starts keying away.

The data collected by the instrument is used by the current surveys to update the new construction sampling frame.

Although it sounds simple, this is a fairly complex instrument behind the scenes. There are many special situations that the instrument must account for. There is also a lot of added functionality to aid the FRs in their listing.

2.2.2 Survey of Inmates and Local Jails/Survey of Prison Inmates (SILJ/SPI)

These two instruments are very similar and alternate in production every couple of years. The instruments are used by the FRs to guide them in obtaining an accurate list of inmates/prisoners, to sample that list based on the number of inmates and sampling information provided, and then to collect the names and IDs of the sample inmates. The instrument then generates the input file needed by the Computer Assisted Personal Interview (CAPI) interviewing instrument.

Once the listing is complete, the FRs conduct CAPI interviews using a separate interviewing instrument. This survey requires additional functionality of being able to copy the new cases to multiple laptops so that one FR can do the listing but, more than one FR can interview at the jail or prison.

2.2.3 Survey of Construction (SOC)

This is a monthly survey that has been in production a long time. This is another instrument that collects permit information from building permit offices. It gathers the permit number, address information, and builder information. It is also a "true" listing instrument that doesn't contain any interview questions. On top of doing a complete listing of the permits for the month, this instrument also conducts complex sampling of these permits and generates input files that are used by another CAPI instrument. Some unique functionality of this instrument is that it generates CAPI cases "on the fly" and allows the FR to call a CAPI interviewing instrument from within the listing instrument.

Once the listing is complete, the FRs conduct CAPI interviewing using a separate interviewing instrument.

3. Unique Functionality of Listing Instruments

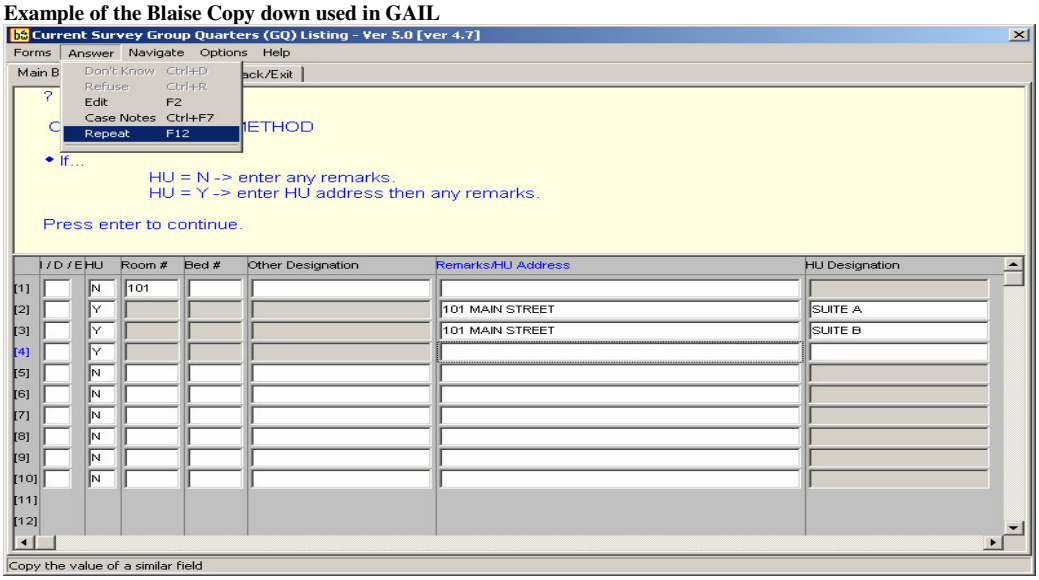
Listing instruments tend to have some unique functionality that isn't normally present within interviewing instruments. This functionality is added to try to help the FR in the listing process and to assure better quality data. Much of this functionality can be handled with Blaise and may be contained in interviewing instruments.

3.1 Copy Contents from Cell Above

In listing operations, much of the data listed may be the same for multiple lines. For example, when listing permits from a Building Permit Office, many of the new permits may be for houses on the same street. These permits are typically taken out at the same time so they will be listed one after the other. Being able to copy the address from the line above not only saves the FR time, but also helps add consistency to the data in the listing.

The Blaise "copy down" feature is a great feature that will receive plenty of use in listing operations. This feature eliminates the need for coding special functionality for a listing instrument.

Example of the Blaise Copy down used in GAIL



3.2 Allow for a Large Number of Rows in the Table

Some listing applications conduct sampling based on the number of units or people at the facility which means they're sampled before the listing is conducted. These applications only require a finite number of listing lines (rows) and are fairly simple to code. However, other listing applications conduct complete listings of an undetermined number of lines. These listings can be thousands of lines long and create an issue for us with using Blaise.

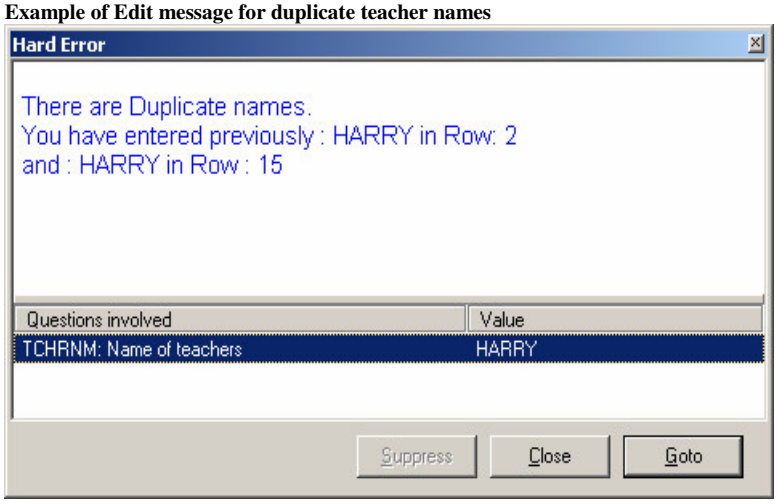
Since one must define an upper limit for a Blaise table, we have to try to determine what the maximum number of listed lines will be for any listing. If we make this number too large, we may create performance issues with the instrument. If we make this number too small, then we risk not being able to list everything for a given assignment. The "maximum number of lines needed" for a listing operation should be supplied by the subject matter experts after they have conducted some research.

3.3 Check for Duplicate Items in the Listing

A fairly common requirement for listing applications is to verify that the FR has not listed any duplicate records. Since listings can be quite large, it is easy for an FR to lose their place and start keying data that has already been keyed. If this is a listing that is going to be sampled by the instrument, many bad things could result.

For example, when creating a complete listing of all of the teachers in a school, we would want to add an edit to verify that each teacher's name entered is not listed on any other line in the listing. Since listings can be quite large, we must be very careful on where this edit is placed or we could create performance problems with the instrument. For SASS, the FR is not allowed to enter in a duplicate name as you can see from the "Hard Edit" below.

Example of Edit message for duplicate teacher names



Example Code used for un-duplication Edit

<u>At the table Level:</u>

```
NumToken : 0..400 3 of rows
Tempstring : String[17200]
Tempstring is defined as an auxfield at table level
Tempstring appendix names delimited by a comma, nutoken is the number of members entered in
tempstring

If Teach[X].TCHRNM <> EMPTY THEN
        Tempstring :=tempstring + Teach[X].TCHRNM + ","
        NumToken := NumToken + 1
Endif
```

<u>At the Row Level:</u>

```
CurrCharPos := 0
PrevCharPos := 1
rowLine := EMPTY
FOR Y := 1 TO NumToken DO
        CurrCharPos := POSITION(',', tempstring, PrevCharPos)
        TempSplit := SUBSTRING(tempstring,PrevCharPos, CurrCharPos-PrevCharPos)
        nametemp[Y] := tempsplit
        PrevCharPos := CurrCharPos + 1
ENDDO
{*** RPV Building a check based on the arrayed stuffed above   ***}



FOR Y := 1 TO NumToken DO
        IF nametemp[Y] = TCHRNM THEN
                rowLine := Y
                ERROR "@L@/ There are Duplicate names. @/
                        You have entered previously : ^ nametemp[Y] in Row: ^rowLine @/
                        and : ^TCHRNM  in Row : ^rownum@L"
        ENDIF
ENDDO
```

If we had allowed the FR to list duplicate teachers, then not only is sampling incorrect, but we could end up with having the same teacher fall into sample twice. That means that we would lose an interview before even trying to obtain it.

3.4 Complex Sampling of the Listing Data:

By automating listing operations, the Mathematical Statisticians (Math Stats) can have the computer do the sampling instead of the FRs. Knowing this, some Math Stats like making the sampling very complex. Not only will they stratify the listing based on the listing data, they'll stratify it four different ways.

Here's a snippet from the SASS sampling specs on how to calculate the Take Every for each stratum:

Compute a take every (TE(h)) for each of the five teacher strata as follows:

Table 3. Take Every Definitions for Teacher Strata

Teacher Stratum	*Take Every*
(A) – Experienced	*Xif SMPSIZAB < (TOTLA + TOTLB), then* $\text{TEA} = \frac{\text{TOTLA} + (\text{OVRSMP} * \text{TOTLB})}{\text{SMPSIZAB}}$
	Xif SMPSIZAB = (TOTLA + TOTLB), then TEA = 1
(B) – New	*Xif SMPSIZAB < (TOTLA + TOTLB), then* $\text{TEB} = \frac{\frac{\text{TOTLA}}{\text{OVRSMP}} + \text{TOTLB}}{\text{SMPSIZAB}}$
	Xif SMPSIZAB = (TOTLA + TOTLB), then TEB = 1
(C) – Bilingual	$\text{TEC} = \frac{\text{TOTLC}}{\text{SMPSIZC}}$
(D) – API	$\text{TED} = \frac{\text{TOTLD}}{\text{SMPSIZD}}$

Since Blaise will not allow you to sort the data efficiently, the complex sampling routine was run in a Manipula script after exiting the DEP. Below is an example of the Manipula script that was used to sort the listing and to sample the first Stratum:

Partial Manipula Code to Sort Listing by Stratum, Subject, and Line Number:

```
FOR Z := 1 TO 4 DO     {//Currently, there are 4 Strata: A thru D}
    currStratum := Z
    currSubj := 0
    FOR J := 1 TO 9 DO   {//Currently, there are 9 subjects}
      currSubj := currSubj + 1
      FOR I := 1 TO Limit DO
            iPntr := I
         IF (UpFile1.Teacher.TeachTbl.Teach[I].TeacStrtum = currStratum) THEN
           SortBySubj(currSubj, UpFile1.Teacher.TeachTbl.Teach[I].SubjSamp,
                                iPntr, SortPntr)
         ENDIF
      ENDDO
   ENDDO
ENDDO
```

Manipula Code Example for Sampling the First Stratum

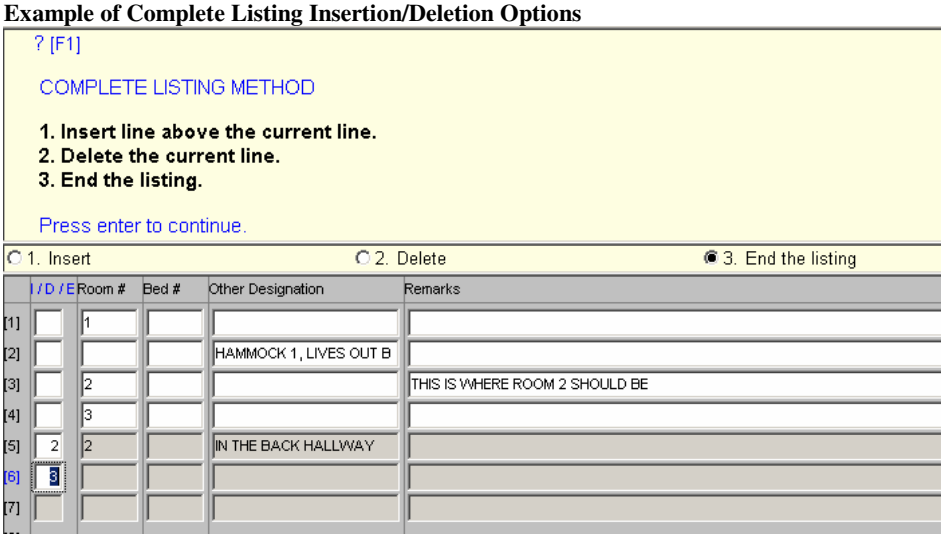
```
{//Initialize the Sampled/Subsampled indicators in the OutTableSampled//}
 FOR I := 1 TO Limit DO
        UpFile1.Teacher.OutTableSampled.Row[I].Sampled := EMPTY
        UpFile1.Teacher.OutTableSampled.Row[I].Subsampled := EMPTY
        IF (StartRowOfStratumC = EMPTY) AND (UpFile1.Teacher.OutTableSampled.Row[I].TStratum = 3) THEN
                StartRowOfStratumC := I
        ENDIF
        IF (StartRowOfStratumD = EMPTY) AND (UpFile1.Teacher.OutTableSampled.Row[I].TStratum = 4) THEN
                StartRowOfStratumD := I
        ENDIF
 ENDDO
 {//Get the raw totals of each stratum.  Remember to skip sampling/subsampling if TOTL(h) = 0!!}
                tTotl_[1] := Upfile1.Teacher.TeachTbl.TotlA
                tTotl_[2] := Upfile1.Teacher.TeachTbl.TotlB
                tTotl_[3] := Upfile1.Teacher.TeachTbl.TotlC
                tTotl_[4] := Upfile1.Teacher.TeachTbl.TotlD
{///Select First Batch from ea. stratum (Step 1)//}
                tSW_[1] := UpFile1.SwA
                tSW_[2] := UpFile1.SwB
                tSW_[3] := UpFile1.SwC
                tSW_[4] := UpFile1.SwD
 FOR stratNum := 1 TO 4 DO
   IF tTotl_[stratNum] > 0 THEN   {//Don't sample if there are no teachers for this stratum (i.e. TOTL(h)=0)!}
       RoundUp(tSW_[stratNum], SampTeacher1st[stratNum])
   ENDIF
 ENDDO
 {//Now do the actual selection for the Step #1 selectees by marking the OutTableSampled table row//}
 FOR stratNum := 1 TO 4 DO
     cntr := 0
     FOR J := 1 TO Limit DO
         IF Upfile1.Teacher.OutTableSampled.Row[J].TStratum = stratNum THEN
            cntr := cntr + 1
            IF cntr = SampTeacher1st[stratNum] THEN
                UpFile1.Teacher.OutTableSampled.Row[J].Sampled := 1
            ENDIF
         ENDIF
     ENDDO
 ENDDO
```

3.5 Insertion and Deletion of Rows

For some listing applications, it is important that the FRs have the ability to insert rows anywhere in the listing. There may be a requirement to have a listing completed in a certain order. For example, we may want units in a GQ to be listed in the order traveled by the FR.

Likewise, it's important to be able to delete a listed line (row). In some cases, we will want this line permanently removed (initial listing) and in some cases we'll want to shade the line (updating a listing) so that we keep a record of it.

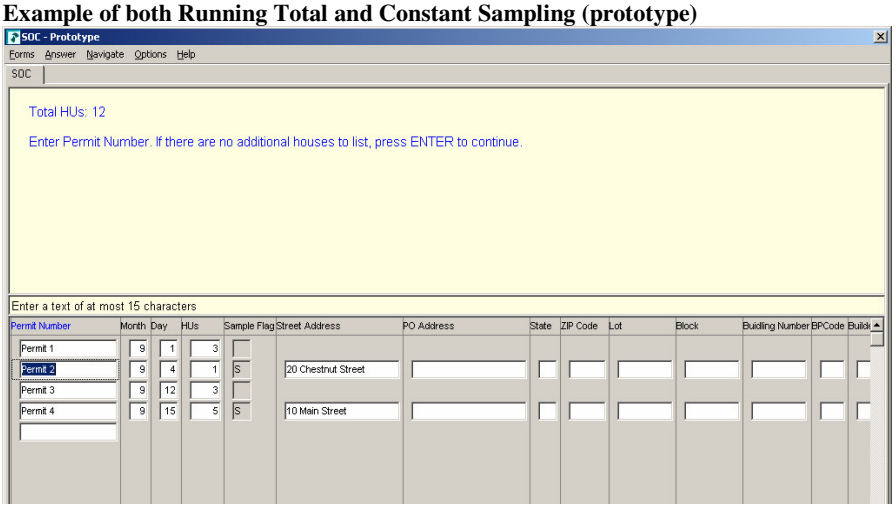
Example of Complete Listing Insertion/Deletion Options



3.6 Running Total of "How Many" Have Been Listed

Some listing operations require a running total to be displayed to the FR. This total is not necessarily the number of lines. Instead, it may be the number of units. For example, a permit may be issued for multiple housing units (HUs). So, the total number of lines listed will most likely not equal the total number of HUs. Total HUs is important information to the FR to help them verify their listing is complete. Again, this functionality is something that can be handled by Blaise.

Example of both Running Total and Constant Sampling (prototype)



3.7 Conduct Sampling After Each Line has Been Listed

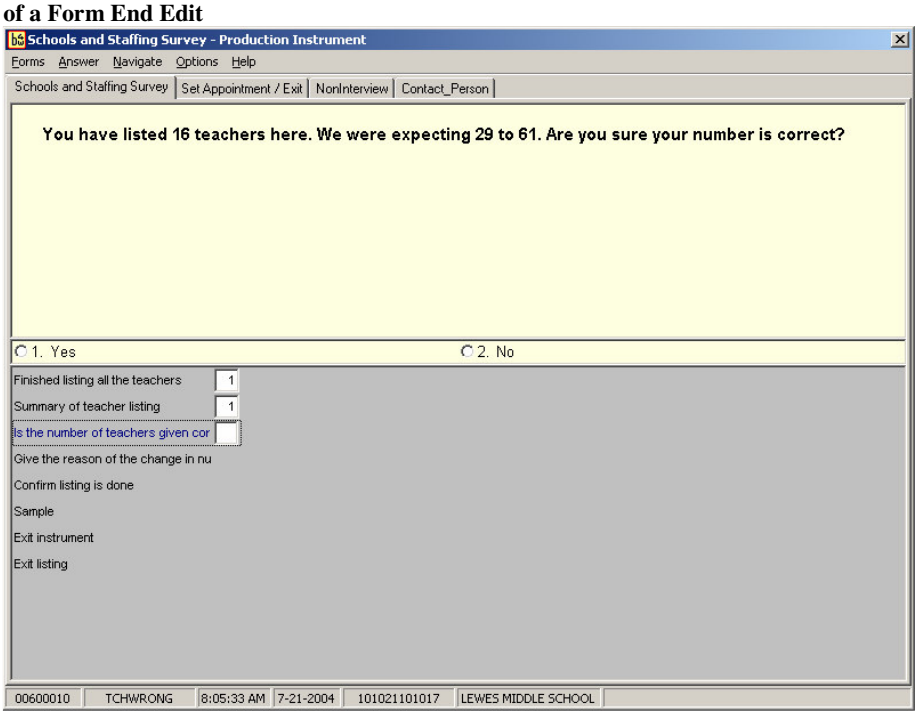
One of our listing applications requires that preliminary sampling be run after every line is entered or modified to indicate to the FR whether that line has a chance to fall into "final sample". This is done so that the FR knows whether to collect additional information from the permit. The FR can collect the information at this time instead of going back through the permits once final sample is run. Blaise handles this functionality as well. In the example above, the "Sample Flag" column is recalculated every time the number of HUs is modified. When the row falls into sample, an "S" is put into the "Sample Flag" column and all of the additional information needed for that permit will come on route.

3.8 Conduct "Form End" Edits

Some listing operations take place in stages. We may collect part of the listing data in one place and finish the listing somewhere else. For example, the FRs may list the addresses for the permits at the building permit office and then finish geocoding each line at home. This means that we have to allow for "incomplete" data in the listing. However, this data has to be completed before the listing is considered "complete". The instrument must be designed so that some edits are only run once the FR says the listing is "complete".

This functionality can be built into a Blaise listing instrument. We just need to have a question to base these edits off of. We would go to an error screen instructing the FR what needs to be "fixed" in order to pass the form end edits.

Example of a Form End Edit



3.9 Exiting a Listing Instrument

Again, the FR must determine when the listing is complete. We allow the FR to exit from the listing in two different ways.

The first way is to F10 to the "Set Appointment/Exit" Screen (see tab in above example). From here, the FR can exit the listing by setting a callback. This sets the outcome of the listing to a partial and the FR must re-enter the listing and complete it. From the "Set Appointment/Exit" screen the FR can also assign a non-interview outcome code to the listing by going to the non-interview screen and selecting one of the options.

The second way the FR can exit the listing, and the only way to assign a complete outcome code to the listing, is to complete the listing, enter "yes" at the "Finished Listing Screen" question, and then resolve all form-end edits. Once that has been done, the listing will wrap up as completed. In the above example, the FR is allowed to complete the listing even though the number of lines listed is outside of the expected range. The FR, however, must explain this discrepancy before exiting the listing.

4 Challenges with Listing Instruments

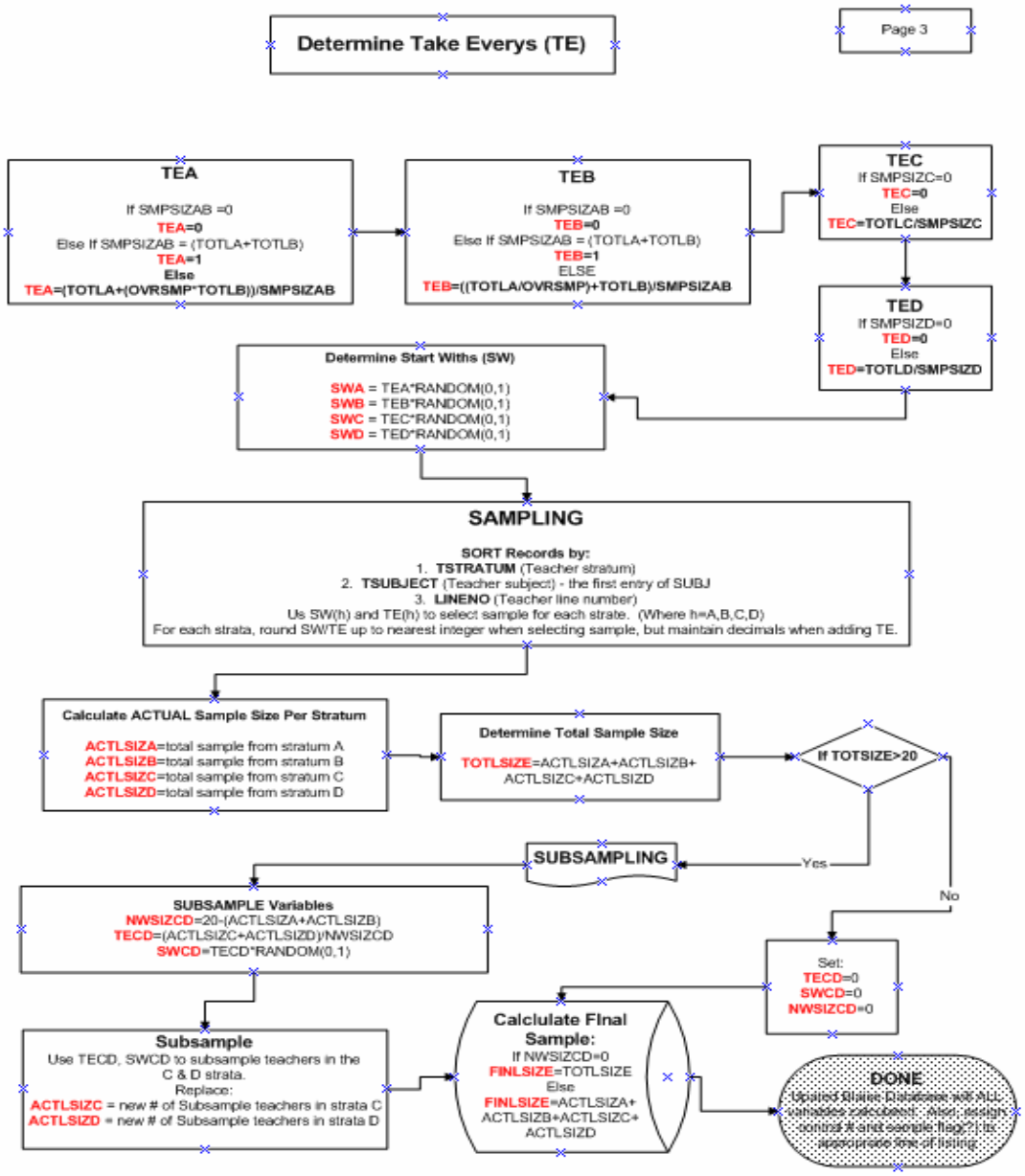
4.1 Coding and Testing Complex Sampling

Regardless of what software package is used, coding complex sampling is not fun. With Blaise it was even more challenging because this had to be handled outside of the instrument. The sampling piece of the instrument is the most important part. If this doesn't work correctly, then it doesn't matter how nice the listing or screener portions worked – the survey is going to be a failure.

The two main challenges with incorporating complex sampling into a Blaise listing instrument were verifying the sample results were correct and trouble shooting if they weren't. With the sampling being so complex, we couldn't easily tell whether the results produced were correct. We just knew that teachers fell into sample. And, with the sampling being done outside the instrument, the watch window was not available for verifying the values assigned to the various variables. It gets very tedious and challenging to verify this is correct. So, what do you do as the programmer?

The first thing one needs to do is flowchart the sampling specifications. This needs to be done to verify that all potential possibilities are covered. This can be very helpful in bringing out the "holes" in the logic. A "hole" in the sampling logic could result in a "fatal error" at the time the sampling program is run in production. Since there are so many possible scenarios, we may not discover this problem during testing.

Example of SASS Sampling flowchart

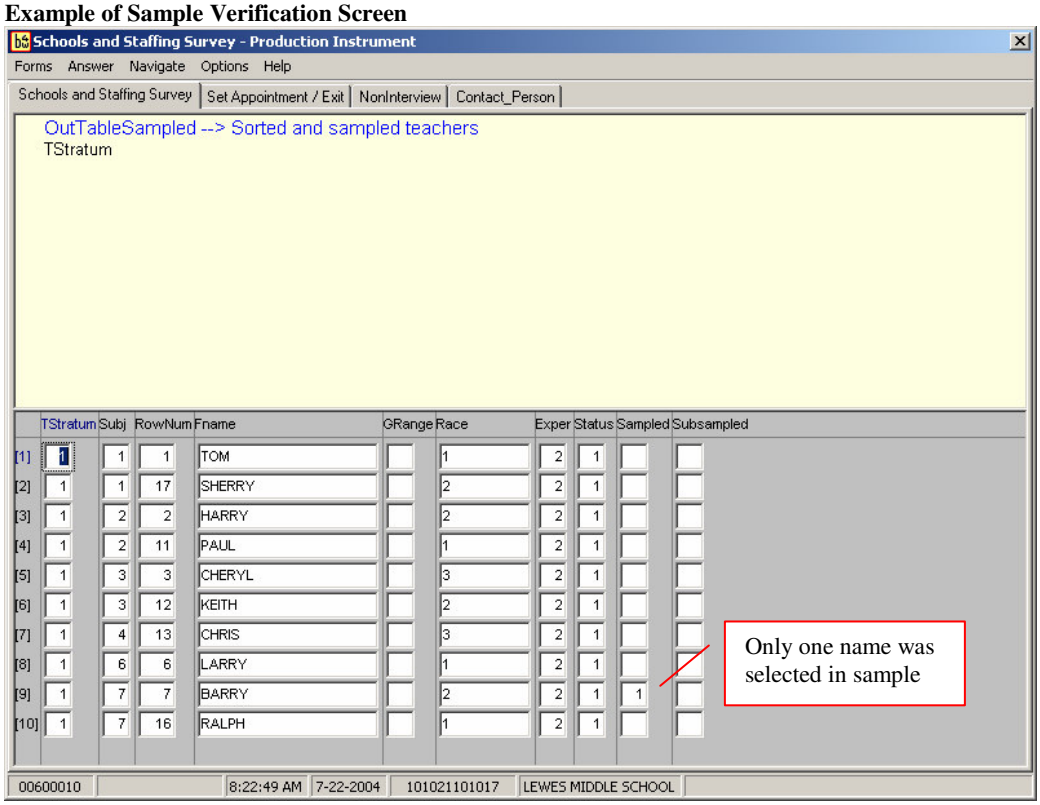


The next step is to program the logic. Since the Manipula code used is somewhat "different" (tricky) than other languages, the code didn't match up nicely to the specs. All of the logic required by the sampling specs couldn't be programmed in a straight forward manor using Manipula. To help verify the sampling specs were understood and that the Manipula code was working as desired, the programmer working on this project decided to code the sampling specs in Visual Basic as well. Results were displayed using XML and compared against the results from the Manipula sampling program. He then verified that the same results were received when testing the same scenario in each language. This not only helped resolve a few issues, but it gave us confidence that the sampling part was working correctly. This made things easier to test as well since the programmer could control the variables in the VB script – he didn't have to worry about the constant re-execution of rules changing the values of sampling variables, such as random number.

Now comes the tough part – testing. How to test and verify the sampling program is working correctly when the sampling takes place outside of the instrument?

A trick we used for testing was to add a "sampling results table" to the instrument that would display the results of the sampling. This table would only be displayed when we re-entered a completed case (this is not allowed in production). This table displayed the stratum assigned to each line, the sort order for sampling, and the sampling and subsampling flags assigned to each listed line. There were also other screens that displayed the values of all of the variables used in sampling. The sponsors and authors simply had to re-enter the case after it was sampled to see the results. This helped the testing process immensely. Note that in the example below, the listing has been resorted based on the sampling requirements. The listing is sorted by stratum, subject, and row number.

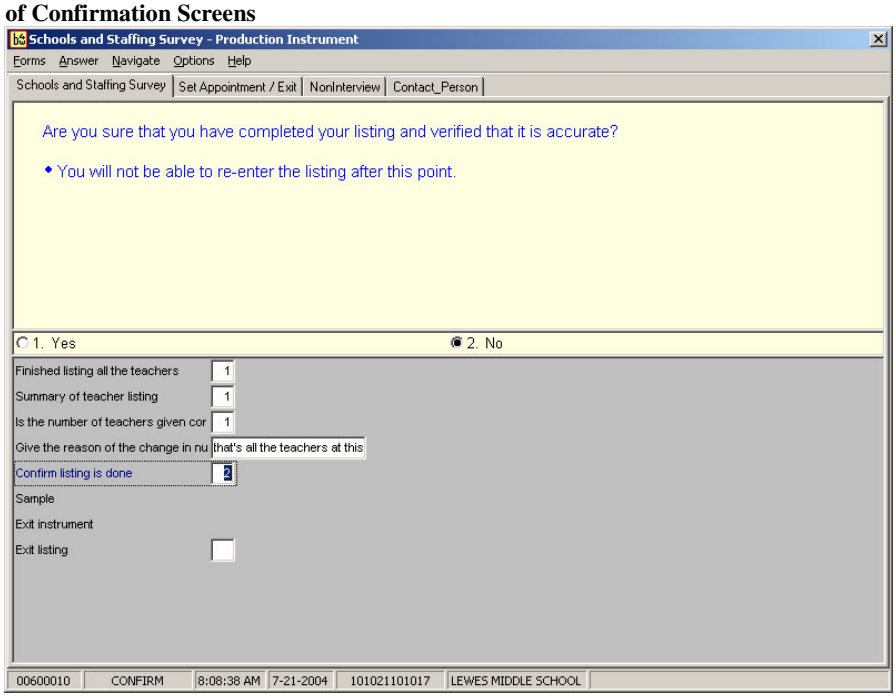
Example of Sample Verification Screen



Although these screens helped us in testing, having complex sampling in a listing instrument makes it very difficult to test. It literally creates thousands of possible scenarios that would need testing in order to verify sampling is working correctly in all of them. We can't possibly test all of them, so we focused on testing the most common ones.

Before running the complex sampling on the listed data, we first had to determine when the listing was complete. We added a series of questions that the FR would answer when they were absolutely sure the listing was done.

Example of Confirmation Screens



Once the FR enters "yes I'm sure it's complete", the instrument exits and the sampling runs. A completed outcome code is assigned to the case and passed out to the Manipula script. Once the Manipula script identifies the case as "complete", it executes the sampling portion of the code.

Manipula Transaction Script Used to Call Sampling

```
IF Upfile.Outcome = '801' THEN  {//This line prevents sampling if case
                                  is incomplete}
       Reslt5 := CALL('subsample /I' + THECASE) {// Susbsample the SASS
                                database's TeacherList table}

{==========================================================================
       Error Checks to verify that Sampling was run & display results
==========================================================================
IF Reslt5 <> 0 THEN
    DISPLAY('Error - Sampling program failed to execute.', WAIT)
  ELSE
    Reslt5            :=            RUN('C:\Progra~1\Intern~2\IEXPLORE.EXE
file:\\fld2\authors\SASS\Sampling\XMLOut.htm' {Sampled.htm}, WAIT)
  ENDIF

==========================================================================}
ENDIF
```

Listing Data from the Blaise database is pulled from the listing table and put into an array where it is stratified, sorted, sampled, and then sub-sampled if necessary. Once the sample lines are identified, the Blaise database is updated with results (assigned sampling and sub-sampling flags). The Blaise database is also updated with the modified sampling variable values used in sampling.

The manipula script then passes out the sample data to the control systems. The control systems are updated with the sampling information so that the FRs can then conduct and monitor the interviews.

Manipula Code used to write out Sampling Information to the Control Systems:

```
IF  (UPFILE1.OUTCOME  <>  '800')  AND  (UPFILE1.OUTCOME  <>  '802')  AND
(UPFILE1.OUTCOME <> '804') THEN         {Completed Case}
 OUTFILE.OutData  := '#ST1CID'
 OUTFILE.WRITE
 OUTFILE.OutData  := UPFILE1.ST1CID
 OUTFILE.WRITE
 OUTFILE.OutData  := '#ST1CNTNUM'
 OUTFILE.WRITE
 OUTFILE.OutData  := UPFILE1.ST1CNTNUM
 OUTFILE.WRITE
 OUTFILE.OutData  := '#TFOUTCOME1'
 OUTFILE.WRITE
 IF UPFILE1.OUTCOME = '801' THEN   {'good' case with data}
    {assign outcome code of the PAPI Questionnaires}
    IF UPFILE1.Teacher.OutTableFinal.TeachSamp[1].TCHRNM = EMPTY THEN
             {Questionnaire not needed - assigns non-sample outcome code}
       UPFILE1.Teacher.OutTableFinal.TeachSamp[1].OutComeCode := '247'
       UPFILE1.WRITE
    ENDIF
       {questionnaire outcome code initialized to 200}
  OUTFILE.OutData := UPFILE1.Teacher.OutTableFinal.TeachSamp[1].OutComeCode
 ELSE        {listing is a Non-Interview}
   OUTFILE.OutData := UPFILE1.SchOutcome
 ENDIF
 OUTFILE.WRITE
 OUTFILE.OutData  := '#STCHRNM1'
 OUTFILE.WRITE
 OUTFILE.OutData  :=upfile1.Teacher.OutTableFinal.TeachSamp[1].TCHRNM
 OUTFILE.WRITE
 OUTFILE.OutData  := '#SUBJ1'
 OUTFILE.WRITE
 OUTFILE.OutData  := STR(UPFILE1.Teacher.OutTableFinal.TeachSamp[1].SUBJ)
 OUTFILE.WRITE
   :

Endif {completed case}
```

4.2 Don't Allow for Re-Execution of Sampling Component

If the listing instrument contains a sampling component, we must be careful not to allow it to re-execute the sampling piece. Once the FR answers "yes the listing is complete" and sampling is run, we can NOT allow the FR to modify anything that will change the sampling results.

For example, if we were conducting sampling inside of the instrument (in Blaise while the FR is collecting data) based on a number entered into a response and then used those sampling results to generate a table that contains sample line numbers that must be keyed, we can't allow the FR to go back and change that number (correct it) and resample. In other words, if that generated listing table is now partially complete, we can't allow the FR to back up and change the value of the "number" or do anything that would then cause Blaise to re-execute the sampling. If this happens, the listing table will most likely have new sample line numbers assigned which would then correspond to different information than that which was already keyed and would be incorrect (i.e., the wrong information is assigned to the sample rows). In the example below, 10 sample lines were generated from the sampling. The "sample line number" column tells the FR which information to key

from the printout or paper listing. If they had partially completed this table and then backed-up and done something to cause re-sampling, the "sample line numbers" would most likely change causing an incorrect listing.

Example of listing table Generated by Sampling

American Community Survey Group Quarters (ACS-GQ) Updating and Sampling ver 1.5

Forms Answer Navigate Options Help

Main | Exit / Non-Interview

? [F1]

• Enter in the person's name or room or bed description from your register for each of the GQ residents that were selected for sample.

We call the person's name, room, or bed description as the GQ unit of measure.

	Delete	Persons, Rooms or Beds	Sample Line #	Phone #	Ext	Control #
[1]			718	( ) -		3904999817504010198911 17
[2]			1952	( ) -		3904999817504010198911 26
[3]			3187	( ) -		3904999817504010198911 35
[4]			4421	( ) -		3904999817504010198911 44
[5]			5656	( ) -		3904999817504010198911 53
[6]			6890	( ) -		3904999817504010198911 61
[7]			8125	( ) -		3904999817504010198911 70
[8]			9359	( ) -		3904999817504010198911 89
[9]			10594	( ) -		3904999817504010198911 98
[10]			11828	( ) -		3904999817504010198912 07
[11]						
[12]						
[13]						
[14]						
[15]						

00000090 | GQ_UOM | 5:16:03 PM | 7-21-2004 | 390499981750401 | ROBINDALE HOUSE | ROBINDALE HOUSE

Depending on the listing, we either have the FRs restart the case and re-sample with the correct number or add in code to the instrument that will "refresh" all of the sample lines whenever the instrument re-samples. This way, we eliminate the possibility of incorrect information being assigned to the sampled lines. We just have to be careful not to "refresh" good data if that's the approach that's chosen.

4.3 Restarting a Case

For most of our interviewing surveys, FRs are allowed to "restart" a case. Restarting a case is basically allowing the FRs a "do-over". When a case is restarted, the current (partially complete) Blaise database and associated files for the case are replaced by the original files sent down for interview. The control systems have to then address any spawns or additional cases that were generated before the case was restarted. The same is true for listing surveys – the control systems have to figure out a way to handle any new cases generated or acted upon by the listing instrument.

For the Census Bureau, this is a little more challenging for listing surveys since the new cases generated by the listing instrument are typically used in a different survey study. If the listing survey generates an input file that is then used to "setup" a new study on the laptop as well as the case management systems, then both the case management system and the newly created study must be "corrected" if you allow for restarts. If the listing survey simply passes out data to the control system then only the case management system needs to be "corrected". This process becomes even more challenging if the FR has already transmitted in completed interviews from cases that were generated by the listing.

4.4 Testing Complex Sampling with the Systems

Just like testing a complex interview instrument, testing a listing instrument with complex sampling is very challenging. First, the author must make sure the sampling script is compiled with the latest instrument and that it's integrated in with the control scripts and Blaise instrument. Before sending off the instrument and sampling program for a systems test, the author should run it "stand alone" and verify that the sampling program is being executed at the appropriate time, the sampling program is producing sampling results, and that the results are in the correct format expected by the control systems.

The most critical piece needed for a good systems test is to have a good realistic input file that has a wide variety of sample cases that cover the most likely scenarios that one can run into during production. If the sampling input data used during a systems test is not realistic then the test is a waste of time.

5. Conclusion

A listing survey instrument is an instrument that is used as a tool by the FRs to collect and sample lists of data in order to generate sample cases for interviewing. These instruments typically don't produce output that is published. This tool has unique functionality that typically isn't required by interviewing instruments. Some of this functionality can be challenging for CAI software packages to handle.

Based on our experience of using Blaise to code a three different listing instruments and to prototype a fourth, we have concluded that Blaise can handle most of the functionality required for listing instruments used at the Census Bureau. The main weaknesses we've identified in using Blaise have to do with having to define an upper limit for our listing tables and the ability to handle complex sampling while running the DEP. Both of these issues can be worked around.

The Census Bureau plans to use Blaise in the conversion of our existing CA Clipper listing applications to a GUI CAI software package. We're confident that Blaise, can handle the required functionality of these applications.

Acknowledgements

I'd like to acknowledge the following authors for their input into this paper and/or work on the various Blaise listing applications coded at the Census Bureau:

Chris Borillo
Alexis Hunt
Roberto Picha

Utilisation de Blaise dans la répartition automatique des fiches adresse entre enquêteurs des enquêtes ménages

Georges Bourdallé, I.N.S.E.E (FRANCE)

1. Présentation

La mise en place de la collecte informatisée sous C.A.P.I d'une enquête auprès des ménages s'accompagne de différentes étapes plus ou moins lourdes, du tirage de l'échantillon national à la répartition en échantillon enquêteurs en passant par la mise à disposition des échantillons régionaux. Trois équipes de personnes sont concernées par celles-ci. Une équipe en charge du tirage de l'échantillon national, une équipe de gestionnaires régionaux en charge du suivi de la collecte et une équipe informatique en charge de la mise à disposition des échantillons enquêteurs. Avant la mise en place d'une répartition automatique des logements ou fiches adresse entre les enquêteurs, la charge du travail entre les équipes n'était pas homogène. Lors de la répartition manuelle de la charge de travail entre les enquêteurs, les gestionnaires d'enquêtes avaient une charge de travail importante sur une période assez courte. En effet, les contraintes faisaient qu'ils n'avaient qu'une semaine pour faire cette répartition. Sur une année courante le nombre de logements enquêtés toutes enquêtes ménages confondues, hors enquête emploi, est de l'ordre de 80000. Au niveau régional, ceci représente un nombre moyen annuel de 3800 logements a répartir . A titre d'exemple, pour l'enquête *Patrimoine* le nombre moyen de fiches adresse à répartir par direction régionale était de 715.

L'objectif de cette répartition automatique de la charge de travail entre enquêteurs est double. Premièrement, diminuer considérablement, de l'ordre de 60%, la charge de travail des gestionnaires d'enquêtes lors de la préparation de la collecte et deuxièmement, une fois le tirage de l'échantillon fait, intégrer la totalité des opérations dans le dispositif C.A.P.I.

1.1 Condition de mise en place d'une répartition automatique

La mise en place d'une répartition automatique des logements enquêtés n'a de sens que si l'on connaît a priori les zones qui seront potentiellement enquêtées. La connaissance des ces zones implique une définition à priori de celles-ci. En France, ceci est rendu possible par la mise en place d'un Echantillon-Maître (E.M) à l'issue du Recensement de la Population (R.P.). L'Echantillon-Maître nous permet d'avoir une liste de communes ou d'îlots (quartiers) parmi lesquelles seront tirés les logements des enquêtes ménages, excepté l'enquête Emploi qui a un échantillon propre. Une fois l'échantillon-Maître constitué, les Directions Régionales procèdent au recrutement de leur réseau enquêteurs en tenant compte de différents critères dont un critère d'éloignement, moins de 30 km, entre la résidence de l'enquêteur et les zones potentiellement enquêtées. Ce réseau enquêteur est donc relativement stable durant la durée de validité de l'Echantillon-Maître. Une fois ce recrutement effectué, les directions régionales sont en mesure de définir les tables de correspondance entre les zones potentiellement enquêtées et les enquêteurs.

1.1.1 Type de logements exclus de cette répartition automatique

Lors du tirage d'une enquête, deux types de logements sont tirés. Les logements issus du dernier Recensement de la Population (R.P. 1999) que nous appellerons les logements R.P et les logements issus de la Base de Sondage des Logements Neufs (B.S.L.N.) que nous appellerons les logements neufs. Ces logements neufs sont situés dans tous les types de ville, aussi bien en milieu rural qu'en milieu urbain. Toutefois, il est à noter que les logements neufs se trouvant dans des villes de plus de 20 000 habitants ne sont pas îlotés, c'est à dire que nous sommes incapables de les localiser au niveau géographique îlot, leur zone d'implantation ne peut donc être affectée a priori à un enquêteur. Dans l'avenir, ceci sera corrigé, l'I.N.S.E.E ayant planifié une opération d'îlotage des logements issus de la B.S.L.N. Les logements neufs représente 7% des logements tirés.

2. Les tables régionales de correspondance entre zones enquêtables et enquêteurs

Du fait de la constitution d'un Echantillon-Maître et d'un réseau d'enquêteur relativement stable les tables de correspondance sont définies une fois pour toutes. Au cours du temps, celles-ci seront mises à jour par les directions régionales à partir d'un datamodel BLAISE. Le taux de pré affectation validé par les directions régionales dépendra du soin apporté lors de la création de ces tables de correspondance.

2.1 Mise à disposition des zones potentiellement enquêtées aux directions régionales et affectation des enquêteurs

Les zones potentiellement enquêtées sont transmises à chaque direction régionale sous forme d'une base de donnée BLAISE. Il est demandé aux directions régionales de compléter cette base de donnée par deux types d'information. Un premier type d'information relatif à l'enquêteur, à savoir le numéro national enquêteur, le nom et prénom de l'enquêteur. Ce premier type d'information permettra la répartition et le paiement de l'enquêteur et un second type d'information relatif aux enquêtes pour lesquelles l'enquêteur est pressenti. Ce second type d'information permettra, en vue de l'appariement avec le fichier échantillonné, la sélection de l'enquête. Les gestionnaires d'enquêtes auront ainsi la possibilité d'affecter une charge de travail à un enquêteur à échéance de 2 ans. La constitution et la mise à jour de ces tables se feront sous forme de questionnaire électronique.

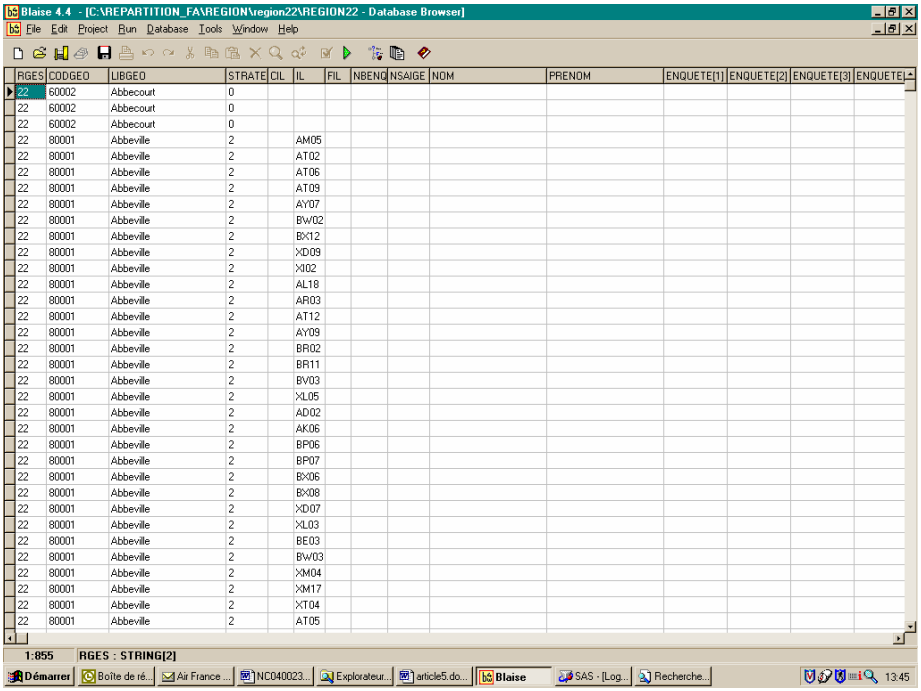
2.1.1 Datamodel de saisie des informations enquêteurs et enquêtes

La saisie par les gestionnaires d'enquêtes des directions régionales I.N.S.E.E des informations relatives aux enquêteurs et aux enquêtes, pour une zone enquêtable donnée, est faite à partir d'un Datamodel développé en Blaise. Quatre types de champs sont proposés aux gestionnaires d'enquêtes. Un premier type de champ relatif aux variables géographiques, ce type de champ indique la zone enquêtable. Un second type de champ relatif au nombre possible d'enquêteurs pouvant agir sur la zone géographique. Un troisième type de champ relatif aux variables d'identification enquêteurs à savoir le numéro d'identification, le nom et prénom de l'enquêteur. Et enfin, un quatrième type de champ relatif aux enquêtes choisies. Concernant ce dernier type de champ, le gestionnaire a la possibilité d'affecter une ou plusieurs enquêtes parmi celles proposées à un enquêteur. Le gestionnaire d'enquête aura la possibilité de saisir les deux derniers types de champ autant de fois que d'enquêteurs sur la zone (voir écran 3 du Datamodel). La liste des enquêtes proposées comprendra toutes les enquêtes planifiées à échéance de deux ans. Seuls les champs relatifs aux variables enquêteurs et enquêtes sont ouverts à la saisie.

2.1.2 Datamodel de mise à jour des zones enquêtables

Trois écrans composent le datamodel de mise à jour. Un premier écran donnant, pour une région, la liste des zones enquêtables. Un second écran définissant le nombre d'enquêteurs pressenti et enfin un troisième écran relatif aux informations enquêteurs et enquêtes.

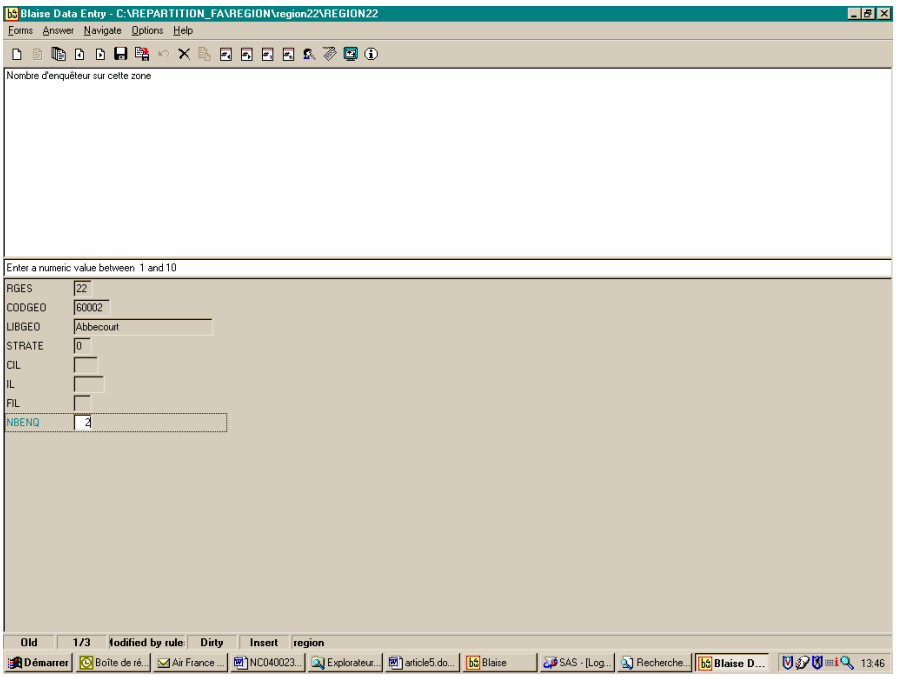
Ecran1



RGES	CODGEO	LIBGEO	STRATE	CIL	IL	FIL	NBENQ	NSAIGE	NOM	PRENOM	ENQUETE[1]	ENQUETE[2]	ENQUETE[3]	ENQUETE
22	60002	Abbecourt	0											
22	60002	Abbecourt	0											
22	60002	Abbecourt	0											
22	80001	Abbeville	2		AM05									
22	80001	Abbeville	2		AT02									
22	80001	Abbeville	2		AT06									
22	80001	Abbeville	2		AT09									
22	80001	Abbeville	2		AY07									
22	80001	Abbeville	2		BW02									
22	80001	Abbeville	2		BX12									
22	80001	Abbeville	2		XD09									
22	80001	Abbeville	2		XI02									
22	80001	Abbeville	2		AL18									
22	80001	Abbeville	2		AR03									
22	80001	Abbeville	2		AT12									
22	80001	Abbeville	2		AY09									
22	80001	Abbeville	2		BR02									
22	80001	Abbeville	2		BR11									
22	80001	Abbeville	2		BV03									
22	80001	Abbeville	2		XL05									
22	80001	Abbeville	2		AD02									
22	80001	Abbeville	2		AK06									
22	80001	Abbeville	2		BP06									
22	80001	Abbeville	2		BP07									
22	80001	Abbeville	2		BX06									
22	80001	Abbeville	2		BX08									
22	80001	Abbeville	2		XD07									
22	80001	Abbeville	2		XL03									
22	80001	Abbeville	2		BE03									
22	80001	Abbeville	2		BW03									
22	80001	Abbeville	2		XM04									
22	80001	Abbeville	2		XM17									
22	80001	Abbeville	2		XT04									
22	80001	Abbeville	2		AT05									

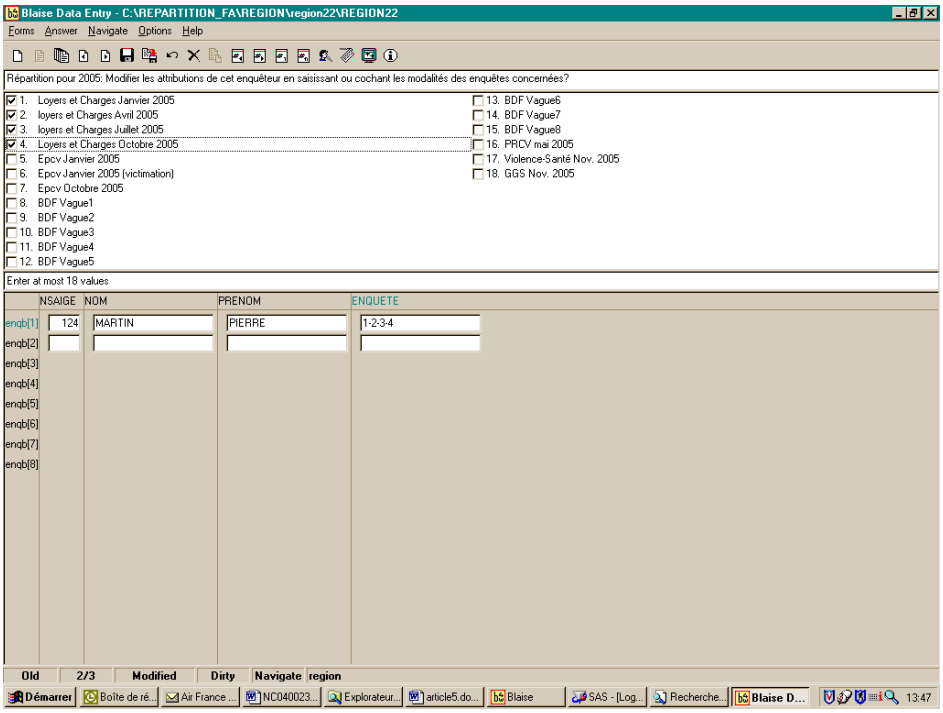
Le gestionnaire d'enquête visualise la liste des zones enquêtables de sa région. Cette liste est triée par département et au sein de chaque département, par nom de commune.

Ecran 2



La première information que le gestionnaire d'enquête doit saisir concerne le nombre d'enquêteurs susceptibles d'être affectés sur la zone concernée. En effet, en fonction de divers paramètres tels que l'étendue de la zone enquêtable, ou bien le type d'enquêtes ou encore des périodes de l'année pendant lesquelles certains enquêteurs ne souhaitent pas faire d'enquêtes, ce nombre peut être supérieur à 1. Il est donné la possibilité aux gestionnaires de définir au plus 10 enquêteurs pour une zone.

Ecran 3



Une fois saisi le nombre d'enquêteurs susceptibles d'être affecté sur la zone, le gestionnaire d'enquêtes saisira les informations relatives aux enquêteurs ainsi que leur charge de travail en terme d'enquêtes.

2.1.2 Contrôle de validité des variables d'identification enquêteurs

Afin d'éviter pour le gestionnaire d'enquête lors de la validation de la répartition une charge de travail importante de ré affectation dû à des erreurs de saisie des variables d'identification enquêteur, nous procéderons à un contrôle avec une base enquêteur national appelée « Référentiel enquêteur » dans laquelle sont déclarés tous les enquêteurs en contrat avec l'I.N.S.E.E. Ce « Référentiel enquêteur » étant particulièrement sécurisé, ces accès sont limités. Aussi le contrôle de validité se fera à deux niveaux. Au niveau de l'équipe en charge de la chaîne de tirage, une fois les informations de toutes les directions régionales réceptionnées et à un second niveau, au niveau du poste de gestion, une fois les ré affectations faites et la répartition validée.

2.2 Périodicité des mises à jour

2.2.1 Mise à jour des tables de correspondances

La périodicité de mise à jour des tables de correspondance zones enquêtées/enquêteurs est variable et dépend essentiellement d'imprévus ou contraintes relatives aux enquêteurs (refus de faire l'enquête, maladie, indisponibilité, charge de travail trop importante,….). Pour une facilité de gestion, elle doit rester marginale. La date limite de mise à jour de ces tables permettant une prise en compte pour une enquête donnée est de 30 jours avant le début de la collecte.

2.2.2 Mise à jour du Datamodel de saisie

La mise à jour et la mise à disponibilité du datamodel de saisie est faite dés la planification d'une nouvelle enquête. Sachant qu'à l'I.N.S.E.E, on essai autant que faire se peut de planifier une enquête au minimum 2 ans avant le début de collecte, les mises à jour du Datamodel se feraient tous les deux ans.

2.3 Restitution des tables de correspondances complétées

Une fois complétées par les informations relatives aux enquêteurs et aux enquêtes, les tables de correspondance régionales sont mises à disposition du responsable du tirage des échantillons d'enquête sur un serveur sécurité FTP.

3. L'affectation des enquêteurs à un échantillon donné

L'opération d'affectation des enquêteurs pour une enquête donnée est faite de façon totalement automatique. Elle consiste en l'appariement d'une table nationale des zones enquêtées/enquêteurs avec le fichier échantillonné. Se faisant en aval du tirage de l'échantillon, cette nouvelle étape n'a aucune incidence sur la chaîne de tirage. Elle n'occasionne aucune charge de travail complémentaire pour l'équipe en charge du tirage.

3.1 Appariement du fichier échantillonné avec les tables de correspondance régionale

Les fichiers régionaux BLAISE des tables de correspondance sont mis au format SAS. A partir des ces tables SAS régionales, nous obtenons une table SAS nationale comprenant les trois types de variables. Les variables de localisation géographiques, les variables d'identification de l'enquêteur et la variable d'identification d'enquête. Une fois la sélection sur la variable d'identification d'enquête faite, le résultat est appareillé avec le fichier échantillonné suivant les variables de localisation géographique.

4. Intégration du fichier échantillonné enrichi de la variable « enquêteur » dans le dispositif CAPI

Le fichier échantillonné enrichi de la variable « enquêteur » est fractionné en deux temps. En un premier temps en fichiers régionaux à destination des gestionnaires et en second temps en fichiers enquêteurs à destination des enquêteurs.

4.1 Quelques informations sur le dispositif CAPI à l'I.N.S.E.E

Le dispositif CAPI à l'I.N.S.E.E repose sur trois types de postes ou applicatifs. Un poste de collecte, un poste de gestion et dans certain cas, un poste concepteur. Le poste de collecte est un applicatif installé sur le micro ordinateur portable de l'enquêteur lui permettant de télécharger son échantillon, c'est à dire ses fiches adresses, de faire la collecte et transmettre l'information collectée sur un serveur FTP. Le poste de gestion est un applicatif installé sur le micro ordinateur de bureau du gestionnaire lui permettant actuellement de préparer et gérer la collecte, c'est à dire suivre l'état d'avancement de la collecte, procéder le cas échéant à de l'apurement et enfin valider les enquêtes pour la rémunération des enquêteurs. A terme, avec la mise en place de ce nouveau dispositif de répartition de l'échantillon entre enquêteur, le poste de gestion intègrera une nouvelle fonctionnalité de validation ou modification de la répartition des fiches adresses entre enquêteurs. Le poste concepteur est un applicatif développé uniquement à la demande du concepteur d'enquête. Le

développement d'un tel applicatif est demandé uniquement lorsque la complexité du questionnement impose qu'une partie de l'apurement soit fait par le concepteur de l'enquête.

4.2 Mise à disposition des échantillons régionaux sur le poste de gestion

Les fichiers régionaux d'une enquête sont mis à disposition des gestionnaires des Directions Régionales pour validation directe ou validation après ré-affectation. Cette validation directe ou après ré-affectation se fait à partir du poste de gestion CAPI.

4.3 Mise à disposition des échantillons enquêteurs sur le poste de collecte enquêteur

Une fois validés par les gestionnaires d'enquêtes, les fichiers régionaux sont éclatés en fichiers enquêteurs et mis à disposition de ceux-ci afin de récupération sur le poste de collecte enquêteur à partir d'une connexion téléphonique sur le serveur FTP.

5. Résultats des premières simulations

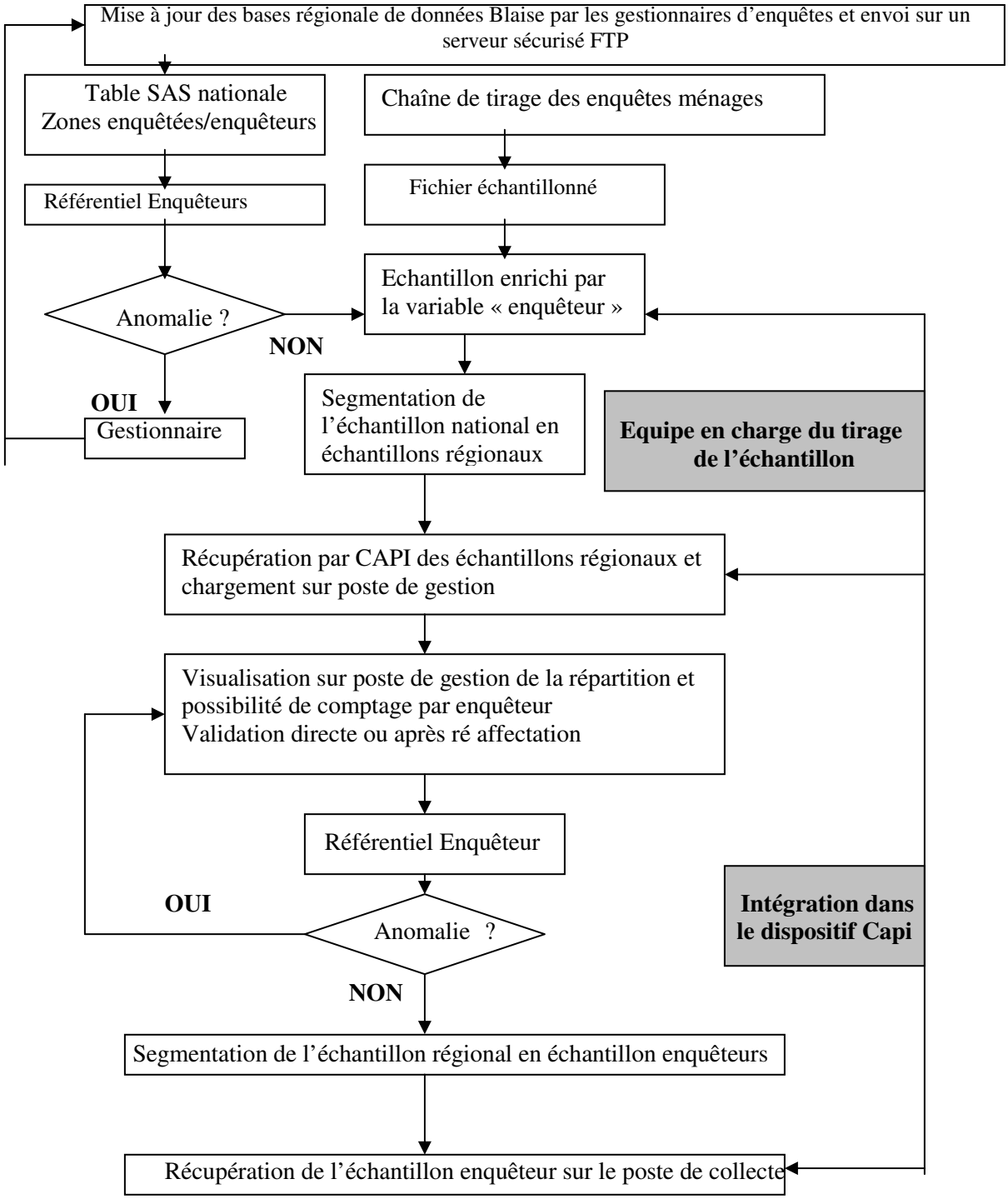
Afin de quantifier le volume des pré affectations, les tests ont été faits auprès de huit directions régionales représentatives en terme de charge d'enquêtes. Ces tests ont portés sur deux types d'enquêtes, un premier type d'enquête, une enquête de grande taille, 15000 logements interrogés, ayant une périodicité de cinq ans, l'enquête PATRIMOINE et un second type d'enquête, des enquêtes de taille moyenne, 8000 logements interrogés, ayant une périodicité de trois fois par an, les enquêtes sur les Conditions de Vie des Ménages (Enquête EPCV). Les logements neufs ne pouvant être îlotés ont été préalablement retirés.

Taux de réussite sur les pré-affectations (en %)

Région	***Patrimoine***	***EPCV Janvier 2003***	***EPCV Juillet 2003***	***EPCV Octobre 2003***
Centre	64	50	46	39
Nord-Pas-de-Calais	36	65	60	61
Lorraine	42	54	50	44
Alsace	31	33	42	32
Franche-Comté	57	66	54	58
Pays de Loire	40	45	40	42
Rhône-Alpes	57	69	69	52
Languedoc-Roussillon	53	55	60	59
S/Total	48	55	53	48
Total	48	52		

Dans certains cas, 69% des fiches adresses sont affectées a priori correctement. Les différences régionales s'expliquent par le soin et la rigueur apportée à la constitution des tables de correspondance. Ceci nous amène à penser qu'avec un suivi plus rigoureux de ces tables, ces taux pourraient nettement s'amélioraient.

6. Schéma d'ensemble de la répartition automatique des fiches adresses



7. Références

Georges Bourdallé, Marc Christine, Laurent Wilms, Echantillons Maître et Emploi, 2000, Actes des journées de méthodologie statistique, Insee Méthodes, N° 100.

Contact History Instrument (CHI)

William E. Dyer, Jr., U.S. Census Bureau July 07, 2004

The Contact History Instrument (CHI) was designed to collect information about each contact attempt made by a field representative (FR), including information about why respondents refuse and what actions the FR took to attempt to obtain the interview. This paper explains the rationale behind this instrument, the development approach we used for this project, CHI's place in the interview sequence, preliminary results from the field, and the vision for the future of this initiative. We are working toward collecting, collating and tabulating data about contact and non-contact attempts within and across surveys. The contact history will provide the field representative, senior field representative, headquarters, and the sponsors with additional data about contact history attempts for all surveys requesting this service.

1. Introduction

The initial goal of the CHI was to obtain information for researchers to analyze trends in respondent refusal. Its intended use has now expanded, and it has become a tool for the FRs, and Regional Office (RO) Survey Managers to look for patterns of what works and what does not work to obtain complete interviews from reluctant respondents. The next step for CHI is to return previous interview contact history data for longitudinal surveys. The Census Bureau is in the process of implementing a generic system capable of gathering contact history data for all types of surveys.

The CHI is a data collection instrument, written in Blaise following Census standards for screen design and programming. The supporting post-data collection Manipula script converts the answers into an ASCII transaction record for the CAPI Case Management (CM) systems. This paper addresses the "generic" aspects of this application. We look at how the Census Bureau intends to use the CHI to collect contact history data and perform analysis on the metrics gathered by this process. In addition the paper will touch on the research effort underway to reduce non-response. We are especially interested in how senior field representatives are using CHI to identify and possibly convert potential refusals into completed interviews.

1.1 History

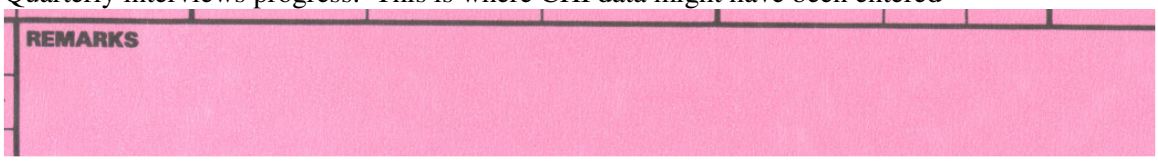
The oldest reference I can find, to Ch'i – (pronounced Chee) The Samurai Martial Art of Jiu-Jitsu Eight Principles of a Warrior hangs on a note above my desk; the 6th tenet is **CHI – Enhance wisdom by broadening your knowledge**. This definition fits the CHI project, pronounced (K-eye) very well. We hope to improve the data collection process for our sponsors by learning more about all interviewers' contact attempts. CHI may be part of the solution to improve response rates and manage rising survey costs.

The decision was made to only allow a single version of CHI in the field at any time. This instrument must be designed to report for all types of surveys. We are trying to save costs and increase quality by having a single instrument instead of creating unique versions for every survey. Also, in this way the FRs who work multiple surveys only have to learn one CHI, thus reducing training time and expense.

Disclaimer: This report is released to inform interested parties of ongoing research and to encourage discussion of work in progress. The views expressed are those of the author and not necessarily those of the U.S. Census Bureau.

The development of CHI did not happen all at once to be sure. Back in the days of old we had paper forms and control cards, which allowed the interviewer to record each interview attempt and to make remarks. For recurring surveys interviewers would record dates and times of interview

attempts and make remarks to record successful methods used to get the interview. The image below is part of the big pink folder an interviewer used to keep track of the Consumer Expenditure Quarterly interviews progress. This is where CHI data might have been entered



Part of: FORM **BC-1669** (CEQ) (6-19-93)

As questionnaires moved from paper to electronic format some of that handwritten information was lost. For CAPI, only the final outcome code for the case was captured. Until a case was made a "resolved" case, the case was given a "202-open" outcome, which gave the Regional Office (RO) no information as to what was going on with the case. We developed a number of ways to record and track the FR notes and observations. The Case Management staff created the Note_Editor application, which is scheduled to run after each interview to record important information about the case not captured during the interview. This technique provided the FRs a good way to pass information from month to month, but the information in the notes is difficult to analyze in a systematic fashion.

When the laptop Case Management system was redesigned for the move from DOS to Windows, a record of calls table was added as a first attempt at capturing date, time and outcome data each time the instrument was opened. Unfortunately, this did not permit us to capture non-contact attempts because the FRs usually do not open the instrument in those cases. Also, the "202" outcome problem still existed; that is, multiple attempts could be recorded but all with a "202" outcome. CATI does maintain call history records that provide more detail than CAPI has, but they do not capture information about why respondents refuse. The focus of CHI now is CAPI.

What started out as an Interagency Household Non-response Group requirement to provide data to headquarters for evaluating non-response, has evolved into something more. The Contact History Initiative is an interactive, real-time tool to support field operations for both interviewers, senior field representatives (SFR) and the Regional Office Staff. We have added reporting capability for the users. Screens display tables of CHI data to the FRs and SFRs and numerous standard and customizable printed reports are available to the RO staff.

CHI version 4.8 was first fielded for the National Health Interview Survey (NHIS) in early January 2004. The returning data was analyzed and seems to confirm our expectations regarding reasons for non-contact, break-offs and refusals.

The Type-A non- interviews require more work, increased focus on making contact and more often result in refusal to participate in our survey.

Average number of contact attempts by final outcome: 2004 National Health Interview Survey Q1W3-W4

Final Outcome	Average Number of Contact Attempts	Total Number
Completed	3.2	1372
Sufficient partial	5.0	264
TypeA noninterviews	6.8	223
Insufficient partial	5.8	24
No one home	8.6	54
Temporarily absent	7.5	11
Refused	6.2	109
Other	6.4	25
TypeB noninterviews	2.1	376
Screened out	2.5	572
TypeC noninterviews	1.9	197

Table provided by: Andrea L. Piani, The use of Field Metrics and Paradata in evaluating CASIC Surveys, CAPI History Data CHI

2. Development Approach

2.1 Requirements

The Contact History Instrument was developed as a series of prototypes. These iterations were further refined by subject matter specialists then reviewed by the user community i.e., Field Representatives, in focus groups. It was necessary to test the instrument and develop a method to integrate CHI with existing control systems prior to release. Our expectation was not to get everything (instructions, categories, codes and navigation) exactly right in the first edition. We chose to put CHI in the field and reevaluate our approach at regular intervals.

The first version of the CHI had more of a National Health Interview Survey (NHIS) focus toward a single survey because this sponsor was willing to initiate CHI development. This survey interviews households only one time. Since the premier version did not address contact history issues for surveys that conduct multiple interviews with the same household or respondents, after the preliminary results were returned and analyzed the Census Bureau formed a group to identify additional requirements for other types of surveys. In February 2003 the Contact History Automated Interview for Recurring & Multi-mode surveys (CHAIRMS) committee met (see Nelson, 2003). The group developed requirements needed to program a single CHI that can be used for all surveys.

2.2 Development Philosophy

When possible, the CHI tried to use existing resources and piggyback on systems already in place. I do not want to imply that we did CHI on a shoestring budget, but our approach has been, and still is, very frugal. We have been diligent in including the stakeholders in the process from the beginning. The single instrument approach gives focus to our efforts. Using accepted established procedures for project development and applicable Census Bureau standards we embrace the Capability Maturity Model (CMM) best practices and meet process improvement goals for

authoring instruments in Blaise. We work hard to limit overhead to our control systems, causing minimal impact. The CHI system tries not to consume additional resources and attempts to reduce FR burden in the process.

2.2 CHI Development

By using written specifications and flow diagrams developed from the original CHI, the CHAIRMS and focus groups have been able to define requirements for the next version of the instrument. The subject matter personnel facilitate our development efforts by documenting the specifications and applying change requests to our system. As changes are approved they are integrated into the instrument using accepted Census Bureau Authoring Configuration Management processes. The TMO Testers Menu is used as a platform for testing and evaluating as well as a distribution method for a stand-alone zip version of the CHI system for independent testing. This permits multiple users to review our work. They may make suggestions for modifications to either the steering group or committee responsible for approving and implementing change requests to the CHI v5.7 instrument and Case Management sub systems.

An example of a flow diagram is included for your review in Appendix A. This documentation has helped make instructions by the design team, developers and testers clear and contributed to making the code easy to write and modify. A staff member who was given the task to make a CHI prototype and apply changes to existing code commented that he liked CHI because it is manageable, straightforward and well documented. Though it is a bit of work to set up the flow chart for the first version, it is invaluable for future iterations of development and testing. Being able to train new employees and communicate with experienced developers who are unfamiliar with the project is a bonus. I strongly recommend this approach given the feedback I receive from programmer and specification writers.

2.3 "The Prime Directive": CHI's Place in the Interview Sequence

CHI will not interfere with the survey instrument data collection effort in any way.

Because the CHI was an untried process, we did not want to risk affecting survey data, so the CHI instrument is kept completely independent of the specific survey instrument processing.

CHI is launched in one of two ways:

1. Directly from Case Management – This option is for contact attempts where the FR did not launch the survey instrument. This covers instances such as phone calls to set up appointments, drive by attempts, meeting neighbors and distribution of literature.

2. At the end of an interview session – The CHI instrument is launched automatically whenever the survey instrument is closed. In this option the post-data collection processes, such as releasing the Blaise database; collection of other information from the note_editor; compacting files and saving the interview back into the control system database, are done before CHI is invoked. This is in part why the interviewers think CHI takes a long time to launch. We actually open Manipula a second time, which also takes some time to load, and create a new CHI database, but this isolates the contact history transaction, insuring non-interference with the survey interview.

3. The CHI Process

3.1 The Instrument

Appendix C lists a subset of the CHI instrument Blaise code and gives the reader a bit of insight to the inner workings of this tool. For programmers, the interesting part of this project is in its simplicity, its design for maintenance and how the CHI instrument approaches a pure data collection effort. The CHI is unhampered by post-data collection issues, irregular navigation, sophisticated edits and the like. It is after all a tool for the interviewer.

The instrument itself is designed to be very simple. It does not require the interviewer to ask questions in the way a traditional CAPI interview does. The InfoPane is entirely made up of interviewer instructions (□ in blue). It is important to note that none of the questions are read aloud. This format for data collection from FR's is more like data keying than interviewing.

The shortest path (best case scenario) through the instrument takes seven keystrokes. (10 seconds)

1) Q: □Are you making a contact attempt or just looking at a case?
   A: Contact Attempt
2) Q: □Are you entering CHI at the time of a contact attempt?
   A: Yes
3) Q: □Was this a personal or telephone contact attempt?
   A: Personal
4) Q: □Select the category that best describes this attempt.
   A: Contact with SAMPLE unit member
5) Q: □Status - Select the category that best describes this attempt.
   A: Completed case - ready to transmit
6) Q: □Select the categories that describe respondent concerns, behaviors, or reluctance during this contact attempt.
   A: No concerns
7) Q: □Select the categories that describe the strategies used on this contact attempt.
   A: Scheduled appointment

We record some data fields automatically or fill them if left blank; for example, date and time. Other variables are always asked, such as, contact or non contact; personal or telephone; status; and now the reluctant respondent screen is always on route. Contrary to some "instrument design" techniques you may learn in school, from books or which are applied to typical interviewing, CHI may duplicate a response or allow the FR to enter "No Concerns;" for example, as a verification that the reluctant respondent screen has been visited. The data we are trying to collect from CHI is a bit more ethereal, and harder to capture, than "yes or no" or enter a number between 0 and 99.

As you can see from inspecting Appendix A, the CHI v5.7 Flowchart, other screens come and go depending on the items offered in the enumerated set and which have been selected in the answer list box. We may want to know more detail about language difficulties, movers or simply allow the FR to fill in "Other Specify ____" entries should the need arise. Most of the screens in CHI have "mark all that apply" and enumerators are trained to mark their answers in priority order. With the most important option being keyed first. This high order event is displayed in Case Management with other lesser issues being available for review, if the FR wishes do go data mining, by selecting the control button (snowflake) in Case Management next to the primary category listed.

To see additional screen detail review Appendix B. The first iteration of CHI. These screens show the scope of the original instrument. A CHI questionnaire is quick to get through no matter the circumstances of the event; i.e., contact or non-contact. It may take longer to run Manipula to load

CHI on the laptops than it does to actually collect the data and make an output file. The instrument is small, the process simple and keying quick. The next issue to be dealt with after the instrument is built and tested is to hook in with the existing control system. To get this done we put Maniplus to work for us.

3.2 The Script

To make this application sharable across Case Management and our Regional Office Survey Control System (ROSCO) a code was defined for each item, Alpha and two-digit numeric codes are used, something like the key codes used at our main data processing center in Jeffersonville, Indiana. I have seen Agriculture, Economic and other forms keyed this way. One advantage is that the coding scheme can be used across surveys and over time if you retire old codes rather than reuse the same codes for different categories for each new release.

C00 Contact
C01 Completed case - ready to transmit
C02 Partial interview - follow-up required
C03 Unable to conduct interview"
L00 Language
L01 Specify language or dialect _____
L02 No household member able to translate
L03 Contact RO about language problem
L04 Unable to find translator
L05 No time left to find translator

These codes map the variable with the text for that answer. When data is passed, files copied or transmitted, it is in the key code + "#write-in" format. The codes need to be translated for display in Case Management or to make some types of reports, but not for tabulations and summary tables. This format should be very easy to query in SAS, for example, once the data is released to HQ.

Keep in mind that the programmer wants to keep the questionnaire simple; simple is good! A consistent look-up table is important to the script which creates a transaction record for each contact attempt. This standardized transaction means that changes and revisions can be isolated in the instrument and transaction processor without having to upgrade the external processes unless the look-up table changes when new codes are added or answer category text is modified. This should be a cost saving and effective method to support CHI over the life of the project.

Following is the transaction file created for Case Management (CM):

#caseid 00000001 #frdate 07/26/2004 11:13 AM #howcontacted P #contactstatus C	#detail CU #types C03$^U04$^L01#German #strat S18$^S17$^S12

This transaction is for caseid 00000001
The FR overrode the automatic date and entered 07/26/2004 11:13 AM
It was a personal visit, Contact, unable to initiate interview
The path in the instrument was: Language problem (Specify _____) – German
Strategies attempted in priority order were:

S18 Transferred to RO for re-assignment
S17 Sought help from Senior Field Representative / Regional Office
S12 Checked with neighbors

When designing and building the CHI system we try to anticipate the need for change and consider maintenance issues for all parties. The best way to think of this process is like a bank. Each deposit or withdrawal is a separate transaction. Every time CHI is run, a new CHI.bdb is created populated, transaction generated and the database is deleted. The instrument displays the screens and collects keystrokes, but the transaction processor Manipula script does the work to translate Blaise data into a flat ASCII file. Actually the output is a file that uses a combination of techniques to send data downstream to other control systems. We use a variety of ways to communicate with our partners; labels; delimited lists; fixed and variable length records or entries all constitute a transaction. It may seem complex to a casual observer to mix all these methods but the resulting transaction ends up being very neat and predictable. Another really great thing about CHI is that it has no memory, which further simplifies the process. Once the process is closed our job is done. We provide a lookup list to our partners with code and English translation (actual answer list text, slightly modified to remove @|@/@B@K etc) for each new release of CHI, making integration and upgrades simple. This is like the magic decoder ring that allows anyone else to decipher our transaction.

CHI Code Examples: Appendix C

- **Example: Blaise CHI.bla program (abstract)**
- **Example: Trans.man (for formatting ASCII)**
- **Example: Lookup List**
- **Example: ASCII Transaction file**

4. Preliminary Results and the Redesign for Longitudinal CHI

After evaluating returning CHI data for NHIS, originally presented by Andrea L. Piani shown in Appendix D, we opted to reduce the number of categories listed on the screens for several reasons. Though the field representatives at the initial focus groups requested we list all possible occurrences for each category of interest, plus allow for a "write in" other specify category, we found through data analysis that many were marked less than two percent of the time. We found that answers which were marked infrequently often occurred with slightly different wording as "other specify" entries. To make the objective clear to our field interviewers and supervisors, we decided to drop little used answers in favor of improved data density on the info pane.

During this initial review the CHI team also realized that one of our goals to collect data on and about potential refusals was not being met. Our interviewers, to their credit, were unwilling to "give up on the data collection effort" and viewed reporting a potential refusal as a bad thing to do; they avoided filling in answer categories for potential refusals. This was quite a quandary for us since we were trying to collect data which would help the field turn non-response or potential refusals into completed cases. I am not sure if this was interviewer bias, a training issue or something else. We needed to rethink our approach to getting the FRs to report accurately. We had a debriefing in preparation for implementing longitudinal CHI and found that both experienced FRs and SFRs agree this is a tool that will help them do their jobs better

Initially, some FRs viewed CHI as a method for headquarters to review their work – "Big brother watCHIng." However, when the data started coming in and enough was available to populate reports, they changed their view. The field staff has realized WIIFM "What's in it for me" and embraces this initiative. Reports like the FR Contact Attempt Report, which shows attempts by day of week and time of day, for example, became a helpful tool for Field to evaluate contact attempts for non-response cases "at a glance." Additional information is being gleaned for contacts, completed cases and workload distribution. The effect of additional assignments on FR work remaining, incomplete or non-contact cases also became apparent. This may be just the tip of the iceberg. As more data become available across surveys and for various survey types (reoccurring and longitudinal), I expect we will realize additional uses for CHI, ways to save time and money as well as hither to unexpected revelations. If we give the statisticians enough data to work with, some time to crunch the numbers and evaluate the results they may be able to come up with information to help us improve response rates, save time and reduce costs. I hope CHI will continue to be supported, improved and "cloned" by other interested agencies, private and public so that we all benefit from this work.

The National Health Interview Survey staff should be credited with having the initiative to accept the CHI challenge. Appendix D shows some of the tables that have been generated from CHI data. What started out as kind of an orphaned project, was adopted (requirements identified), visited the pediatrician (analysis), nurtured with care (designed), put through school (tested), graduated (placed in the field/implementation) and is now being redesigned (planned obsolescence) and it looks as if it will be used in 2005 for longitudinal surveys. In effect CHI has an advanced degree and is moving out on its own. The Census Bureau, the Technologies Management Office (TMO), Demographic Surveys Division (DSD), and Field Divisions (FLD) are supporting this effort to keep CHI generic and available across all surveys.

5. Vision for CHI

In the past Contact history has been collected for interviews either by memory, on paper notes, or other more automated techniques. Our best and senior field representatives have a good concept about response rates, ways to complete difficult interviews and to persuade respondents not to quit the survey; however, intuition is not enough. Part of the issue is that we need metrics about the contacts we make, non-contacts and the nature of reluctance, so that we can take a more scientific approach to understanding human nature, cultural concerns and individual issues surrounding survey response.

In summary, this type of information should be made available to Regional offices, senior, and rank and file interviewers to help reduce non-contacts. In many surveys, interviewers are given their entire assignments at the beginning of the field period – with previous call history information and a little planning, work efforts can be more efficiently spent on days and times when chances are best for contact.

Another hypothesis to explore is whether attrition households appeared disproportionately represented by one mode over another, personal visit versus telephone contacts. We want to look at those respondents who are 'on the fence' and are in need of refusal conversional or other special efforts to secure the interview.

For the future, we hope to provide better reporting of the CHI data. The initial focus was to collect the data for analysis by headquarters researchers. For the most part the Field staff is seeing more or less raw data. We would like to provide more detailed reports, such as identifying outliers or recommending a best time to attempt a case, that will be more useful to the interviewer.

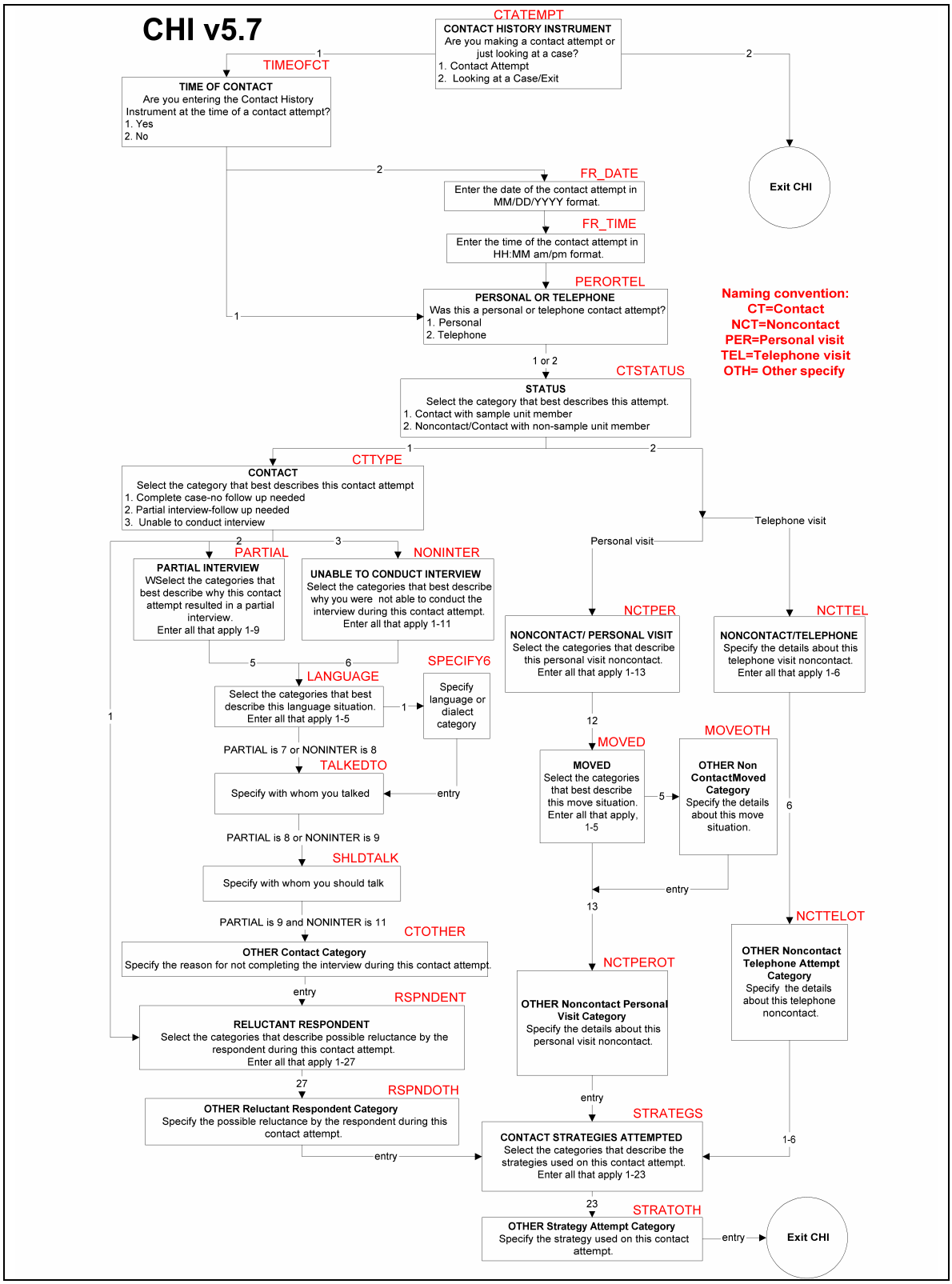
The CHI was used in production for the first time in the National Health Interview Survey, a cross-sectional survey conducted annually (see Piani, 2004). Currently, the original instrument is undergoing revisions based on interviewer feedback and analysis of data from NHIS.

Several other presentations and papers have been written based on the initial CHI. The Census Bureau has presented articles at the American Association for Public Opinion Research (AAPOR), (see Bates 2003, 2004), which give us a perspective and some direction to this project. Nancy Bates has been a staunch supporter of the CHI throughout the development process she and others have contributed to making CHI a success.

The next version of CHI is being developed with plans for use in the 2005 Consumer Expenditure Survey, the 2005 NHIS and possibly the Survey of Income and Program Participation (SIPP) Panel wave of interviewing occurring in 2006. A new enhancement to allow senior field representatives to view their team's CHI records is also in the works. We intend to implement the plan to use CHI for longitudinal surveys in such a way that previous interview cycle's contact records can be fed forward and be available to interviewers in later rounds of interviewing. We anticipate the expanded use of CHI will produce even richer datasets upon which to continue attrition research and expand our understanding of how contact histories may reduce it.

Appendix A

CHI Flow Chart



Our subject matter specialist specifies variable names, question text, navigation and coding schemes to facilitate development, testing and making change requests.

Appendix B

Contact History Instrument (CHI) V4.6
William E. Dyer, Jr. & Adriana Gonzalez

Why CHI?

A subgroup of the Interagency Household Survey Non-response Group (IHSNG) made a proposal to gather more information on the reasons behind refusals, no one homes and other type A non-interview outcomes. The Census Bureau developed a data collection instrument that collects information about every contact attempt made. The reengineered National Health Interview Survey (NHIS) will be the first survey to use the CHI in production.

The data from CHI are viewed in Case Management and ROSCO, providing FRs and survey supervisors with immediate feedback about a case's contact history. Survey supervisors and FRs may use this information to improve response rates by converting potential refusals into completed surveys.

Who will use the CHI?
(Senior) Field Representatives, Survey Supervisors, HQ, and sponsors.

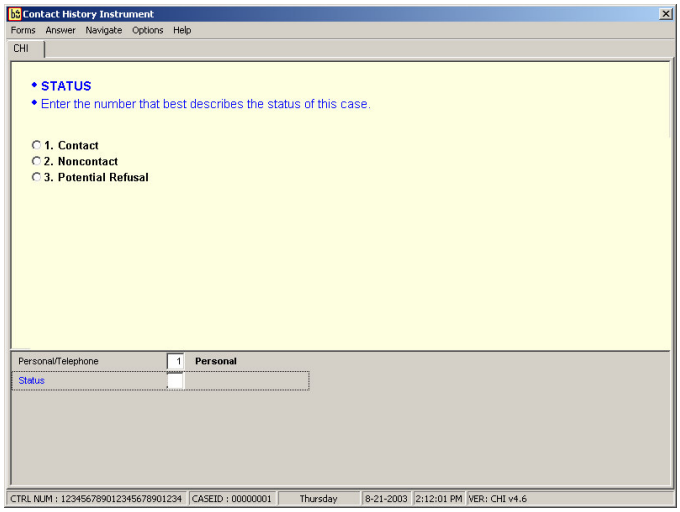


Figure 1: Establish the case status

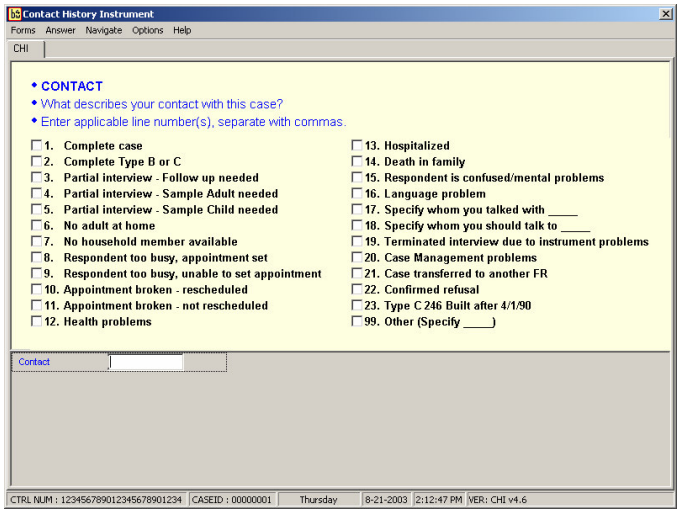


Figure 2: Contact Screen allows additional Specify __ entries

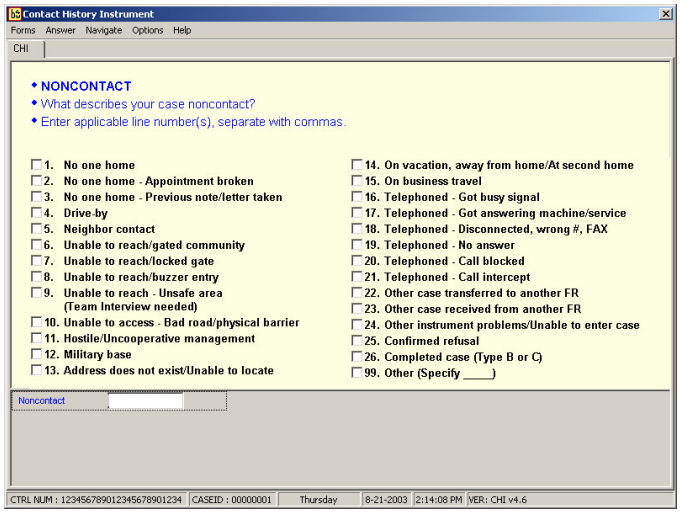


Figure 3: Non-contact answer list categories are standardized

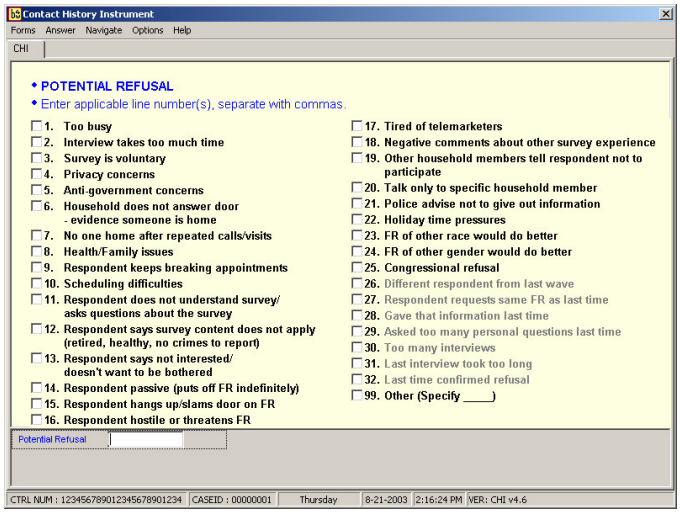


Fig 4: Potential Refusal (longitudinal survey categories in gray)

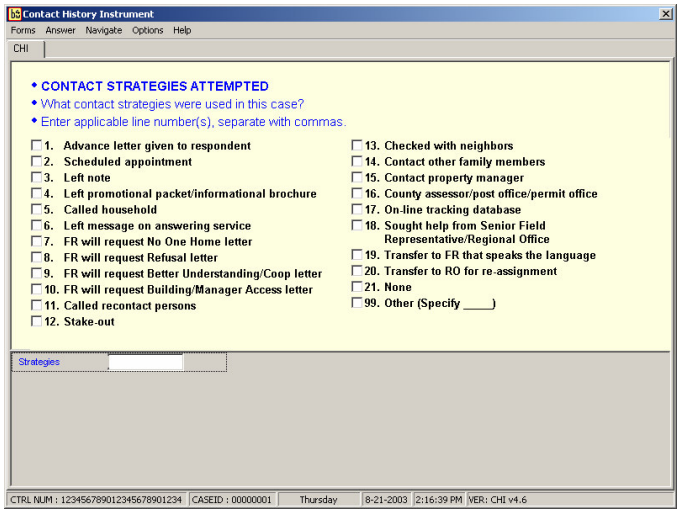
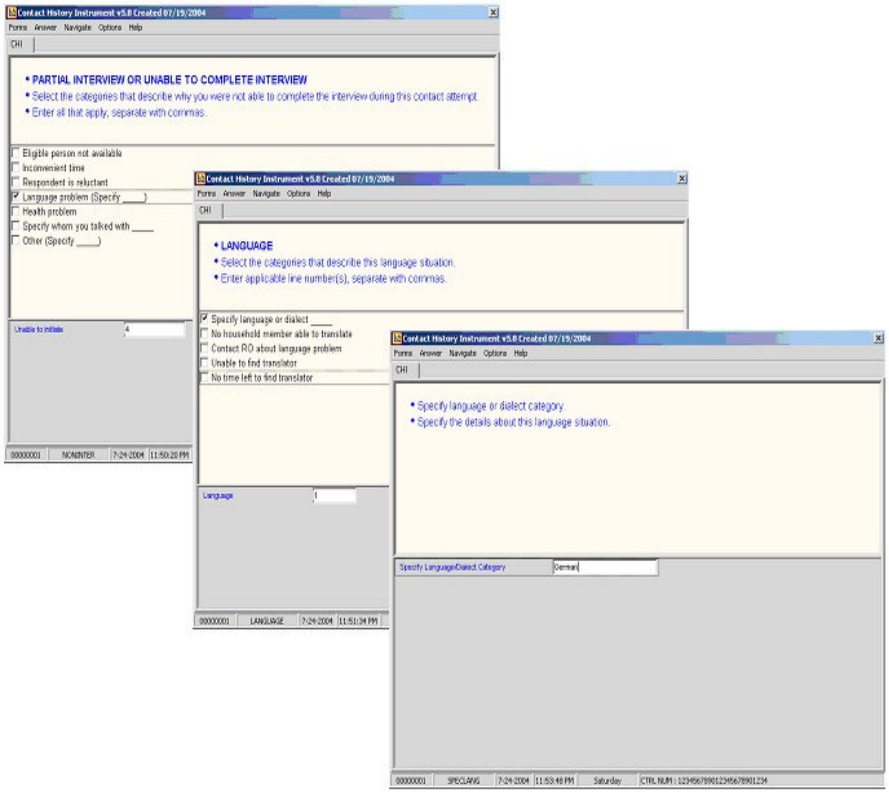


Figure 5: Contact Strategies Attempted (final CHI screen)

Case Management view from the original CHI showing contact type detail.

Training cases provide an example of the FR's view of CHI in Case Management



These screens are from CHI v5.8 and represent a series of instructions from the FR to log why they were unable to complete the interview for this contact attempt. The code example tries to follow this transaction example through the process.

Appendix C

Blaise Code examples do not have the complete syntax for recreation of the complete CHI application. However, this should give both new and experienced coders a peek into the process. The screens show an unable to complete interview, language problem, German scenario. I hope this code is enough to give you the idea of how the Contact History Initiative system functions.

- **Example: Blaise.bla programType, Field, Rules**

```
{Contact History Instrument
 AUTHOR: Ed Dyer
 DATE; 04/17/2002
 PROGRAM DESCRIPTION
  Collecting Case Characteristics for Interagency Household Survey Non-response Group
  07/16/2004 Revised v5.8 strawman additional adjustments and changes from ..
            Nancy Bates, Joseph Brunn, Ed Dyer, Valerie Loundon, Michael Lucero,
            Laura Nehilson, Dawn Nelson, Andrea Piani,
  07/19/2004 Revised v5.8 updates from flow and other requests - Michael Lucero and Ed Dyer.
*****************************************************************************}
DATAMODEL CHI "Contact History Instrument v5.8 Created 07/19/2004"

LANGUAGES =   ENG "English",
ATTRIBUTES = Dk, Rf, NOEMPTY
TYPE
   Trccother = STRING[80]
   Trcccontact =
            (C01  (1)  "Completed case - ready to transmit",
             C02  (2)  "Partial interview - follow-up required",
             C03  (3)  "Unable to conduct interview"
            )
   Trcclanguage = SET OF
            (L01   (1)  "Specify language or dialect _____",
             L02   (2)  "No household member able to translate",
             L03   (3)  "Contact RO about language problem",
             L04   (4)  "Unable to find translator",
             L05   (5)  "No time left to find translator"
            )
   Trccunable = SET OF
            (U01  (1)  "Eligible person not available",
             U02  (2)  "Inconvenient time",
             U03  (3)  "Respondent is reluctant",
             U04  (4)  "Language problem (Specify _____)",
             U05  (5)  "Health problem",
             U06  (6)  "Specify whom you talked with _____",
             U07  (7)  "Other (Specify _____)"
            )
FIELDS
   RCCC       (CTTYPE)
         Eng "@/@L@Zs@Z@B CONTACT @B
               @/@Zs@Z Select the category that best describes this contact attempt. @L "
            / "Contact"
            : Trcccontact

   RCCL       (LANGUAGE)
         Eng "@/@L@Zs@Z @BLANGUAGE@B
               @/@Zs@Z Select the categories that describe this language situation.
               @/@Zs@Z Enter applicable line number(s), separate with commas.@L@/"
            / "Language"
            :Trcclanguage

   RCCU      (NONINTER)
         Eng "@/@L@Zs@Z @BPARTIAL INTERVIEW OR UNABLE TO COMPLETE INTERVIEW@B
               @/@Zs@Z Select the categories that describe why you were not able to
               complete the interview during this contact attempt.
```

```
                @/@Zs@Z Enter all that apply, separate with commas.@L@/"
            / "Unable to initiate"
            :Trccunable

  specifylanguage    (SPECLANG)
        Eng "@/@L@Zs@Z Specify language or dialect category.
                @/@Zs@Z Specify the details about this language situation.@L@/"
            / "Specify Language/Dialect Category"
            :Trccother

  specifytalkto    (TALKEDTO)
        Eng "@/@L
                @/@Zs@Z Specify with whom you talked.@L@/"
            / "Talked with"
            :Trccother

  specifyunable    (CTOTHER)
        Eng "@/@L@Zs@Z @BOTHER Contact Category@B
                @/@Zs@Z Specify the reason for not completing the interview during this
                contact attempt.@L@/"
            / "Other Contact Category"
            :Trccother
RULES
{…}
{CONTACT ----------------------------------------------------}
  {Contact}
    IF ( CaseContact = Contact ) THEN
      RCCC
    ENDIF

  {Partial or Unable to initiate interview}
    IF (( CaseContact = Contact ) AND ( RCCC = C02 )) OR     {contact and partial }
      (( CaseContact = Contact ) AND ( RCCC = C03 )) THEN     {contact &unable to initiate}
      RCCU
    ENDIF

  {contact partial interview talk to}
    IF ( U06 IN RCCU ) THEN
      SPECIFYtalkto
    ENDIF

  {contact partial interview other category}
    IF ( U07 IN RCCU )THEN
      SPECIFYunable
    ENDIF

  {Language}
    IF ( U04 IN RCCU ) THEN
      RCCL
    ENDIF

    IF (L01 IN RCCL) THEN
      SPECIFYlanguage
    ENDIF
   ENDIF  {endif contact not empty}

{There is a section missing here for non-contact …}

{RELUCTANT ---------------------------------------------------}
   NEWPAGE
   IF ( CaseContact = Contact ) OR (U03 IN RCCU ) THEN       {ask for reluctance in contacts or u03}
     RCCR
     IF RCCR <> EMPTY THEN
       {reluctant respondent other}
       IF ( R23 IN RCCR ) THEN
```

```
          SPECIFYreluctant
        ENDIF
      ENDIF
    ENDIF


{STRATEGIES ------------------------------------------------}
  {always ask contact strategies attempted}
    NEWPAGE
    RCCS
    IF RCCS <> EMPTY THEN
      {attempt other}
      IF ( S23 IN RCCS ) THEN
        SPECIFYStrategy
      ENDIF
    ENDIF
{…}
 LAYOUT
      AT RCCC          INFOPANE INFOP15x11x01 FIELDPANE FP38x15x13 GRID Grid01x01 {CONTACT}
      AT RCCL          INFOPANE INFOP15x11x01 FIELDPANE FP50x26x24 GRID Grid01x01 {LANGUAGE}
      AT RCCS          INFOPANE INFOP15x11x02 FIELDPANE FP50x26x24 GRID Grid01x01 {ATTEMPTED}
      AT RCCU          INFOPANE INFOP15x11x01 FIELDPANE FP50x26x24 GRID Grid01x01 {UNABLE}

      AT SPECIFYtalkto      INFOPANE INFOP10X02X01 FIELDPANE FP50x26x24
      AT SPECIFYreluctant  INFOPANE INFOP10X02X01 FIELDPANE FP60x35x24
      AT SPECIFYlanguage  INFOPANE INFOP10X02X01 FIELDPANE FP60x35x24
      AT SPECIFYStrategy   INFOPANE INFOP10X02X01 FIELDPANE FP60x35x24
ENDMODEL
```

- **example.man (for formatting ASCII)**

```
{Contact History Instrument Transaction program
 ... Interagency Household Non-Response Group
 AUTHOR: Ed Dyer
 DATE: Started back in 04/17/2002 with NHIS

 PROGRAM DESCRIPTION
   This is the transaction driver for CHI }

 MODIFICATIONS {…}

PROCESS Run_Interview

SETTINGS
  DESCRIPTION = 'CHI Transaction processing'
  {AUTOREAD = NO } { Comment this out for MANIPLUS }

USES
 Interview_instrument  'CHI'  { Meta file (i.e. bmi file for the survey) }

DATAMODEL TRANSACTION  { TRANSACTION record }
                       { Also used for logging transactions}
  FIELDS
    CASEID              : String [8]
    BLANK1              : String [1]
    CONTROL_NUM         : String [24]
    TRANS_CODE          : String [2]
    OUTCOME_CODE        : String [3]
    AGENDUM_CODE        : String [2]
    FLDREINT_FLG        : String [1]
    INSTRUMENT          : String [8]    {.bmi file }
    UPDATE_CASE         : String [20]   {capi_casemgt_in Manipula script name}
    UPDATE_CASEMGT      : String [20]   {capi_casemgt_out Manipula script name}
    LOG_TRANS           : String [1]    {If '1', transaction will be written to log file}
    BLANK2              : String [1]
```

```
ENDMODEL

DATAMODEL  outfilefmt
FIELDS
  CCList : STRING[1000]
ENDMODEL

UPDATEFILE    {The survey database}
 UpFile : Interview_instrument ('CHI', BLAISE)
SETTINGS
   CONNECT = NO
   OPEN = NO {Gets opened in the manipulate section}

INPUTFILE     {The TRANSACTION file that case management creates}
 InTRANS : TRANSACTION ('trans_file', ASCII)
  SETTINGS
   CONNECT = NO

OUTPUTFILE
  Output : outfilefmt('CHI_flat.asc', ASCII)
  SETTINGS
    MAKENEWFILE = YES

{Error trapping and AUXFIELDS omitted …}
{... Start CHI the low-response non-response questionnaire procedure}
PROCEDURE RunNRQ
INSTRUCTIONS

 IF FILEEXISTS('CHI.bmi') THEN
   {CHI databases are created on the fly by CASEID and deleted by casemanagement at the end}
   AUXDBNAME :=  THECASE + 'CHI.bdb'
   Upfile.OPEN(AUXDBNAME)
   Upfile.READNEXT
   Upfile.INITRECORD
   Upfile.CHI.CASEID := InTRANS.CASEID
   Upfile.CHI.CONTROL_NUM := InTRANS.CONTROL_NUM
   UpFile.WRITE
   Upfile.RELEASE

   Reslt2:= edit ('CHI.bmi' + ' /N ' + ' /K' + THECASE +
              ' /M..\Config\CHI.bwm' + ' /F' + AUXDBNAME + ' ' +
              ' /G /X')
 ENDIF
{Error trapping omitted …}
ENDPROCEDURE
{... End Procedure}

PROCEDURE RunBLAISTOASCII
INSTRUCTIONS

Upfile.RELEASE
Upfile.open(AUXDBNAME)
Upfile.READNEXT

I := 1

output.open(THECASE+'.CHI')      {write out data to a transaction record}
                                 {Setup CATI-CAPI, CaseContact, Date, Time, Personal-or-Telephone, etc …}
 CCList := '#caseid'
 output.write
 Output.CCLIST := THECASE
 output.write
{Omit contact status – detail labels …}

 Output.CCLIST := ''
```

```
                                    {format from CHI program}

{Contact}
 IF ( UpFile.CHI.CaseContact = Contact ) THEN
    tmpfld := '0' + str(UpFile.CHI.RCCC)
    Output.CCLIST := Output.CCLIST + 'C' + tmpfld + '$^'
 ENDIF

 IF (UpFile.CHI.CaseContact = Contact) AND (Upfile.CHI.RCCC = C02) THEN

   FOR ctr := 1 to UpFile.CHI.RCCU.CARDINAL DO
     IF (UpFile.CHI.RCCU[ctr]  < 10 ) THEN                {make sure the codes are all three characters}
       tmpfld := '0' + str(UpFile.CHI.RCCU[ctr])
     ELSE   { (RCCu[ctr]  > 9 ) then}
       tmpfld := str(UpFile.CHI.RCCU[ctr])
     ENDIF

     Output.CCLIST := Output.CCLIST + 'U' + tmpfld + '$^'

     {Language}
     IF ( UpFile.CHI.CaseContact = Contact ) AND ( UpFile.CHI.U04 IN UpFile.CHI.RCCU ) THEN
        FOR ctr := 1 to UpFile.CHI.RCCL.CARDINAL DO
          IF (UpFile.CHI.RCCL[ctr]  < 10 ) then
            tmpfld := '0' + str(UpFile.CHI.RCCL[ctr])
          ELSE
            tmpfld := str(UpFile.CHI.RCCL[ctr])
          ENDIF

          Output.CCLIST := Output.CCLIST + 'L' + tmpfld + '$^'

          IF UpFile.CHI.RCCL[ctr]  = 1 THEN  {Other Language}
            Output.CCLIST := Output.CCLIST + '#' + UpFile.CHI.SPECIFYlanguage + '$^'
          ENDIF
        ENDDO
     ENDIF

     IF UpFile.CHI.RCCU[ctr] = 4 THEN  {specify language unable shared with language}
       Output.CCLIST := Output.CCLIST + '#' + UpFile.CHI.SPECIFYlanguage + '$^'
     ENDIF

     IF UpFile.CHI.RCCU[ctr] = 6 THEN  {spcify talk to unable}
       Output.CCLIST := Output.CCLIST + '#' + UpFile.CHI.SPECIFYtalkto + '$^'
     ENDIF

     IF UpFile.CHI.RCCU[ctr] = 7 THEN  {specify other unable}
       Output.CCLIST := Output.CCLIST + '#' + UpFile.CHI.SPECIFYunable + '$^'
     ENDIF
   ENDDO
 ENDIF
{
{Noninterview - Unable to initiate interview} {EAJ}
{code for this section omitted …}

 linelen := LEN(Output.CCLIST)
 lastchars := SubString(Output.CCLIST,linelen-1,2)
 {display(lastchars,wait) }
 IF lastchars = '$^' THEN
   Output.CCLIST :=  SubString(Output.CCLIST,1,linelen-2)
 ENDIF
 {display(Output.CCLIST,wait)}
output.write
output.release
ENDPROCEDURE
```

```
MANIPULATE

 InTRANS.OPEN
 InTRANS.READNEXT    {Read the TRANSACTION file }

 WHILE NOT (InTrans.EOF) DO

  THECASE       := InTrans.caseid          {CASEID}
  {... Call the Contact History Instrument (CHI)  ... Transaction code '08'}
  IF InTRANS.TRANS_CODE = '08' THEN   {SFR observed case update set FLDREINT from trans file}
    RunNRQ
    RunBLAISTOASCII
  ENDIF
  InTRANS.READNEXT    {Read the TRANSACTION file }
 ENDWHILE
```

- **Example: Part of the lookup code list provided to CM, ROSCO & HQ**

C00 Contact
C01 Completed case - ready to transmit
C02 Partial interview - follow-up required
C03 Unable to conduct interview"
L00 Language
L01 Specify language or dialect _____
L02 No household member able to translate
L03 Contact RO about language problem
L04 Unable to find translator
L05 No time left to find translator
S00 Strategies
{omitted}
S12 Checked with neighbors
{omitted}
S17 Sought help from Senior Field Representative / Regional Office
S18 Transferred to RO for re-assignment
{omitted}
U00 Unable to initiate
U01 Eligible person not available
U02 Inconvenient time
U03 Respondent is reluctant
U04 Language problems (Specify ____)
U05 Health problem
U06 Specify whom you talked with _____
U07 Other (Specify _____)

- **Example: ASCII Transaction file**

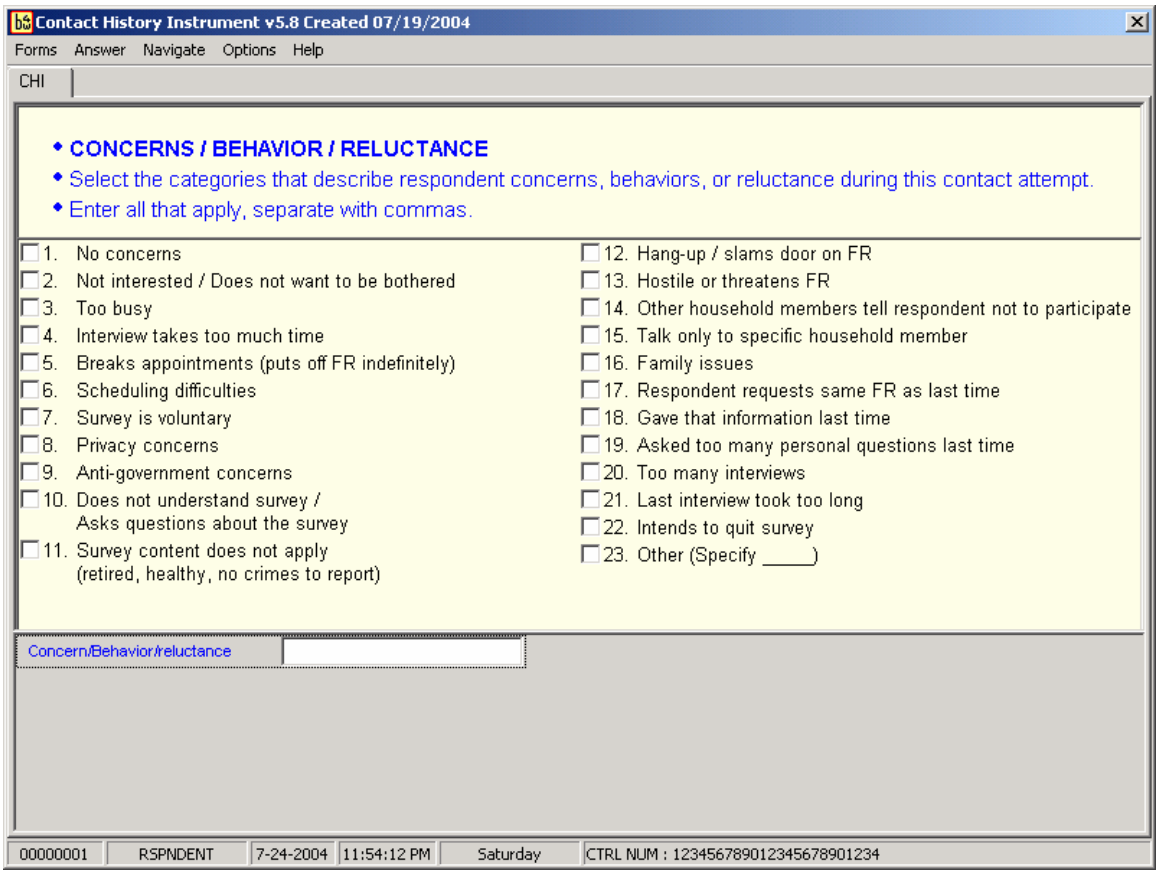
Following is resulting transaction file for Case Management (CM).

#caseid
00000001
#frdate
07/26/2004 11:13 AM
#howcontacted
P
#contactstatus
C
#detail
CU

#types
C03$^U04$^L01#German
#strat
S18$^S17$^S12

This transaction will remain in this format for all the subsystems CM and ROSCO it will eventually be returned to HQ for review by the survey analysts. This method compresses code, standardizes our approach, saves resources both for storing the data on the laptop hard drive & regional office servers and minimizes the transmission of CHI data through the system. This format is similar to our other control system interfaces, does not require retraining and extensive reprogramming and retains the order of the entries as can be seen in the strategy category. FR's are trained to enter the most important items first.

The following screen images were taken from the current CHI v5.8 and show part of the redesign effort. This is just to give the reader an idea how exhaustive the entries are.



Contact History Instrument v5.8 Created 07/19/2004

Forms Answer Navigate Options Help

CHI

- **CONTACT STRATEGIES ATTEMPTED**
- Select the categories that describe the strategies used on this contact attempt.
- Enter all that apply, separate with commas.

- [ ] 1. Advance letter given
- [ ] 2. Scheduled appointment
- [ ] 3. Left note / appointment card
- [ ] 4. Left promotional packet / informational brochure
- [ ] 5. Called household
- [ ] 6. Left message on answering machine
- [ ] 7. FR will request No One Home letter
- [ ] 8. FR will request Refusal letter
- [ ] 9. FR will request Better Understanding letter
- [ ] 10. Called contact persons
- [ ] 11. Stake-out
- [ ] 12. Checked with neighbors
- [ ] 13. Contacted other family members
- [ ] 14. Contacted property manager
- [ ] 15. Visited county assessor / post-office / permit office
- [ ] 16. On-line tracking database
- [ ] 17. Sought help from Senior Field Representative / Regional Office
- [ ] 18. Transferred to RO for re-assignment
- [ ] 19. Offered incentive
- [ ] 20. CED double placement
- [ ] 21. Used MAF or ALMI
- [x] 22. None
- [ ] 23. Other (Specify _____)

Strategies attempted: 22

00000001	STRATEGS	7-24-2004	11:55:09 PM	Saturday	CTRL NUM : 12345678901234567890 1234

Appendix D

CHI Reports

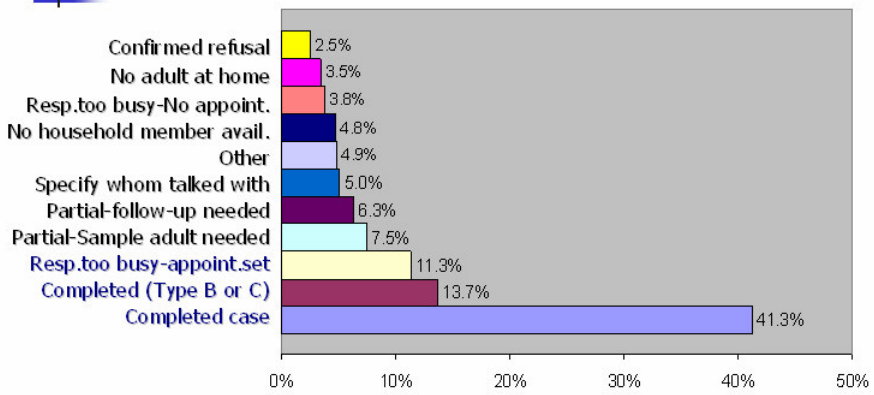
Andrea L. Piani, Gave a very interesting presentation about the use of data from Contact History Initiative (CHI), at FEDCASIC in March of 2004. With her permission we are able to include a few summary tables for your review. These represent a sample of the tables and graphs that can be generated once CHI data is collected and analyzed.

Percent distribution of contact attempts by final outcome: NHIS Q1W3-W4

Final outcome	1	2-3	4-6	7-10	11+	Total	N
Completed	27	41	22	8	2	100	1372
Sufficient partial	10	29	34	20	7	100	264
TypeA noninterviews	9	23	31	20	17	100	223
Insufficient partial	17	25	33	8	17	100	24
No one home	6	15	26	31	22	100	54
Temporarily absent	0	27	27	27	18	100	11
Refused	11	22	33	19	15	100	109
Other	4	40	36	4	16	100	25
TypeB noninterviews	54	33	10	3	0	100	376
Screened out	42	39	13	5	1	100	572
TypeC noninterviews	58	31	7	3	1	100	197

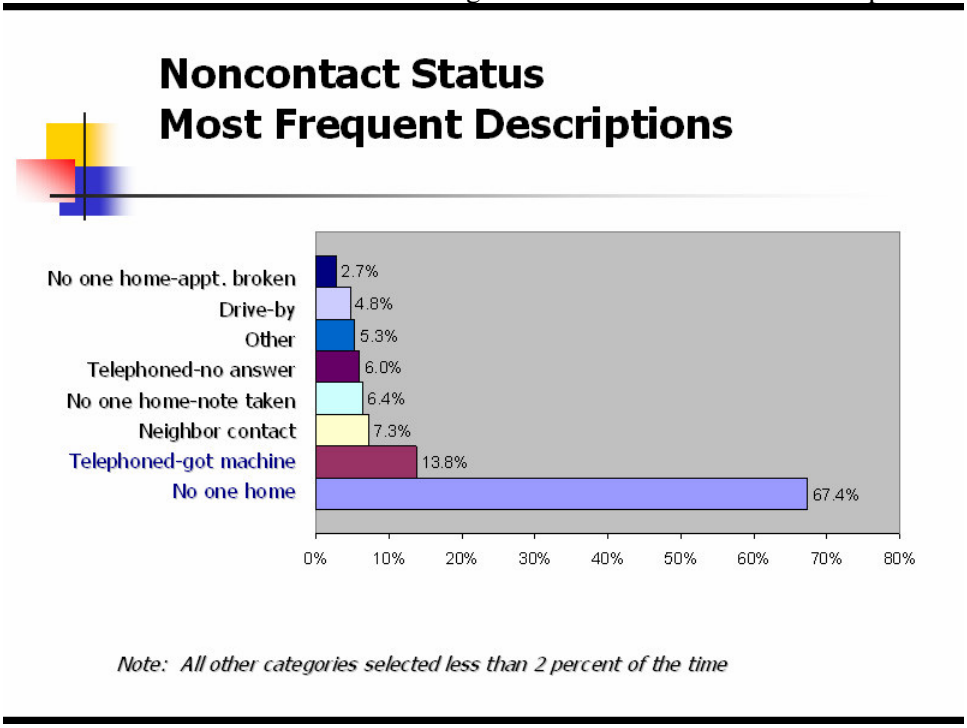
Type-A non-interviews require many contact attempts by the FR

Contact Status Most frequent descriptions

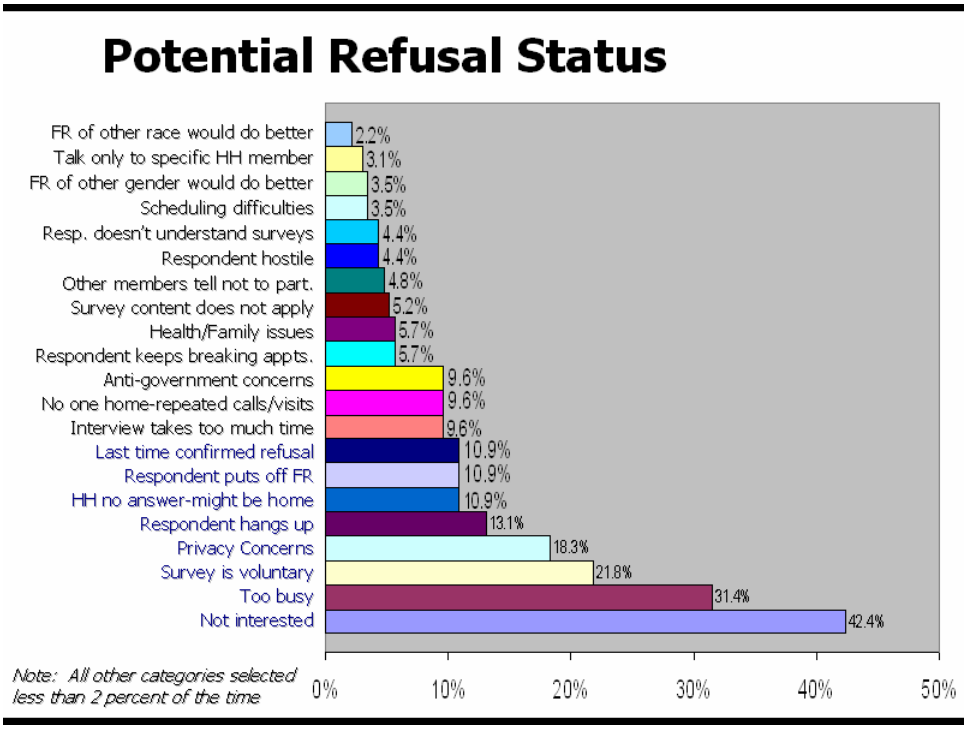


Note: All other categories selected less than 2 percent of the time

After two weeks of interviewing 55% of the interviews were completed.



The most common non-contact occurrence was "No one home" 67%



Metrics gathered by the Contact History Initiative for the National Health Interview Survey during January 12-28, 2004 confirm that the top five reasons given to interviewers for potential refusal (by reluctant respondents) are: Not interested; Too busy; Survey is voluntary; Privacy concerns and hang-up. What if anything can be done to improve our interviewers chances of getting a complete interview. Now that we have a baseline the SFR may look at printed reports, suggest other times to visit or in time Field Division may use this data to change procedures to educate the respondents, shorten the interviews or provide incentives. This is just the beginning, as more data is available we

will refine the CHI to meet Field Division's requirements to collect and process contact history data.

Contact History Reports are generated as required in the Regional Offices for supervisory review of survey contact attempts. ROSCO generates a number of standard reports. Many are - Census Confidential Title 13 – so we can not give examples without performing disclosure. However, the **"Type of Contact by Time of Attempt"** report, which does not show any individual respondent information is a good example of the way CHI data is summarized for our Field Division and supervisory interviewers.

NHIS Contact History Instrument

Type of Contact by Time of Attempt Report

Interview Periods: 2003210

Outcome Type	Contact Status	Total Attempts		Time of Attempt											
				6:00 - 8:59 AM		9:00-11:59 AM		12:00 - 2:59 PM		3:00 - 5:59 PM		6:00 - 9:59 PM		0:00 - 11:59 P	
		M-F	W/E	M-F	W/E	M-F	W/E	M-F	W/E	M-F	W/E	M-F	W/E	M-F	W/E
201	Contact	364	39	11	0	99	16	155	18	71	3	21	1	6	0
	Non Contact	18	3	0	2	8	0	4	0	3	1	2	0	1	0
	Total by Outcome	**382**	**42**	**11**	**2**	**107**	**16**	**159**	**18**	**74**	**4**	**23**	**1**	**12**	**0**
	First Attempt - Final Outcome	259	27	6	0	66	13	127	10	47	3	13	1	0	0
203	Contact	126	5	4	0	39	2	48	1	24	2	5	0	6	0
	Non Contact	13	2	0	1	1	0	5	0	6	1	1	0	0	0
	Potential Refusal	1	0	0	0	0	0	0	0	1	0	0	0	0	0
	Total by Outcome	**140**	**7**	**4**	**1**	**40**	**2**	**53**	**1**	**31**	**3**	**6**	**0**	**6**	**0**
	First Attempt - Final Outcome	43	3	0	0	16	2	20	0	5	1	1	0	1	0
A's	Contact	141	10	3	0	41	5	65	4	17	0	6	0	5	1
	Non Contact	288	16	4	0	71	6	94	10	46	0	24	0	17	0
	Potential Refusal	13	1	0	0	7	0	3	0	0	0	1	0	0	1
	Total by Outcome	**442**	**27**	**7**	**0**	**119**	**11**	**162**	**14**	**63**	**0**	**31**	**0**	**25**	**2**
	First Attempt - Final Outcome	281	19	2	0	59	9	103	10	40	0	23	0	20	0
B's	Contact	14	1	0	0	3	1	5	0	2	0	1	0	2	0
	Non Contact	62	4	1	1	18	0	19	2	16	1	5	0	0	0
	Total by Outcome	**76**	**5**	**1**	**1**	**21**	**1**	**24**	**2**	**18**	**1**	**6**	**0**	**2**	**0**
	First Attempt - Final Outcome	49	4	0	1	14	1	15	2	10	0	6	0	0	0
C's	Contact	86	4	1	0	32	1	26	3	15	0	10	0	1	0
	Non Contact	199	18	6	4	34	3	56	0	46	10	30	0	25	0
	Potential Refusal	8	0	0	0	4	0	2	0	1	0	0	0	0	0
	Total by Outcome	**293**	**22**	**7**	**4**	**70**	**4**	**84**	**3**	**62**	**10**	**40**	**0**	**43**	**0**
	First Attempt - Final Outcome	117	2	0	0	11	0	51	1	35	0	15	0	3	0

Titles of other CHI Reports available to NHIS field supervisors:

NHIS Contact History Summary Report
- (CHI 1) Cases Not Checked In
- (CHI 2) Type A Report

NHIS Contact History Instrument
- (CHI 3) Partial Report (Outcome 203)
- (CHI 4) Number of Attempts Report
- (CHI 5) Type of Contact by Time of Attempt Report
- (CHI 6) Strategies Report (cases w/at least 1 Potential Refusal)
- (CHI 7) Number of Attempts Report by FR
- (CHI 8)Strategies Report (cases w/ at least 1 Potential Refusal) by FR
- (CHI 9) FR Contact Attempt Report

References

Andrea L. Piani, The use of Field Metrics and Paradata (Process Data) in evaluating CASIC Surveys, CAPI History Data Contact History Initiative (CHI), presentation for FEDCASIC 2004 Workshop TA-3 at the Bureau of Labor Statistics, March 4, 2004

William E. Dyer, Jr. and Adriana Gonzalez, CHI v4.6 Why CHI? Prepared for Field Day, U.S. Census Bureau, August 27, 2003.

Laura Nehlson, CHI Flow Chart V5.7, U.S. Census Bureau

Nancy Bates, Contact Histories in Personal Visit Surveys: The Survey of Income and Program participation (SIPP) Methods Panel U.S. Census Bureau, Demographic Surveys Division, 3384-3, Washington, DC 20233 A paper presented at the annual conference of the American Association for Public Opinion Research (AAPOR), Nashville, Tennessee May 15-18, 2003

Nancy Bated, Contact Histories as a Tool for Understanding Attrition in Panel Surveys
U.S. Census Bureau, Demographic Surveys Division, 3384-3, Washington, DC 20233 A paper presented at the annual conference of the American Association for Public Opinion Research (AAPOR), Phoenix, Arizona May 13-16, 2004

Dawn V. Nelson, Charter for the Recurring & Multi-mode Surveys Contact History Committee, U.S. Census Bureau, February 11, 2003.

Michigan CATI Sample Management System (SMS) – Blaise CATI Enhanced

Jim Hagerman & Karl Dinkelmann, Survey Research Center, University of Michigan

1. Overview of the CATI SMS

The Survey Research Center (SRC) at the University of Michigan began implementing its Blaise CATI SMS in 2001. This was on the heals of the development of SRC's proprietary CAPI sample management system, SurveyTrak, that transmits sample data, versions of the instrument, and information on call attempts and sample outcomes back and forth from the decentralized staff and the central office.

The experience SRC gained from its previous CATI systems and SurveyTrak brought on the notion of adding features to the existing CATI system to create a robust and enhanced CATI SMS version. In contrast to the decentralized sample management system, the CATI SMS had different requirements such as: the interviewers sharing sample lines and working predefined shifts; the sample is pooled and managed by sample types and attributes, e.g., RDD and Recontact subsamples, Language, interviewer experience, refusal conversion required, "hard" versus "easy" case workgroups. All of these requirements brought forth the need for a different network-based system. However, the two sample management systems share the same reporting database, which allows for the implementation of the Dynamic Reporting System (DRS). DRS will be discussed briefly at the end of this paper.

In the SMS version implemented in 2001, interviewers could maintain sample line information; telephone number; type of sample (recontact, new sample); time zone, strata; collect data on contact attempts; results of each call; record interviewer notes and observations; and allowed for Supervisor review and sample assignment. However, after using the first generation CATI SMS system it was determined that several shortcomings needed to be addressed. One of the principal shortcomings was that sample assignment was executed through paper coversheet distribution (Pennell, et al. 2004). This meant significant Supervisor effort, including determining: lines to contact the next day, managing number of calls on a sample line, manually assigning specific types of sample to different interviewers (by language, refusal conversion, appointment follow-up, etc.). It was also labor intensive to focus interviewer effort on particular strata. Therefore, it was decided that the next generation of the CATI SMS system should implement the Blaise Call Scheduler to allow for automatic case delivery.

Other second-generation SMS issues that needed to be addressed were the ability to manage sample distribution and survey costs while adding flexibility to the overall architecture of the enhanced version of the CATI SMS II system. It was determined to manage the sample distribution more effectively, the following priorities were identified (Pennell, et al. 2004):

- Hierarchical priorities, including hard appointments, soft appointments, last call busy;
- Sorting based on study-specific requirements, including strata, states, or time zones;
- Managing to quotas, for example by recontact cases;
- The ability to mange appointments by initially distributing sample to the interviewer setting the appointment, then distributing to the general or group interviewer pool.

Other items impacting the management of survey costs were (Pennell, et al. 2004):

- Having the ability to automatically prioritize calls by their likelihood to contact. In addition, appointments needed to be kept and busy calls followed up on, even if the calling interviewer gets another respondent on the line;

- Automated distribution of cases with no contact based on preset call windows, would guarantee a balanced effort on the case;

- Automated limits on number of contact attempts per sample line; meeting contract cost / effort specifications.

Adding flexibility to the overall architecture of CATI SMS II meant all the features existing in the current CATI SMS would be retained, this includes: Supervisor review of cases and bringing up a sample line by Sample ID or telephone number. Additionally, by working with groups, Supervisors can assign specific types of cases to specific interviewers; for example, flagging cases for refusal conversion. The fully integrated second-generation CATI SMS II system is currently in its testing phase and is expected to be in production by fall 2004.

The current CATI SMS system does not employ the Blaise CATI "Dial Screen" nor will it be used in CATI SMS II. The "INHERIT CATI" requirement has been separated from the Interview data and is instead associated with a smaller CATI-only related datamodel used exclusively for CATI management. The CATI SMS also features two separate and distinct modules; one for the Supervisor and one for the Interviewer.

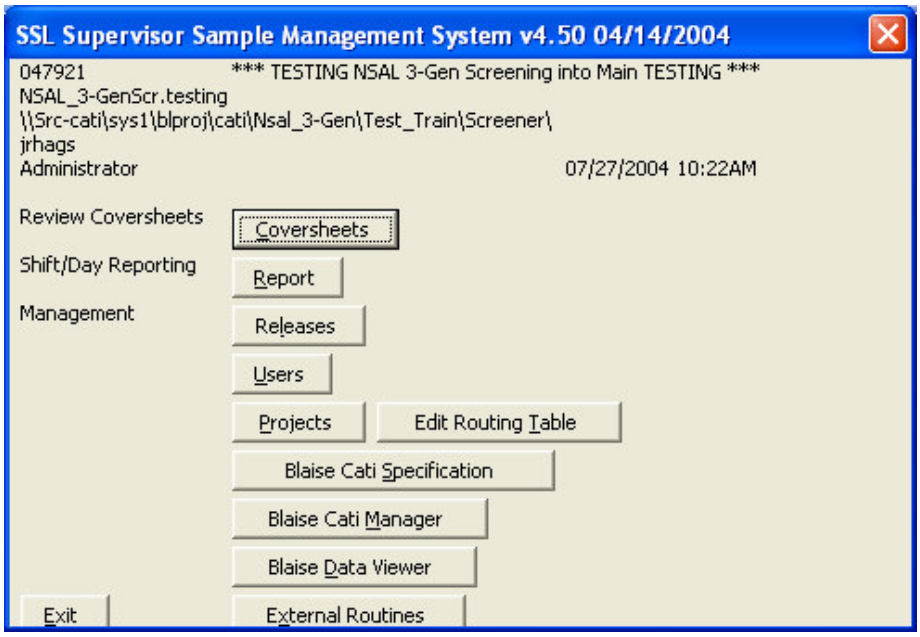
The system has the advantage of permitting standardized reports to be produced across all studies which eliminates the need to build a separate set of reports for each project. There are also a detailed set of Supervisory Management tools that allow Supervisors to manage projects, result codes, and users (Interviewers).

The system includes the capability of running external routines and provides for a set of "Special Calls" that can run Manipula scripts prior to the Coverscreen, prior to the Interview, and after the Interview has been completed. The main portion of this paper will review the CATI SMS II features as they interact with Blaise CATI. First, an exploration of the supervisory module functions will be discussed, then briefly discussing the interviewer module, and ending with a section discussing the Dynamic Reporting System (DRS).

2. SMS Supervisor Module

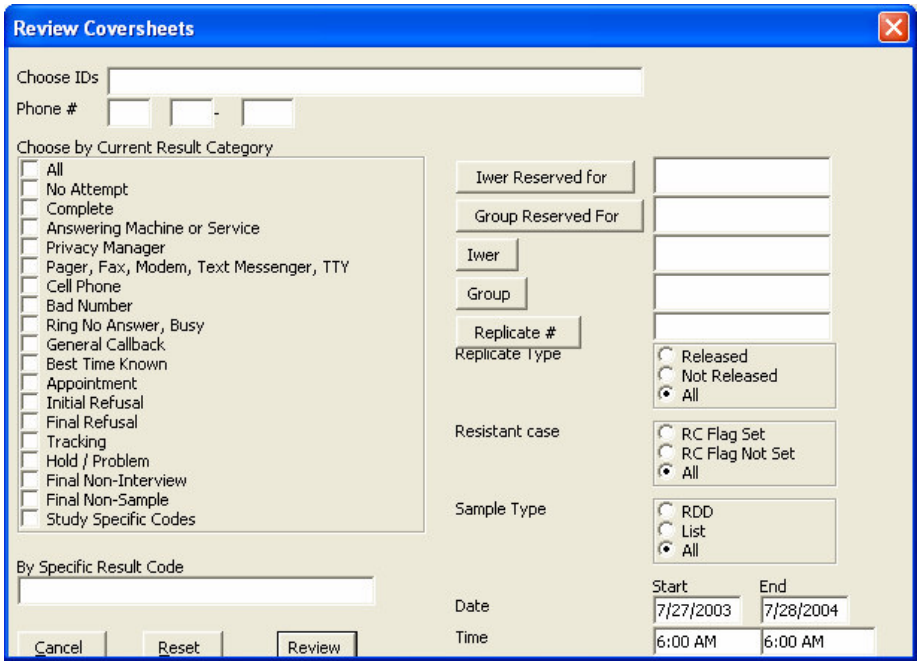
The Supervisor (SMSSuper) module provides a variety of features for Project Managers, Team Leaders, and Shift Supervisors to manage sample lines and interviewers. A login screen prompting for username and password permits entry into the Supervisor module.

Once a project has been selected, a main dialog screen appears. From this screen, many management functions can be performed and these are discussed below.



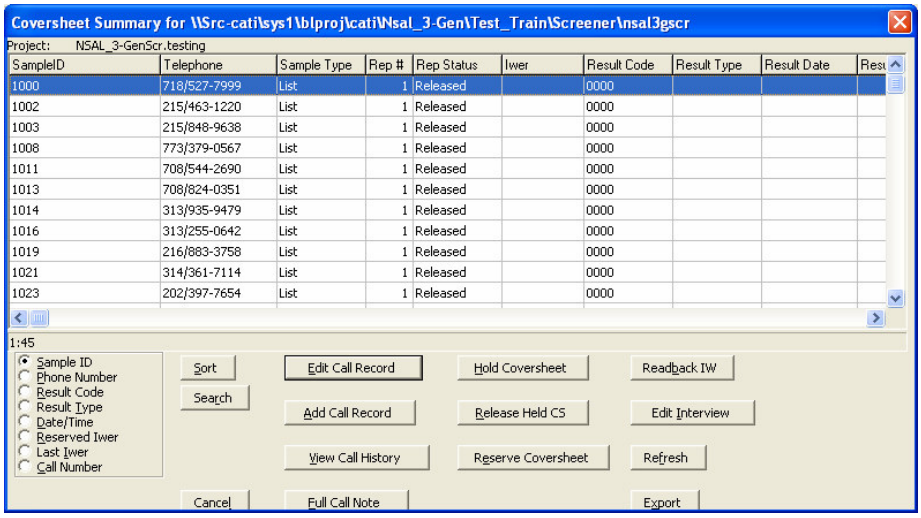
2.1 Main Dialog: Coversheets

Upon selecting the "Coversheets", the Supervisor may review the coversheets by many different parameters. They may select cases for review by Sample ID, Telephone Number, Result Code, Interviewers, Groups, Replicate Types, Resistance Type, and Sample Type, as well as by Date and Time.



Once criteria have been selected, the resulting screen displays those sample lines. Globally, the user may sort the lines by Sample ID, phone number, result code, result type, date/time, reserved interviewer (a case reserved for a specific interviewer), last interviewer, and call number.

On any particular sample line, the user may "Edit Call Record," "Add Call Record," "View Call History," review "Full Call Note," "Hold Coversheet," "Release Held CS," "Reserve Coversheet," "Readback IW," "Edit Interview," "Refresh" (refresh the current view), and "Export" (exports the data in the view to a comma separated file).

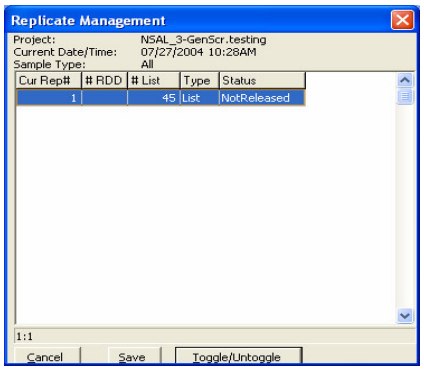


2.2 Main Dialog: Reports

Due to the current restructuring of the SMS –Reporting utility, report functions are discussed briefly at the end of this paper.

2.3 Main Dialog: Releases

SMS manages releases (replicates) in two parts, handled through the same interface. There is a master list of releases stored in the system for the study, and each case has a release number and a flag whether the case has been released. Setting the flag on individual cases is important for automatic case delivery because the Blaise delivery algorithms are based upon flags and other status for each case. Note: because a release may happen during production interviewing, the release management is conducted in "shared mode" (SMS2002 Supervisor Manual, V.1, 2003).

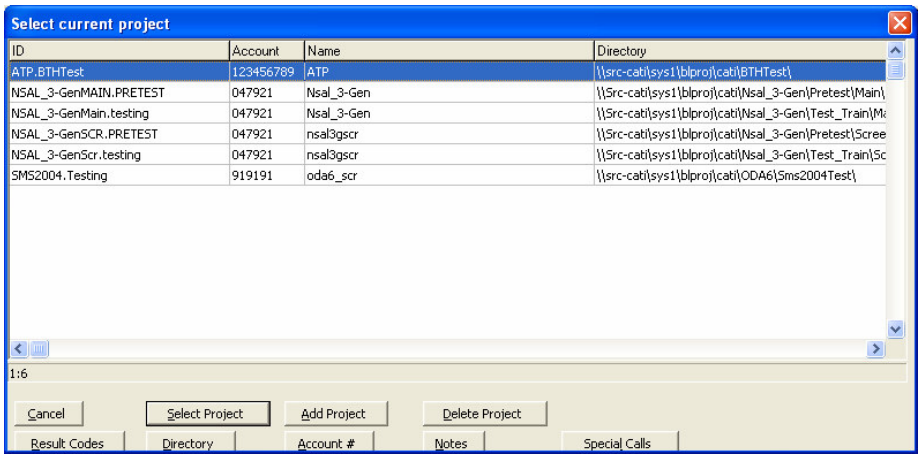


2.4 Main Dialog: Users

The management of all users in the system is handled by the "Users" button; bringing up a list of users and other project-specific information. Features available from the User dialog screen are: manage groups, add new groups, add or remove users.

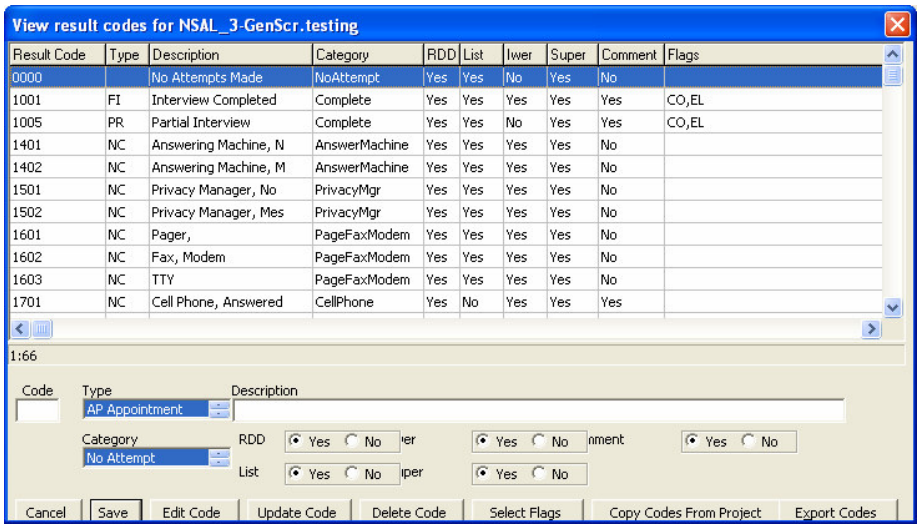
2.5 Main Dialog: Projects

A list of projects is presented to the user for selection. Also on this screen, additional functions can be performed such as "Add Project," "Delete Project," "Result Codes," updating the directory path of the project, update the account number, enter notes specific to the project, and "Special Calls".



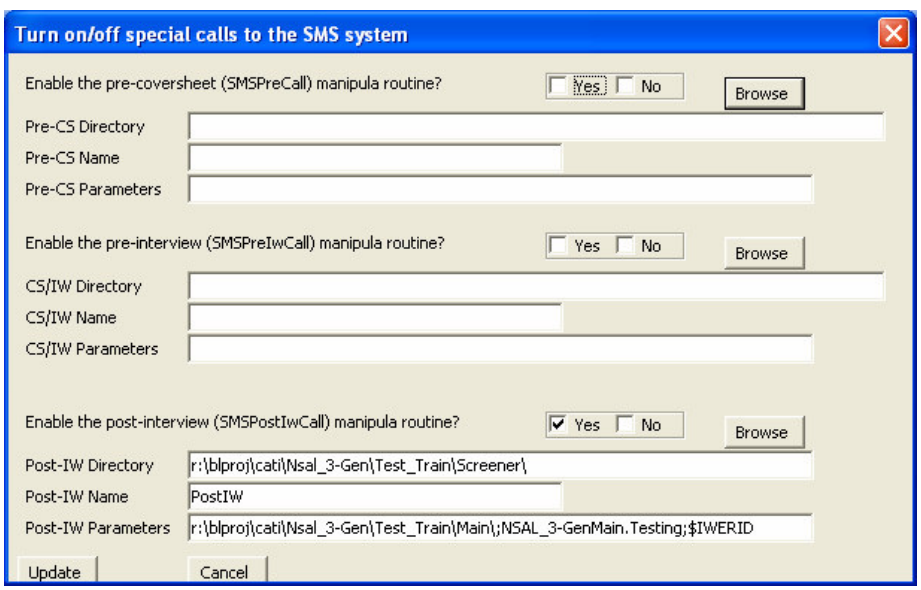
2.5.1 Main Dialog: Projects: Result Codes

The Result Codes interface allows the user, at the start of a project, to select which result codes, from a standardized list, they wish to include.



2.5.2 Main Dialog: Projects: Special Calls

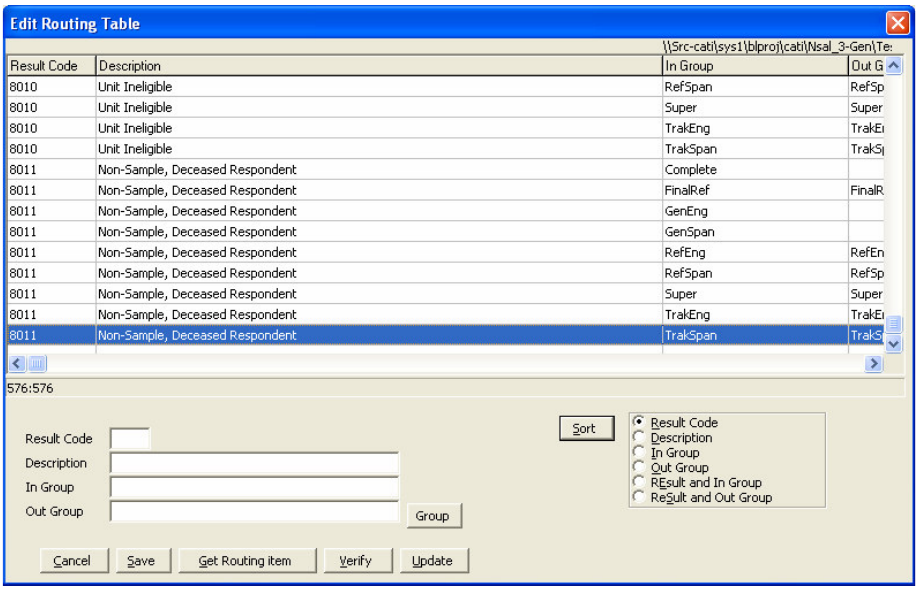
"Special Calls" allows the user (CAI Programmer) to select from three different types of Manipula routines; "pre-coversheet" (this feature is not available with automatic case delivery via the call scheduler), "pre-interview," and "post-interview" depending on the needs of the project at hand.



The example above displays the set-up for a "post-interview" Manipula routine. The directory where the routine resides is declared; the name of the routine, and finally the specified set of parameters being passed into the routine.

2.7 Main Dialog: Edit Routing Table

The Routing Table concept was provided to SRC by Statistics Canada. This new feature in CATI SMS II allows for the routing of cases to various groups of interviewers. This routing is dependent on the most recent result code for the case.



2.8 Main Dialog: Blaise CATI Specification

This provides direct access to the CATI spec file for an individual project using the standard Blaise CATI Spec Editor.

2.9 Main Dialog: Blaise CATI Manager

This also provides direct access to the Blaise CATI Manager for each individual project.

2.10 Main Dialog: Blaise Data Viewer:

The data viewer allows the user to browse through Blaise databases (and other data) via SMSSuper. There are three kinds of viewing: browse coversheet**,** browse interviews**,** browse other data.

2.11 Main Dialog: External Routines

The External Routines button gives the option to run supplementary programs via SMSSuper. There are three options to run programs (SMS2002 Supervisor Manual, V.1, 2003):

- Browse for executable programs: Executable programs are like Notepad, Word, Excel, and other special programs.

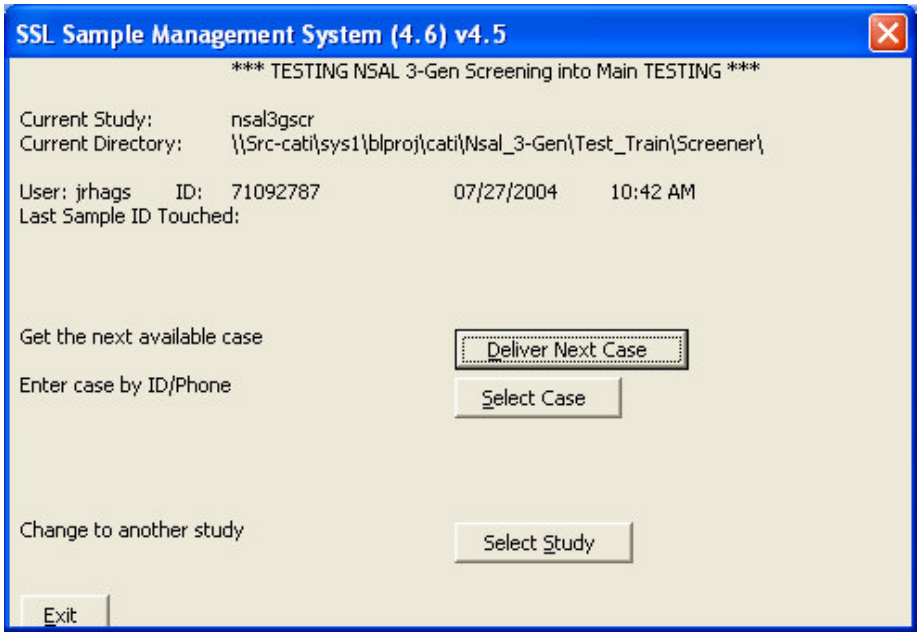
- Browse for Manipula scripts: Similar to browsing for .EXE programs, this is designed to run prepared Manipula/Maniplus programs (.MSU extension).

- Browse other program types: Ability to open other file types that are not EXEs and MSUs.

3. SMS Interviewer Module

The SMS Interviewer (SMSCati) module provides the interviewer with direct access to their daily work. As seen in the Supervisor's module, the interviewer is presented with a login screen prompting for username and password; this permits entry into the Interviewer module. They are then provided with a list of projects that they are allowed to access:

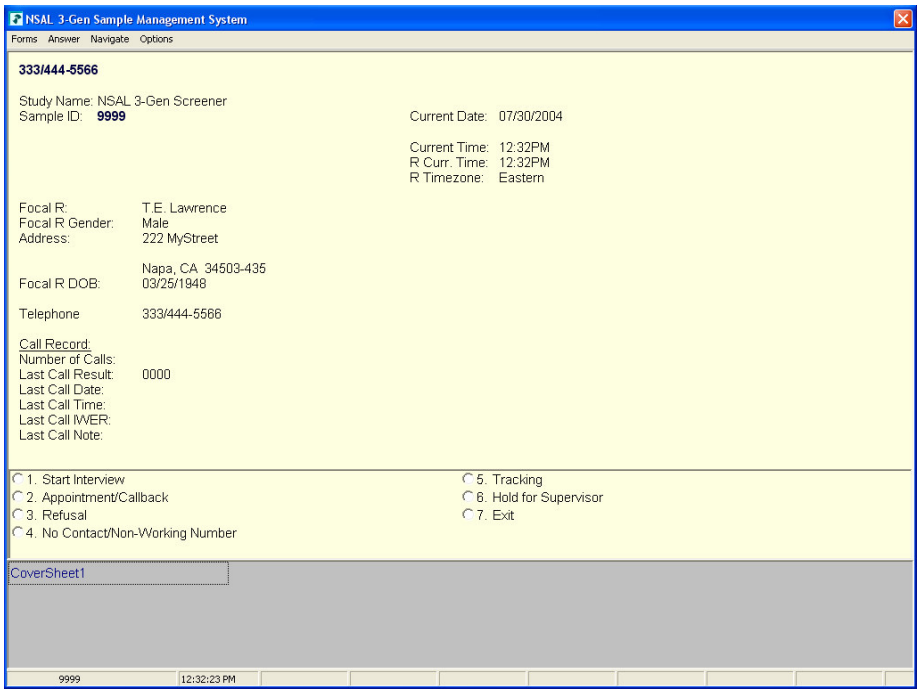
The SMSCati Main dialog provides the interviewer with more than one option by which to access sample lines; it also includes a variety of information that the interviewer can utilize. The "Deliver Next Case" option allows an interviewer to access cases that have been given priority by CATI SMS II.



The "Select Case" button allows access to a specific case by either Sample ID or Telephone Number. The default is by Sample Id.

3.1 Accessing the Coverscreen

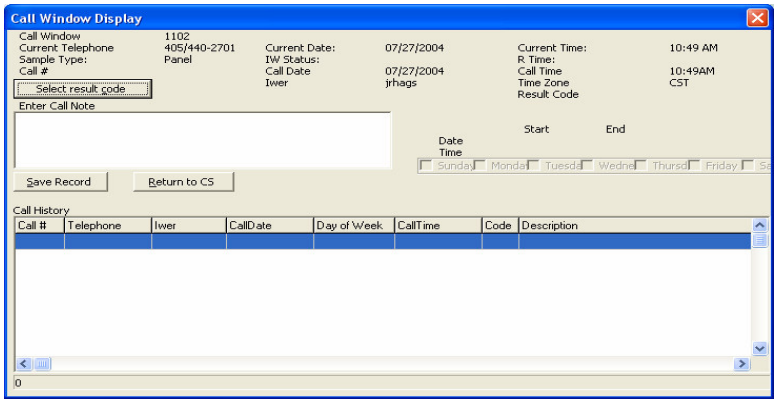
This screen appears when a case has been accessed via either the "Deliver Next Case" or "Select Case" commands. The Coverscreen includes a variety of information about the respondent as well as other information that will help the interviewer develop a calling strategy for the sample line (SMS2002 Supervisor Manual, V.1, 2003). This additional information includes the Sample ID, date, current time, and the respondent's current time, as well as further information relative to the composition of the respondent's household. This information can be very helpful to an interviewer in their preparation for calling a particular case.



Depending on the response, the interviewer will either conduct the interview or enter a call record and result code for the sample line.

3.2 Call Window Display

The "Call Window Display" provides the Interviewer with the ability to record a result code and call record for the sample line. At the same time, they can see the call history for the sample line.



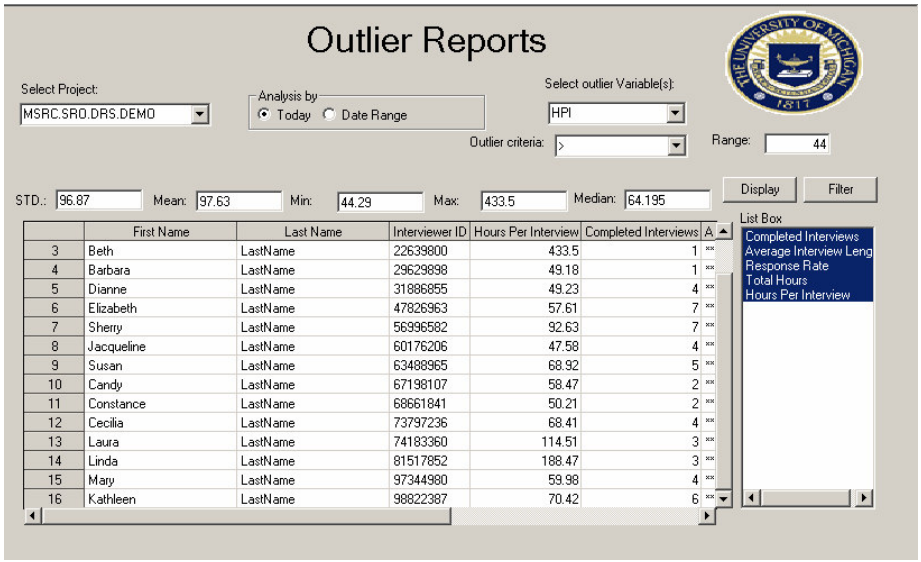
4. Dynamic Reporting System (DRS)

The DRS is intended to integrate CATI SMS II and SurveyTrak reporting databases into a single reporting database. It will create analytical reports in addition to descriptive reports, serving multiple user types: field managers, project managers; and principal investigators. DRS will ultimately enable further independent analysis of process data by simplifying the output procedure (Pennell, et al., 2004).

DRS will have several ways of outputting reports including the creation of reports for the following issues: outliers, trends, key statistics, snapshots, and standard reports. These are explained briefly below with a screen shot of how these reports will look:

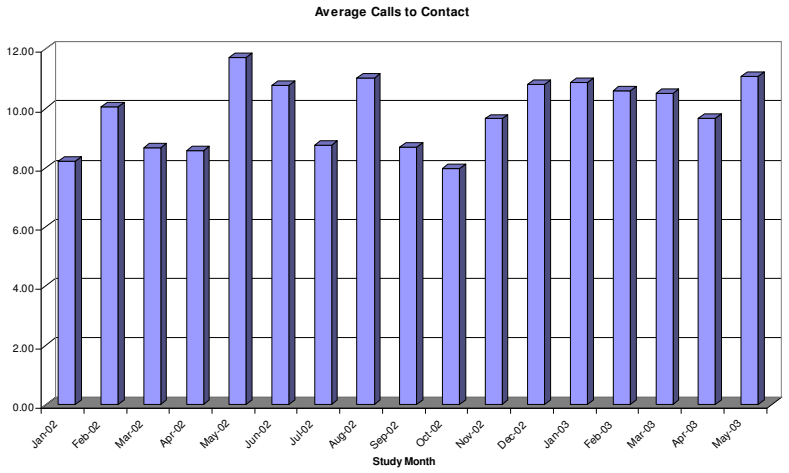
4.1 DRS: Outlier Reports

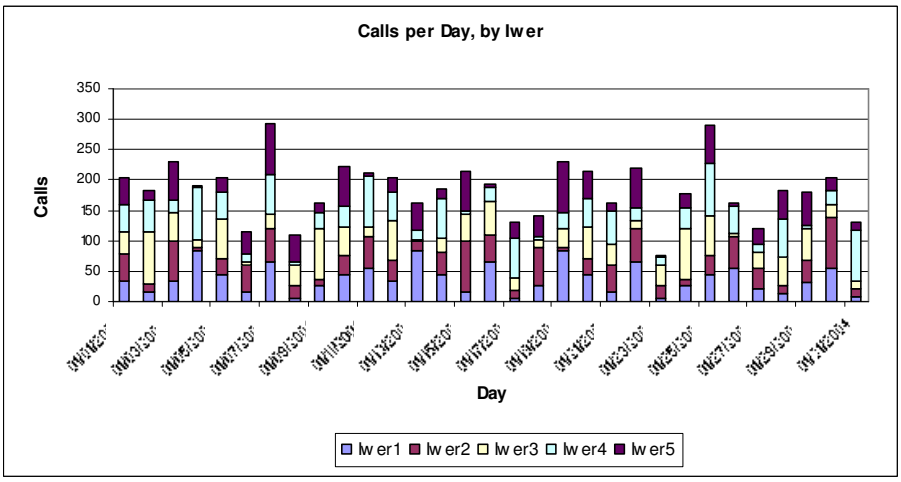
Interviewer level data to identify which interviewers perform at exceptional levels (for good or bad) on key performance characteristics



4.2 DRS: Trend reports

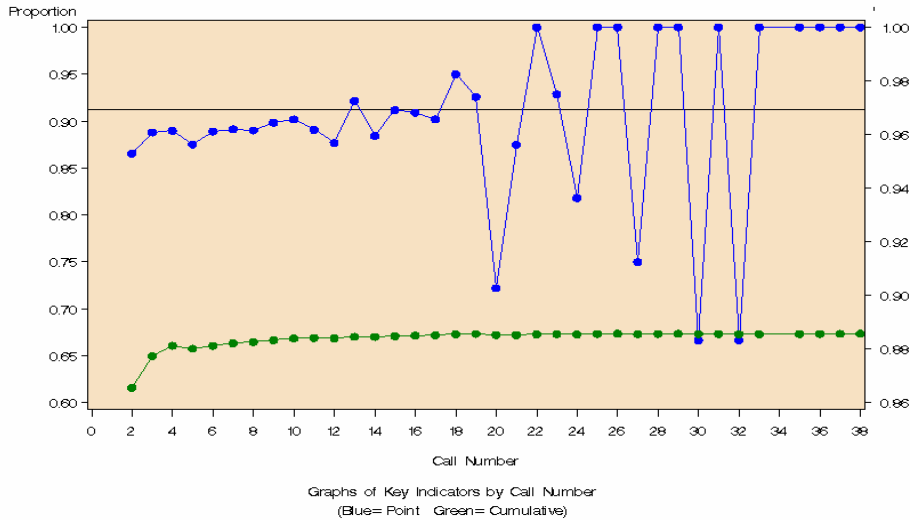
This report outputs interviewer level data to determine trends in sample development at different points of a study, the ability to select all datasets. (Additional variables include Minutes between calls, Verification type results)





4.3 Key statistics reports

Call level data to uncover potential bias.



4.4 Snapshot reports:

Interviewer level data enabling the analysis of work patterns at specified time frames.

4.5 Standard Reports:

Standard Field Progress Report (FPR); Client FPR; Standard Cost Report; Client Cost Report; Standard Verification Report; Client Verification Report; etc.

5. References:

Pennell, B.E., Benson, G., Chardoul, S., Cheung, G-Q. (2004). "Technical Developments in SRO". June Primary Research Staff Seminar, Survey Research Center, University of Michigan.

SMS2002 Supervisor Manual, V.1. (2003). Manual produced by the Technical Services Group, Survey Research Operations, Survey Research Center, University of Michigan.

The Progamma Library of Blaise Routines

Dirk Sikkel, Sixtat, The Netherlands

1. Introduction

The Progamma Library of Blaise Routines contains a number of block and procedures that are in a way awkward, given the original philosophy of Blaise. This awkwardness has different causes:

1. *Randomization*. This is a problem because after each change in the data Blaise recalculates all its fields, including those which contain random numbers
2. *Binary data storage of set questions* (multiple response questions). This has not been included in the basic data structure of Blaise
3. *Procedures for marketing research of scientific research*. Such procedures have not been emphasized as Blaise was developed in an official statistics environment.

Part of the 'legacy' of Interuniversitair Expertise Centrum Progamma has been used to deal with these problems. The software that has been developed by Sixtat handles the need for randomization, binary storage and marketing research procedures in the best possible way, although some solutions still may seem artificial to the end user. The source of the Progamma Library is available to Blaise users. They are encouraged to adapt the software to their needs. No support by either IEC Progamma or Sixtat is available. Those who will use the randomization procedures should read section 4, in which two problems are described which have cost the author a lot of time.

2. Contents of the library

The Progamma Library of Blaise Routines contains the following elements.

PROCEDURE SortInd. Sorts an index array of integers.
Typical application: show a sorted list of strings to the respondent.

BLOCK BRandomize. Generates a permutation of the numbers 1, 2, ..., n.
Typical application: randomization of a set of strings, to be used for randomization purposes.

BLOCK BToss. Flip a coin (unbiased)
Typical application: create two equally sized random groups each of which answers a different version of the questionnaire.

BLOCK BBiatoss. Flip a coin (biased)
Typical application: create two unequally sized random groups each of which answers a different version of the questionnaire.

PROCEDURE Decompose. Decomposes a string into an array of substrings.
Typical application: storage of a number of a number of descriptions of response categories in one large string.

BLOCK BRandAns. Generate a question with response categories in random order.
Typical application: presentation of questions of which it is suspected that the response is influenced by the order of the response categories.

BLOCK BRandSet. Generate a random set question (multiple response question) with response categories in random order and write the answers in binary format.
Typical application: presentation of set questions of which it is suspected that the response is influenced by the order of the response categories.

BLOCK BBinSet. Writes the answers to a set question (multiple response question) as binary data to the data file.
Typical applications: (1) when all answers have to be inspected for subsequent calculation or routing this is possible by a simple for-loop; (2) in data analysis the results can be used immediately for e.g. multiple response analyses in SPSS.

PROCEDURE CompStr. Match open answers with known strings.
Typical application: recognition of products and brands in open answers, which can be used for subsequent routing or calculations.

BLOCK BGaborg. Measures purchase intentions for different price levels for a single product.
Typical application: calculation of demand curves of a brand or product without considering the prices of competing brands. This is known as the Gabor-Granger procedure.

BLOCK BBPTO. Measures product preferences for different price levels.
Typical application: calculation of effects of reactions of competitors of a given product.
This is known as the Brand Price Trade Off procedure.

3. Description of the individual routines

In order to use the Programma Library, the source of the questionnaire should contain the following statement

```
INCLUDE "Progamma_Library.inc";
```

This makes available the following types, which are used by the library

```
TIntArr = array[1..30] of integer;
TPermutation = array[1..30] of 1..30;
TOrder = (asc, des);
TCoin = (head, tail);
TAnsArr = array[1..30] of string[80];
TQuestTx = string[1200];
TChar = string[1];
TProdArr = array[1..12] of string[80];
TPricArr = array[1..12] of real;
```

The bounds of the arrays are in a way arbitrary, and may be changed by the user. He should, however, be aware that by doing so he also changes the limitations within the different routines.

PROCEDURE SortInd

Purpose

Sorts an index array of type TPermutation of integer array Arr of type TIntArr

Syntax

```
SortInd(Arr, Ind, n_items, Order)
   Arr: TintArr, array of integers to be sorted
   Ind: TintArr, index array; after sorting Arr[Ind[.]] is an ordered
        sequence of integers
   n_items: integer, number of elements of Arr to be sorted
   Order: Torder, if Order=asc: sort ascending;
                  if Order=des: sort descending
```

Application
SortInd is a method for showing sorted lists of strings, when the order has to be based on answers of the respondent.

BLOCK BRandomize

Purpose
Generates a permutation of the numbers 1, 2, ..., n_items, stored in an array of type TPermutation.

Syntax

```
Randomize(Ind, Inv, n_items).
   Ind: array that contains the permutation
   Inv: array that contains the inverse permutation, such that Inv[Ind[i]]=i
   n_items: number of elements of Ind to be permutated
```

Application
BRandomize can be used for indexing a series of strings in random order. These strings can be used for generating questions with text variables, which appear in random order or for a set of possible responses which appear in random order. The array Ind can be used to determine the answer in the original order. The array Inv contains additional information about the rank order of given texts. When Ind and Inv are defined as fields, this infomation is stored. When they are displayed as auxfields, the information about order in the questionnaire is lost.

Limitation
Maximum size of permutation is 30; this can be increased by changing all numbers 30 to a higher number (also in the type-definition of TPermutation!)

Example

```
DATAMODEL IndInv;
INCLUDE "Progamma_Library.inc";

LOCALS
  i: integer;
  Older: array[1..12] of string[100];

FIELDS
  IndOld, InvOld: TPermutation;
  Randomize: BRandomize;
  QOlder {example of questions of a single type in random order}
     "Please indicate how much you agree with the following statement
     @/@/^Older[IndOld[i]]
     @/@/1 means: completely disagree
     @/5 means: completely agree":
  array[1..3] of 1..5;

RULES
  Older[ 1]:='I feel very responsible for my loved ones';
  Older[ 2]:='I feel myself more responsible for others than when I was younger';
  Older[ 3]:='I am just as care-free as when I was 18';
  Randomize(IndOld, InvOld, 3);
  for i:=1 to 3 do QOlder[i] enddo;
```

BLOCK BToss

Purpose
Generate two equally likely branches in a questionnaire

Syntax

Toss(Nickel)
  Nickel: Tcoin, where Nickel is equally likely head or tail

Application
Toss can be used for the random creation of two equal groups that answer different questions

Example

```
DATAMODEL TossUnBiased;
INCLUDE "Progamma_Library.inc";
FIELDS
  Nickel: TCoin;
  Toss: BToss;
  NickHead "Nickel is head": string[1];
  NickTail "Nickel is tail": string[1];
RULES
  Toss(Nickel);
  if Nickel=head then NickHead else NickTail endif
```

BLOCK BBiaToss

Purpose
Generate two branches in a questionnaire with unequal likelihood

Syntax

```
BiaToss(Nickel,p)
   Nickel: Tcoin, where Nickel is has probability p to be head and probability 1-
p
           to be tail
   p: real number between 0 and 1
```

Application
Toss can be used for the random creation of two unequal groups that answer different questions

Example

```
DATAMODEL TossBiased;
INCLUDE "Progamma_Library.inc";
FIELDS
  Nickel: TCoin;
  BiaToss: BBiaToss;
  NickHead "Nickel is head": string[1];
  NickTail "Nickel is tail": string[1];
RULES
  BiaToss(Nickel);
  if Nickel=head then NickHead else NickTail endif
```

PROCEDURE Decompose

Purpose
Decomposes a string into an array of substrings, based on a user-specified separator.

Syntax

```
Decompose(Ans, AnsArr,c,m)
   Ans: TQuesTxt, the input string of length 1200 (can be modified)
   AnsArr: TAnsArr, the array[1..30] of string[80], the array of substrings
   c: Tchar, the separator between two substrings
   m: integer, the number of substrings (output parameter)
```

Application
In progamma_library Decompose is used for the construction of a variable number response categories out of a single string. Possibly, users may find other applications.

Example

```
Decompose ('French,German,English,Dutch',language,',',m )
```

Which yields
Language[1]=French,
Language[2]=German,
Language[3]=English,
Language[4]=Dutch
m=4

BLOCK BRandAns

Purpose
Generate a question with response categories in random order.

Syntax

```
RandAns(QuesTxt, AnsTxt, n)
   QuesTxt: TQuesTxt, the string[1200] that contains the question text.
   AnsTxt: TQuestTxt, the string[1200] that contains the texts of the response
          categories, separated by comma's
   n: integer, the number of response categories
```

The (original) number of the given answer is written to the field RandAns.Answer.
The separator between the response categories may be changed by the user, by changing the value of separator, e.g.

Separator:='_'

It may happen that the order of the last one or two response categories has to fixed, e.g. in the case of *don't know* or *not applicable*. This can be achieved by specifying n be smaller than the actual number of answers. If AnsTx contains m separate texts, then the last m-n answers are not randomized. If n is specified as 0, the number of response categories is determined within BRandAns.

Application
BRandAns is a closed question with possible answers in random order.

Example

```
DATAMODEL RandSing;
INCLUDE "Progamma_Library.inc";

LOCALS
  Color: array[1..7] of string[10];

FIELDS
  Prefer: BRandAns; {single answer, random order}
  ShowPref "The best liked color has (original) number ^Prefer.Answer and is
    ^Color[Prefer.Answer]": TContinue;

RULES
  Color[1]:='Yellow'; Color[2]:='Red'; Color[3]:='Blue'; Color[4]:='Green';
  Color[5]:='Purple'; Color[6]:='Aqua'; Color[7]:='Magenta';
  Prefer('What color do you like best?','Yellow,Red,Blue,Green,Purple,Aqua,
          Magenta',0); {separator is comma (default)}
  ShowPref;
```

Limitations
BRandAns allows for a maximum of 30 response categories; this can easily be changed by the user by

- adding response categories to BRandAns.Questio
- modifying the bounds in the block Brandomize

BLOCK BRandSet

Purpose
Generate a random set question (multiple response question) with response categories in random order.

Syntax

```
RandSet(QuesTxt, AnsTxt, n)
   QuesTxt: TQuesTxt, the string[1200] that contains the question text.
   AnsTxt: TQuestTxt, the string[1200] that contains the texts of the response
          categories, separated by comma's
   n: integer, the number of response categories
```

The given answers are written to the array [1..30] of 0..1 RandSet.Answer. Thus, they are coded binary, and in the original order as corresponds to AnsTxt.
The separator between the response categories may be changed by the user, by changing the value of separator, e.g.

Separator:='_';

It may happen that the order of the last one or two response categories has to fixed, e.g. in the case of *don't know* or *not applicable*. This can be achieved by specifying n be smaller than the actual number of answers. If AnsTxt contains m separate texts, then the last m-n answers are not randomized. If n is specified as 0, the number of response categories is determined within BRandSet.

Example

```
DATAMODEL RandMult;
INCLUDE "Progamma_Library.inc";

LOCALS
  i: integer;
  Color: array[1..7] of string[10];
  LikeCol: string;

FIELDS
  Like: BRandSet; {multiple answer, binary data, random order}
  ShowLike "The liked colors are ^LikeCol": string[1];

RULES
  Color[1]:='Yellow'; Color[2]:='Red'; Color[3]:='Blue'; Color[4]:='Green';
  Color[5]:='Purple'; Color[6]:='Aqua'; Color[7]:='Magenta';
  Separat:='_';
  Like('Which colors do you
like?','Yellow_Red_Blue_Green_Purple_Aqua_Magenta_none
        of these',7); {separator changed to underscore}
  LikeCol:='';
  for i:=1 to 7 do
    if Like.Answer[i]=1 then LikeCol:= LikeCol + ' '+Color[i] endif
  enddo; {adressing answers in a loop!}
  ShowLike;
```

Limitations
BRandSet allows for 30 response categories. This can easily be changed by the user by

- adding or deleting response categories in BRandSet.Questio
- modifying the bounds in the block BRandomize
- changing the dimension of BrandSet.Answer

BLOCK BBinSet

Purpose
Writes the answers to a set question (multiple response question) as binary data to the data file.

Syntax

```
BinSet(QuesTxt, AnsTxt)
   QuesTxt: TQuesTxt, the string[1200] that contains the question text.
   AnsTxt: TQuestTxt, the string[1200] that contains the texts of the response
          categories, separated by comma's
```

The given answers are written to the array [1..30] of 0..1 BinSet.Answer. Thus, they are coded binary.
The separator between the response categories may be changed by the user, by changing the value of separator, e.g.

separator:='_';

Example

```
DATAMODEL BinMult;
INCLUDE "Progamma_Library.inc";

LOCALS
  i: integer;
  Color: array[1..7] of string[10];
  HateCol: string;

FIELDS
  Hate: BBinSet; {multipe answer, binary data, fixed order}
  ShowHate "The hated colors are ^HateCol": TContinue;

RULES
  Color[1]:='Yellow'; Color[2]:='Red'; Color[3]:='Blue'; Color[4]:='Green';
  Color[5]:='Purple'; Color[6]:='Aqua'; Color[7]:='Magenta';
  Separat:='/';
  Hate('Which colors do you hate?',
    Color[1]+'/'+Color[2]+'/'+Color[3]+'/'+Color[4]+'/'+Color[5]+'/'+Color[6]+'/'
     +Color[7]+'/none of these/do not know');
  HateCol:='';
  for i:=1 to 7 do
    if Hate.Answer[i]=1 then HateCol:= HateCol + ' '+Color[i] endif
  enddo;
  ShowHate;
```

Limitations
BBinSet allows for 30 response categories. This can easily be changed by the user by

- adding or deleting response categories in BBinSet.Questio
- changing the dimension of BBinSet.Answer

PROCEDURE CompStr

Purpose
Match open answers with known strings with the possibility of wildcard and three alternative spellings.

Syntax

```
CompStr(t, t1, t2, t3,equal)
  t: string, input parameter (typically an open answer)
  t1, t2,  t3:  string, input parameters (e.g. product names or brand names)
  equal: integer, output parameter, which is
    1 if t matches t1, t2, or t3
    0 otherwise
```

t2 and t3 may be empty strings (no match)
t1, t2 and t3 may contain up to 5 wildcards, denoted by the underscore (_)
Comparison is carried out case-insentitive (no distinction is made between UPPER CASE and lower case)

Application
A typical application is the matching of an open answer of the respondent with a known product name or brand name, allowing for spelling errors by the respondent.

Example

```
CompStr(t,'Statistic_Ne_erland','Centraal_Bur_Statistiek','',match) yields
   t= Statistics Netherlands              match=1
   t= Centraal Bureau voor de Statistiek  match=1
   t= centraal bureau voor de statistiek  match=1
   t= CBS                                 match=0
   t= Statistiks Netherlands              match=0
```

BLOCK BGaborg

Purpose
BGaborg is a block that contains a standard procedure in marketing research to estimate demand curves of a brand or product without considering the prices of competing brands. This is known as the Gabor-Granger procedure.

Syntax

```
Gaborg(ProdName, BasePric, Increm, n)
   ProdName: string, name of the brand or product
   BasePric: real, the base price of the product (usually the price as marketed)
   Increm: real, steps by which the price is increased and decreased
   n: integer, number of increments and decrements (maximum = 5)
```

ProdName has to be a complete description of the product, e.g. “2.5 kilograms of Dreft Washing Powder”, as the respondent has to judge if the indicated price is a fair price.
The price of the product is set at different levels, and the respondent is asked if he would buy the product for that price. The prices are presented in random order.

The answers are stored in the array Gaborg.purch; purch[1] contains the response (yes or no) on the base price; purch[2] ... [purch[n+1] contain the responses to the decreased prices (BasePric-Increm,

..., BasePric-n*Increm); purch[n+2] ... purch[2n+1] contain the responses to the increased prices (BasePric+Increm, ..., BasePric+n*Increm)

Application
The Gabor Granger procedure is used to determine the price elasticity in a market where no direct reaction from competitors is to be expected. From the data can be estimated what percentage of the consumers buys a product at a given price.

Example

```
DATAMODEL ShowGaborg;
INCLUDE "Progamma_Library.inc";

LOCALS
  Product: string;
  MarketPr: real;
  inc: real;
  n_inc: integer;

FIELDS
  Gaborg: BGaborg;

RULES
  Product:= '2.5 kg of Dreft washing powder';
  MarketPr:= 6.0 ;
  inc:= 0.25; n_inc:=3;
  Gaborg(Product, MarketPr, inc, n_inc);
```

BLOCK BBPTO

Purpose
BBPTO is a block that contains a standard procedure in marketing research to estimate demand curves of a brand or product while considering the prices of competing brands. This is known as the Brand Price Trade Off procedure.

Syntax

```
BPTO(ProdArr, PricArr, Increm, n_prod, n_steps)
     ProdArr: TprodArr, array[1..12] of string[80] with the names of the products
     PricArr: TpricArr, array[1..12] of the base prices
              (usually the prices as marketed)
     Increm: real, the increment, steps by which the prices are increased
     n_prod: integer, number of products
              (maximum 12; it is recommended not to exceed 8)
     n_steps: integer, the number of steps (maximum 20)
```

ProdArr has to contain complete descriptions of the products, e.g. “2.5 kilograms of Dreft Washing Powder”, as the respondent has to judge if the indicated prices are fair prices.
The respondent has to indicate his product preference with prices at base level; subsequently, the price of the preferred product is increased by Increm, and the procedure is repeated n_steps times.

The results are written in the array BPTO.pref; pref[i] is the number of the preferred product in step i. It is recommended to present the products in random order; the most efficient way to do this is outside the block BBPTO, i.e. the array ProdArr already contains the products in random order, see Brandomize.

Application

The BPTO procedure is used to determine the price elasticity in a market where direct reactions from competitors are to be expected. From the data can be estimated onder what conditions what percentage of the consumers buys a product at a given price.

Example

```
DATAMODEL ShowBPTO;
INCLUDE "Progamma_Library.inc";

LOCALS
  inc: real;
  n_inc: integer;

AUXFIELDS
  Product: TProdArr;
  MarketPr: TPricArr;

FIELDS
  BPTO: BBPTO;

RULES
  Product[1]:= 'Blue Band'; Product[2]:='Becel'; Product[3]:='Bona';
  MarketPr[1]:= 1.0; MarketPr[2]:= 0.94; MarketPr[3]:= 1.12;
  inc:= 0.10; n_inc:=8;
  BPTO(Product, MarketPr, inc, 3, n_inc);
ENDMODEL.
```

4. Pitfalls for randomization routines

As noted in section 1, the basic structure of Blaise is not very 'randomization-friendly'. Therefore, the randomization routines should be used with care. The basic rule is that an instantiation of a block can be used only once. So for two tosses of a coin, two fields of type BToss have to be declared:

```
Toss1, Toss2: BToss;
Dime1, Dime2: TCoin;
```

Only then the rules

```
Toss1(Dime1);
Toss2(Dime2);
```

yields two independent throws of a coin, whereas

```
Toss1(Dime1);
Toss1(Dime2);
```

yields identical outcomes, i.e. Dime2=Dime1.

A second problem to keep in mind is that Blaise evaluates the randomization procedure the first time it is activated on the route and never changes the outcome afterward. This may create a problem for random permutations when the number of integers for permutation is not known beforehand. Both the blocks BRandAns and BrandSet are vulnerable to this problem. The solution to the problem is to make sure that the permutation is not on the route as long as the number of integers to be permuted is unknown. An example of how to solve this problem is shown in the file ShowNestSet.bla.

5. Available files

All routines are contained in the file

Progamma_Library.inc

Apart from the source, this file also contains more technical information about the routines than is given in this document. The routines are illustrated by a number of files with examples:

ShowBPTO
ShowGaborg
ShowIndInv
ShowNestSet
ShowRandAns
ShowSort
ShowStrComp

Understanding the Blaise Selective Checking Mechanism and the Appropriate Use of KEEP Statements

Pamela Ford, Statistics Canada

1. Background

In 1998 Statistics Canada adopted Blaise as the standard development tool for survey collection instruments.

As described by Mayda and Ford (2003), Computer Assisted Personal Interviewing (CAPI) applications at Statistics Canada consist of a combination of Blaise and Visual Basic components. Due to time constraints for the initial implementation of these applications, a pre-existing Visual Basic component was used for the collection of demographic and relationship information. The introduction of full Blaise applications for Computer Assisted Telephone Interviewing (CATI) operations in Regional call centres meant that demographic and relationship components would have to be developed in Blaise. Some of the initial prototypes that were developed and implemented had simpler requirements than the existing CAPI surveys wanted. The more complex prototypes suffered from noticeable performance issues.

With the move toward CAPI CATI Integration of the Labour Force Survey, it became clear that a more robust Blaise membership and demographic component that could be used by this and other surveys would have to be developed. This paper details the development process and the lessons learned.

2. Household Membership - Specification

In birth interviews where no previous information is available, membership data is collected in three rosters. In the first roster, interviewers enter names of those who *usually* live or stay in the dwelling. Next the interviewer is prompted to ask whether there is anybody temporarily staying in the dwelling and if so a second roster is used to collect the names of these *temporary* members. Finally the interviewer is prompted to ask for any persons who might not otherwise have been accounted for. Names of these *other* members are entered in a third roster. To minimize confusion for interviewers, all members listed are shown in all three rosters, giving the impression that only one roster is being manipulated.

In subsequent interviews, the first roster presented is pre-filled with names collected in *previous* collection cycles. A third question on the roster is used to indicate whether the members still reside in the dwelling. Once this data is confirmed and/or updated, the interviewer is prompted to ask for names of *other* members who have moved into the dwelling since the last collection cycle. Some surveys have specified that a special 'ghost' roster be used to display names of members who have lived in the household/dwelling at one time but were not present during the most recent collection cycle.

In both birth and subsequent interviews the member rosters are followed by yet another roster which is used to confirm, update, and/or collect demographic data.

The final step in this process is to determine each member's relationship to every other household member. This information is collected in a matrix format.

Additional complexities include:

- having to display all member names in any or all rosters
- being able to delete members entered erroneously
- being able to correct the spelling of previous members' names while ensuring that they aren't being inappropriately replaced completely,
- being able to copy the last name of a higher row (using the row #)
- being able to do both birth and subsequent interviews within the same application.

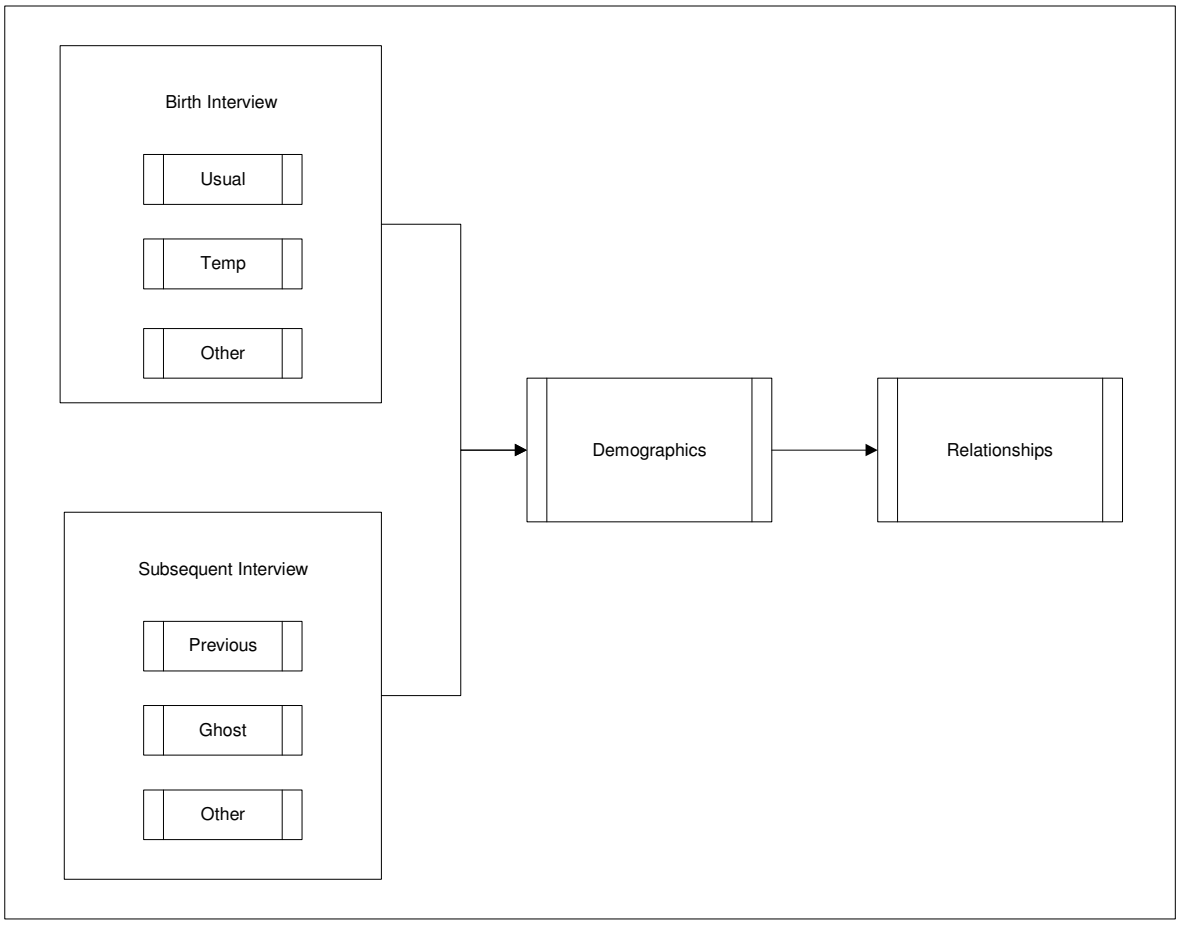


Figure 1 Household Membership Specification

3. Household Membership - Analysis and Development

3.1 Base Design

Having to display all members on every roster led to a number of key design principles.

First, there would have to be a one-to-one relationship across all rosters, whereby a person record would occupy the same line in any roster in which it is shown or asked.

Second, a *Master* roster was conceived to simplify the process of having to update all rosters. That is, data collected in each roster would be copied to the master, and the master would be the sole source for internal updates, including the updating of names in the demographic roster.

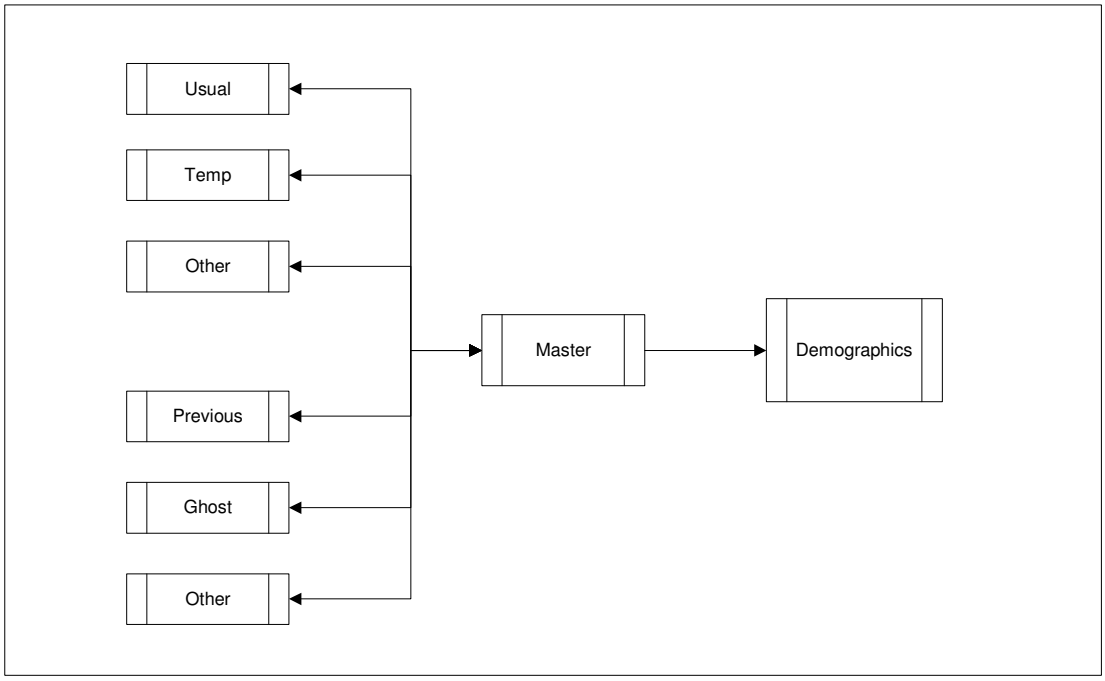


Figure 2 - Master Roster Model

Third, the existence of prompt questions before the Ghost, Temporary, and Other Rosters (Rosters are only on path if 'Yes' is answered to the prompt questions) meant that code had to be included to ensure consistency (If 'Yes' is answered to the prompt, then an edit would ensure that at least one member was enter in the associated roster. If 'No' is answered to the prompt, then any members entered in the associated roster would have to be deleted).

Finally, a 20 household member limit was set and it was necessary to ensure that no further prompts or rosters are presented in the stunningly rare instances where the limit is actually hit.

3.2 The Selective Checking Mechanism and the Placement of KEEP Statements

With the knowledge that Blaise always checks all appropriate rules in an instrument (Pierzchala and Razoux Schultz, 1996), it was anticipated that the continuous updating of the demographic roster could lead to performance degradation. This was managed by including logic which prevents the updating of the demographic roster and relationship matrix until the collection/confirmation of member names is complete.

The most challenging aspect of the development process proved to be the utilization and placement of KEEP statements. In early prototypes, by showing 'block checks' in the watch window, it was obvious that KEEP statements were either excessive or misplaced. Block-level KEEP statements were required to ensure that previous cycle data loaded ahead of time was retained, and to ensure that certain field values were known prior to their being used in rules logic, but how could these be implemented so as to minimize the checking of block rules?

To gain a better understanding of the KEEP functionality of Blaise, a simple datamodel was created and various arrangements of block level KEEP statements were explored. In the course of this research, an anomaly with the use of the <End Key> was uncovered which added further complication until Blaise 4.6 became available.

```
RULES
  {
   Model 1:  Base case, No KEEP Statements
    - Since there are no KEEP statements we risk losing data if the case is re-entered and the
      answer of BlockA.A4.SB1_F1 changed so that blocks B,C, and D are no longer on the path.
      This flow is comparable to the situation where an interviewer re-attempts a case to complete
      an interview and answers 'No' to 'Have you made contact?' in the event of a no-answer or
      busy etc. dial result  We must still KEEP the subject matter component in this situation.
  }

  BlockA
  IF BlockA.A4.SB1_F1 = Yes THEN
     BlockB
     BlockC
     IF BlockC.C4.SB1_F1 = Yes THEN
        BlockD
     ENDIF
  ENDIF
  BlockE
```

Figure 3 No KEEP Statements

```
RULES
  {
   Model 2:  KEEP statements at the beginning
    - The result of the KEEP statements at the beginning is that at times, block rules are
      unnecessarily checked twice for some blocks.
    - N.B. In V453B650, using the endkey can result in blocks/fields being skipped.
  }
  BlockB.KEEP
  BlockC.KEEP
  BlockD.KEEP
  BlockA
  IF BlockA.A4.SB1_F1 = Yes THEN
     BlockB
     BlockC
     IF BlockC.C4.SB1_F1 = Yes THEN
        BlockD
     ENDIF
  ENDIF
  BlockE
```

Figure 4 KEEP Statements at Beginning

```
RULES
   {
    Model 3:  KEEP statements in ELSE sections
     - Here the blocks are only checked once as appropriate.
     - N.B. In V453B650, using the endkey can result in fields being skipped.
   }
   BlockA
   IF BlockA.A4.SB1_F1 = Yes THEN
      BlockB
      BlockC
      IF BlockC.C4.SB1_F1 = Yes THEN
         BlockD
      ELSE
         BlockD.KEEP
      ENDIF
   ELSE
      BlockB.KEEP
      BlockC.KEEP
      BlockD.KEEP
   ENDIF
   BlockE
```

Figure 5 If ASK Else KEEP

```
RULES
   {
    Model 4:  KEEP statements on path only if last question is answered.
     - Here the blocks are only checked once as appropriate.  (N.B. If BlockE.Bye were not
       an auxfield, it would have to be initialized to EMPTY the first time through the rules.)
     - This is a viable work-around for the End-Key issue in V453B650.
   }
   BlockA
   IF BlockA.A4.SB1_F1 = Yes THEN
      BlockB
      BlockC
      IF BlockC.C4.SB1_F1 = Yes THEN
         BlockD
      ENDIF
   ENDIF
   BlockE
   IF BlockE.Bye = Ciao THEN
      BlockB.KEEP
      BlockC.KEEP
      BlockD.KEEP
   ENDIF
```

Figure 6 KEEP if last question answered

From this research, it was obvious that the If-ASK-Else-KEEP approach should be used wherever possible. Though there are several instances where this method was not feasible, the checking of rules within the roster component has been optimized as follows:

- At the highest level, the Roster block is kept only if the 'ASK' is off path.
- A KEEP statement for each of the Usual, Temp, Other, Previous, and Ghost tables is included at the beginning of the Roster rules as much of the block logic is dependant on their contents being known.
- A KEEP statement is applied for the Demographic block, only if the ASK is off path.
- No KEEP statement was required at all for the Relationship block.

4. Lessons Learned

The development of this complex component not only provided further insight into the usage of KEEP statements, but also reinforced existing knowledge of how block parameters should be implemented. Furthermore, it demonstrated how crucial a step prototyping can be in the development process.

It is critical that developers understand the Blaise selective checking mechanism and the impact of both KEEP statements and parameters so that the performance of their instruments will be optimized. The 'Show Block Checks' option in the watch window proved to be an invaluable tool and its use by developers will be encouraged throughout Blaise development areas.

5. Conclusion and Future Direction

A strategic streamlining initiative at Statistics Canada has resulted in the specification and development of standard front and backend templates to be used with all social survey applications. Although the method of multiple rosters was proven to be satisfactory in terms of performance and addressing client needs, it was recommended that the household membership component of the standard templates consist of a single roster for entering member names. This is considered to be a better approach as it will allow for simpler code that can be maintained by a wider range of developers.

6. References

Mayda, J. and Ford, P. (2003). Integration of CAPI and CATI Infrastructures at Statistics Canada. Proceedings of the Eighth International Blaise Users Conference, Denmark

Pierzchala, M. and Razoux Schultz, G. (1996). Optimal Instrument Performance in Blaise III. International BLAISE User Group Newsletter, no 8, pp. 11-18, ONS, London publisher.

Instrument assembly, documentation and release

Fred Wensing, Australian Bureau of Statistics

1. Introduction

Survey instruments usually consist of a package of compiled files that need to be kept together so that the systems can make use of them. While the collection of files that are needed to form this package is not unduly complex, difficulties can arise if one or other elements of such a package are missing or has been incorrectly prepared. For this reason a system has been developed to manage the assembly, packaging and release of population survey instruments at the Australian Bureau of Statistics (ABS).

2. A standard environment

Successful management of instrument preparation is firstly dependent on a standard environment where staff can develop and maintain their programs. The standard environment for instrument preparation at the ABS consists of the following:

- a single network location for all instrument development;
- an agreed folder structure to store all source programs;
- a protected location for all shared source code elements;
- a set of standards and guidelines for instrument design.

2.1 Single network location

A single network location means that instrument code can readily be located by anyone who needs to have access. A single location also means that it is only necessary to have one master copy of shared elements. A single location also means that support and assistance for instrument developers can be more readily provided.

Access to the single network location at the ABS is granted as a part of the process associated with providing access to the Blaise software itself. Therefore, any staff who receives access to the developer installation of Blaise will also have read/write access to the development location.

2.2 Agreed folder structure

An agreed folder structure in the development environment is essential for proper management of source code. The folder structure enables people to readily identify their survey and keep their source code separate from others. Access to survey folders can be restricted to specific staff if required.

At the ABS, a simple structure has been developed whereby each instrument is developed within a folder which exists at the root of the shared network drive. The following simple naming convention is employed to assist with proper identification of instrument folders:

<nickname><reftime><cycle>

where:

<nickname> = a small number of letters (usually 3) for the survey (eg AHS)

<reftime> = two (or more) numbers representing the year or year and month that the survey is to take place (eg 98, 99, 0104)

<cycle> = two or three letters corresponding to the cycle (eg PT1, DR, FIN)

Generally there are only two subfolders below each survey folder, one for local survey-specific modules to be included in the instrument (called INCLUDE) and one for module testing (called MINIMOD).

2.3 Protected location for all shared files and shared source code

In addition to the survey folders there are a number of folders which contain source code that can be included or referenced by all instruments. These folders are write-protected to avoid inadvertent update but are otherwise accessible to all developers.

These shared folders are:

COMMON - a folder containing further sub-folders of common source code. The sub-folders are all identified by a nickname and a version number which relates to the particular component and version release of that component.

CONFIG - a folder containing shared mode libraries, menu and configuration files

EXTERNAL - a folder to hold compiled and loaded Blaise data files which contain look-up lists that have been developed for sharing across surveys.

SYSFILES - a folder of system utility programs, including those used to assemble and prepare instruments

TEMPLATES - a folder to hold templates of programs that can be used by the instrument assembly system to create survey-specific programs (see section 4).

2.4 Illustration of folder structure

An illustration of folder structure implementation at the ABS is given in Figure 1.

```
L:\CONFIG              - for system configuration files
L:\EXTERNAL            - for common look-up data files (code lists etc)
L:\COMMON\HFv3.08      - common modules from v3.08 of the Household form
L:\COMMON\HFv3.09      - common modules from v3.09 of the Household form
L:\COMMON\SYS1.10      - common modules from v1.10 of the system
L:\EXTERNAL            - all coder files named by version
L:\ISS02PT\            - main ISS pilot test project file, instrument and Manipula
L:\ISS02PT\INCLUDE     - ISS pilot test INCLUDE files
L:\ISS02PT\MINIMOD     - ISS pilot test module testing files
L:\ISS02DR\            - main ISS dress rehearsal project file, instrument and Manipula
L:\ISS02DR\INCLUDE     - ISS dress rehearsal INCLUDE files
L:\SYSFILES            - programs used for management of the source code
L:\TEMPLATES           - template or shell programs used to generate standard modules
```

Figure 1. Examples of folders found on the ABS shared network drive

2.5 Standards and guidelines for instrument design

A set of standards and guidelines has been developed at the ABS to assist people with preparation of instruments. These cover various elements such as naming conventions and good practice in Blaise programming, too detailed to be included here. However, the following aspects of ABS standards and guidelines are relevant to instrument assembly:

- all instruments will be prepared in the shared environment;
- all instruments will be prepared using a Blaise Project Definition file which includes the use of version numbering (see section 3);
- all instruments will make use of the standard system modules which include standard type definitions and a common (standard) mode library;
- all instruments will make use of the common (standard) modules for the collection of household composition and demographics (if relevant);
- all modules will include a history section at the top which can be used to track any changes that are made (see section 3);
- all INCLUDE statements will use the relative reference format (rather than the explicit reference format).

2.6 Format for the use of INCLUDE statements

The use of a single network location for instrument source code makes it simple to include instrument modules from anywhere on the network drive. This is achieved through the Blaise INCLUDE statement and a relative reference format.

To include modules of code which are local to the survey, the include statement takes the form:

```
INCLUDE "INCLUDE\<modname>"
```

Where:

<modname> = the name of the module of code to be included

To include modules of code which come from the common folders, the include statement takes the form:

```
INCLUDE "..\COMMON\<release>\<modname>"
```

Where:

<release> = the release version of the common module(s)
<modname> = the name of the module of code to be included

The main reason for using a relative reference format is to avoid the problem which might arise if the drive letter is changed. The relative reference format is also used in preference to the Source Search Path Option in the Project Definition file because the same named module may appear under various release folders.

An example of some include statements found in a typical ABS instrument may be seen in Figure 2.

```
INCLUDE "..\COMMON\PROCS\REPLACEDQ.BLA"
INCLUDE "..\COMMON\SYSv5.04\STDTYPES.BLA"
INCLUDE "..\COMMON\SYSv5.04\SPECS.BLA"
INCLUDE "..\COMMON\HFv6.04\CAIDEMOG.BLA"
INCLUDE "..\COMMON\LFv2.11\INTERVIEWSTATUS.BLA"
INCLUDE "..\COMMON\LFv2.11\STATUSFLDS.BLA"
INCLUDE "INCLUDE\PersonQre.bla"
```

Figure 2. Typical include statements found in ABS instruments

3. Version control at both module and instrument levels

In order to manage the succession of changes that can happen to an instrument, a system of version control has been implemented at the ABS.

Not only are instruments at the ABS given a version number but each module of source code is required to have provision for the recording of version history which is to be updated whenever that module changes.

3.1 Instrument version numbering through the Blaise Project Definition file

The Blaise Project Definition file enables a composite version number to be added to an instrument through the Project Options menu/Version Info tab. This composite version number is a combination of Major and Minor version plus Release and Build number (see figure 3).

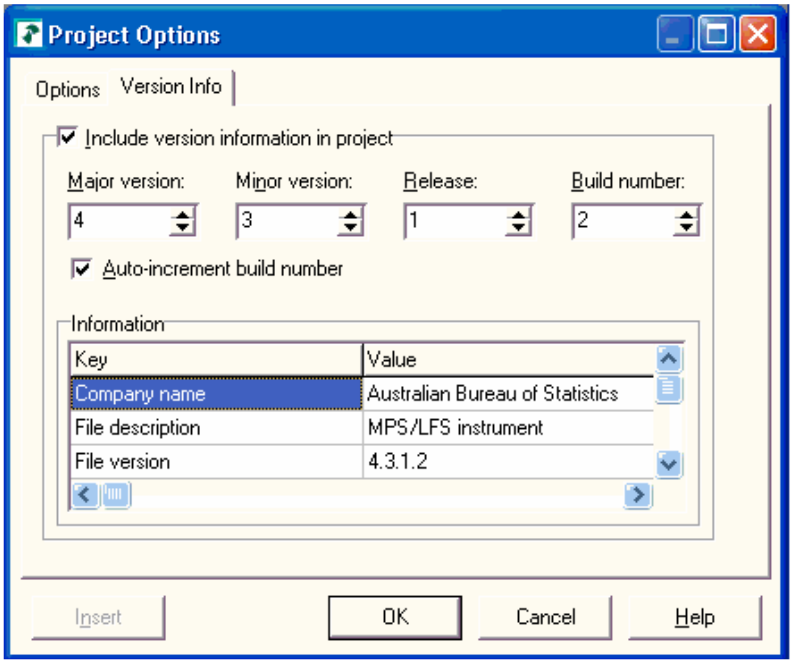


Figure 3. Version information found under Project Options

Local rules are devised for each survey to enable people to decide which number to change. It is also good practice to ensure that the Auto-increment build number option is turned on so that each compilation of an instrument (for whatever reason) will have a new build number and therefore have a unique version number.

The main version rule, which is followed for all surveys, is that a new release must always receive a new number. In most cases the new number will involve a change to the Major, Minor or Release number, unless it is definitely known that the new compilation will produce an instrument which is

fully compatible with the previous release. In such cases, a change will occur in the build number only.

3.2 Accessing the version number from the compiled instrument or data file

One advantage of incorporating a version number into an instrument, is that the version number of both an instrument and the collected data can be subsequently identified. This is done through the Help/Info menu which then brings up the Dialog shown in Figure 4.

Figure 4. Help info dialog

To access the version information for either the Metadata or the Data press the relevant button of the info dialog. An information screen such as that shown in Figure 5 will then be displayed.

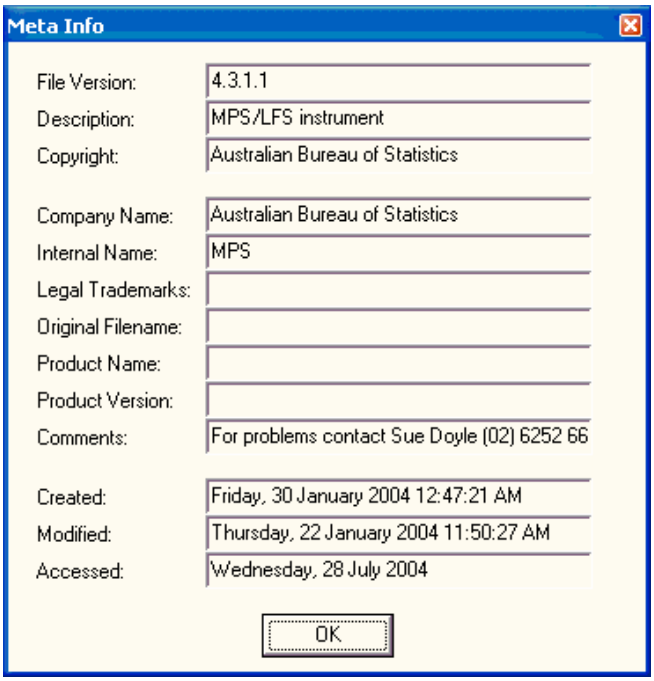


Figure 5. Version information for the metadata

3.3 Maintaining a history of module changes

Because instruments are usually composed of modules of source code, it is important that a history of the changes to those modules is maintained.

For ABS instruments a simple system of history recording has been implemented which enables all changes to be identified and dated. The system involves the use of a comment placed at the top of each unique module of code. The format of that history is such that information from it can be extracted by the instrument preparation utility.

The general format of the history is that of a Blaise comment in which there are some lines of free-form text to identify the module, its main author and its main purpose. Following that information, provision is made for update history in the format:

<Version> <dd/mm/yyyy> <Person> <Change>

where:

<Version> = a decimal number that identifies a major and minor version number for any change
<dd/mm/yyyy> = a properly formatted date
<Person> = the unique user name of the person making the change
<Change> = a brief description of the change

An example of a properly formatted history is shown in Figure 6.

```
{************************************************************
 Module Name            :  MPHS04FIN.BLA
 Author                 :  Fred Wensing
 Update History:
 Vsn  dd/mm/yyyy Person Change
 ---- ---------- ------ ------
 2.05 26/05/2004 Halls  - Add derivation for HoursMainJob
                        - Deleted edits WTW1, WNM1 and WNM3
 2.13 21/06/2004 doylsu - changed 'includes' to point to COMMON
 2.14 23/06/2004 doylsu - released with v1.063 using August MPS
************************************************************}
```

Figure 6. Example of formatted history

4. Templates used for standard programs

Survey systems, which are required to support a variety of surveys, ultimately end up using standardised instruments, programs and practices. Despite the fact that standardised programs are used, however, those programs often need to be modified (eg. to point to new survey instruments) and recompiled before use. While the nature of the day-to-day changes to the standard programs is often small (eg: a changed instrument name; a changed label) there is a risk that facilities will not work propery, or fail altogether, if the changes are not made correctly.

For this reason, a system of template programs has been developed at the ABS in which certain sections of those programs are updated by the system before use. The template programs are, for all intents and purposes, functional programs in which particular sections of the source code have been tagged for modification or replacement by the assembly system.

The general method of tagging is to insert some identifiable and formatted text within a comment in the source code. This text can be located by the assembly system which scans the source program(s) for the tags and takes appropriate action.

Two forms of tagging have been devised:

- tagging of a line to be replaced;
- tagging of a segment of code to be included, removed or replaced.

To complete the assembly system, details of tag replacement information are stored in a tag definition dataset. Reference is made to this dataset during the assembly process.

4.1 Tagging of a line to be replaced

The general format of a tag for a line to be replaced is:

```
{>>> <tagname>}
```

Where:

{}	= the standard delimiters for a comment in Blaise or Manipula
<tagname>	= a text string of 5 characters which identifies the tag by name
>>>	= an identifying characteristic.

The above tag is placed at the start of any line which is to be replaced. When the assembly system finds such a line it rewrites the remainder of the line (after the tag) using information found in the tag definition dataset.

All comment lines which contain the >>> characteristic are assumed by the assembly system to be a tag of some kind. This set of characters was selected because it can never appear as part of a Blaise or Manipula program. While this set of characters could appear within a comment, for other reasons, the assembly system also needs to recognise the tag name before taking any action to modify the line of source code. The set of characters is also readily recognised by persons looking at the program source code.

Note that the same tag name can be used multiple times within an instrument if more than one line is to be changed in the same way.

Figure 7A shows a section of Manipula program which contains a line that has a tag named META1 to identify a metadata reference (the template currently has **'LFS211'** on that line).

```
...
USES
   SurveyMeta
{>>> META1} 'LFS211'
   DATAMODEL mList
      Fields
         xPSU      : STRING[5]
...
```

Figure 7A. Manipula program showing a line tag before update

Figure 7B shows the same section of Manipula program, containing a new metadata reference for META1 (now **'LFS403'**) after it has been processed by the assembly system.

```
...
USES
   SurveyMeta
{>>> META1} 'LFS403'
   DATAMODEL mList
      Fields
         xPSU      : STRING[5]
...
```

Figure 7B. Manipula program showing a line tag after update

4.2 Tagging of a segment of code to be included, removed or replaced

When a segment of code is to be included in a program then the tag is written in the form:

```
{>>> INCLUDE <inclname>}
```

Where:

{}	= the standard delimiters for a comment in Blaise or Manipula
INCLUDE	= a keyword used by the assembly system
<inclname>	= a text string to identify the segment of code to be included
>>>	= an identifying characteristic

When the assembly system encounters such a tag it will remove the tagged line and insert the lines of text (usually progam text) from a file which has the name **<inclname>.INC**. The include file is expected to be in the same folder as the program being updated.

When a segment of code is 'included' in this way the system adds two new tags to the program so that the segment of code can be removed by the assembly system at a later date. The two new tags are:

```
{>>> START <inclname>}
```

and

```
{>>> ENDOF <inclname>}
```

Where:

{}	= the standard delimiters for a comment in Blaise or Manipula
START	= a keyword used by the assembly system
ENDOF	= a keyword used by the assembly system
<inclname>	= a text string to identify the segment of code which has been included
>>>	= an identifying characteristic.

These tags are added to the start and end of any segment of code which has been included.

Figure 8A shows a section of a Blaise instrument which contains a tag to include a segment of code named **Suppfields.INC**. Note that the use of <<< at the end of the tag definition serves no functional purpose but is done for appearances.

```
    xNum2 : TNum99
    xChar1 : TString1
    xChar2 : TString2
{>>> INCLUDE SuppFields <<<}
    AUXFIELDS
        aName, aNamePoss, aIsAreLC, aIsAreUC, aHasHaveLC,
        aHasHaveUC, aWasWereLC, aWasWereUC,
        aDoesDoLC, aDoesDoUC, aVerbEnd
            : STRING
```

Figure 8A. Code segment showing the INCLUDE tag

Figure 8B shows the same section of a Blaise instrument with the included **Suppfields.INC** code and the additional tags which mark the start and end of the included code.

```
    xNum2 : TNum99
    xChar1 : TString1
    xChar2 : TString2
{>>> START SuppFields <<<} {v3.14 04/04/2004}
    INCLUDE "INCLUDE\Introduction.bla"
    FIELDS
        Introduction: BIntroduction
    INCLUDE "INCLUDE\Usualwork.bla"
    FIELDS
        Usualwork : BUsualwork
    INCLUDE "INCLUDE\Absences.bla"
    FIELDS
        Absences : BAbsences
    INCLUDE "INCLUDE\Childcare.bla"
    FIELDS
        Childcare : BChildcare
{>>> ENDOF SuppFields <<<}
    AUXFIELDS
        aName, aNamePoss, aIsAreLC, aIsAreUC, aHasHaveLC,
        aHasHaveUC, aWasWereLC, aWasWereUC,
        aDoesDoLC, aDoesDoUC, aVerbEnd
            : STRING
```

Figure 8B. Code segment showing the text included and the tags for START and ENDOF

Note that the assembly system has added version information, transferred from the history header of the main program, as a comment to the included code. This is so that staff looking at the code subsequently are made aware of when the code was added. Once again, the use of <<< at the end of the tag definition has been done for appearances only.

4.3 Further discussion of the tag approach

The tag approach was devised in answer to a need to be able to include or replace small or large segments of code in a program. Blaise provides an INCLUDE statement for use within instrument definitions but it is limited to complete sets of fields and/or rules only. In addition there is no provision for an INCLUDE statement in the Manipula language.

The advent of prepare directives (from version 4.6 of Blaise onwards) may provide an alternative to the tag approach described in this paper. However, at the time that this system was developed, prepare directives were not available in Blaise.

5. Assembly utility

With the existence of template programs, that are suitably tagged, it is possible to generate usable copies of the templates in which the tagged elements have been replaced or updated accordingly. To do this an assembly system has been built, in Maniplus and Manipula, containing the following elements:

- a register of templates
- a register containing tag set definitions
- an interface to enable the registration of templates and updating of tag set definitions
- a set of procedures associated with the generation of programs from templates.

5.1 Interface to the Assembly system

A simple Maniplus interface (see Figure 9) is used to display a list of tag set definitions for the assembly system. The interface also contains a series of buttons to activate the various procedures associated with the assembly system.

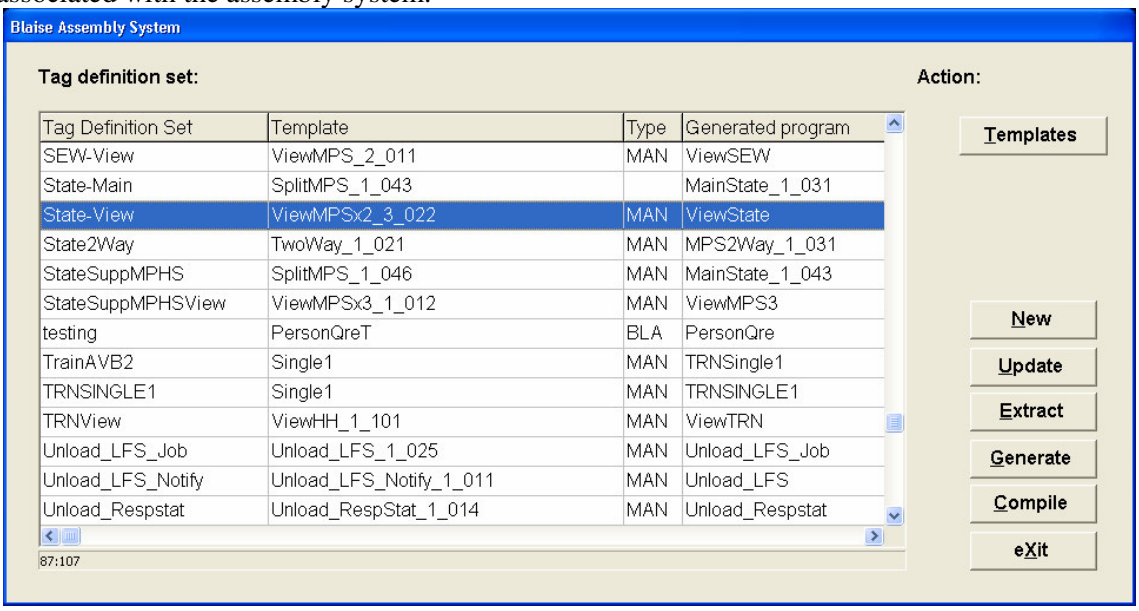


Figure 9. Assembly system interface

5.2 Register of templates

Template programs are assigned a unique identity consisting of a name and version number (Major, Minor and Release number combined), and registered in a simple Blaise database. Other information included in the register is the type of template (Blaise or Manipula) and a brief description.

Once they are registered in that database they can be linked to any tag set definition.

5.3 Register of tag set definitions

The register of tag set definitions contains the information required by the assembly system to modify one or more of the tags it finds within a given template.

Figure 10 shows an example of a tag set definition in which the values are given for the various tags which are expected within the template.

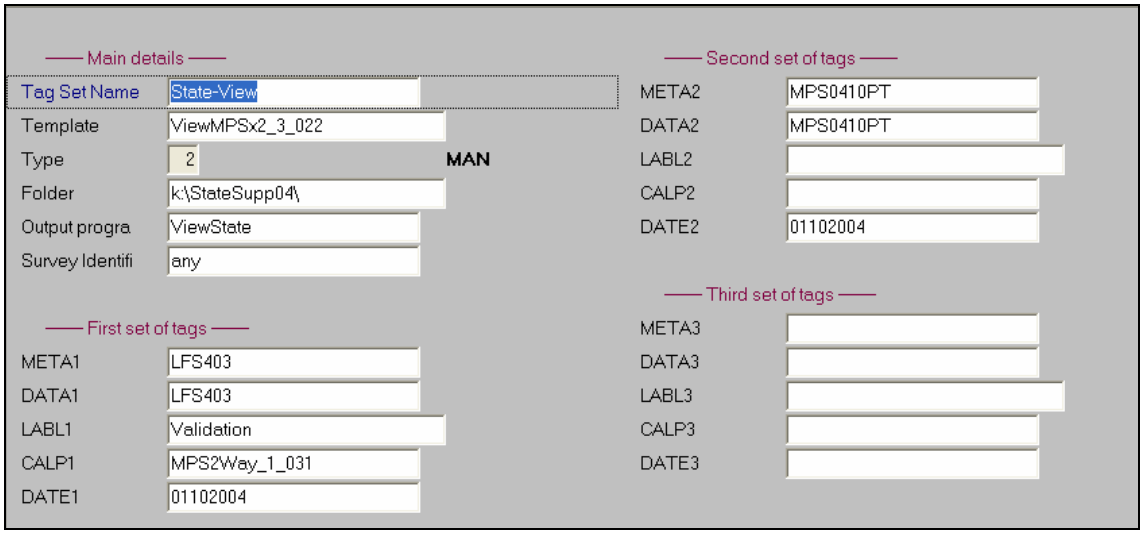


Figure 10. Example of a tag set definition

In this example, the **State-View** tag set definition contains the tag values to be used with the template called **ViewMPSx2_3_022** to produce an output program called **ViewState.MAN**. You can see that the tag called META1 will be set to the value **`'LFS403'`** when processed by the Assembly system. This is the origin of the tag value used in Figure 7B.

Note that tags which are not used in the corresponding template are left blank in the tag set definition.

5.4 Set of procedures associated with the generation of programs

There are two main procedures which have been developed to support the assembly system:

- procedure to generate an operational program from a Template (using the selected tag set definition)
- procedure to extract the included segments of code from an existing operational program.

These procedures are maintained as Manipula programs that are called by pressing the appropriate button in the interface.

The other procedures in the assembly system enable the tag sets and the template definitions to be updated.

5.5 Operation of the Assembly system

The usual mode of operation of this system is for staff to activate the interface and generate program(s) as and when required.

Special provision has been made, however, for the generation process also to be run by the survey system itself, via a scripted call to the generation Manipula program. This is achieved by setting up the tag set definition in a special way that enables the values of tags to be passed to the generation program through parameters.

The advantage of this assembly system is that tailored programs to support the operations of ABS surveys can be quickly and reliably produced.

6. Instrument package utility

Once all the instruments and associated programs (some generated) exist for a survey, it is necessary to collate the compiled files into a package for installation into the survey systems. At the same time, it is important to collate the contributing source files into another package for archival purposes and to assist with any problems that may arise.

An instrument preparation utility (called PrepareZips), built mainly in Maniplus and Manipula, has been produced at the ABS to take care of the packaging process. The main features of the utility are:

- a register of instrument release details;
- an interface to manage the steps involved;
- enforcement of version control;
- preparation of zip files containing various parts of the instrument;
- a list of contributing source code drawn from the common (shared) store;
- a report of the version history of all local contributing modules of source code;
- a listing of the edits found in the source code.

6.1 A register of instrument releases

Each release is recorded in a register of instruments. The registry entry records the type of release (instrument, code list, template etc.), the version number for the release and identifies the files (Blaise datamodels, Manipula and text files) to be included.

Details for instrument : MPSLM03_3_02 Release 2
(To locate the page containing the version number press the Home key)

Instrument type

- (•) 1. CAPI instrument
- ( ) 2. Data Entry instrument
- ( ) 3. Code list
- ( ) 4. Set up file
- ( ) 5. System file(s)
- ( ) 6. Templates

General specs			Datamodels	
Type	1	CAPISurvey	Model 1	MPSLM03
Survey	Labour Mobility		Model 2	
Source folder	c:\data\blaise\mpslm03demo\		Model 3	

Includes			Manipulas	
Include	1	Yes	Manipula 1	MPSLM_4_011
Extra folder			Manipula 2	
Modelib	2	No	Manipula 3	ViewLM
Menu file	2	No		

——> More on next page

Figure 11. First page of a typical entry in the instrument register

6.2 An interface for the preparation utility

The interface for the instrument preparation utility (see Figure 12) shows a list of the current released instruments and a series of buttons which activate the various steps.

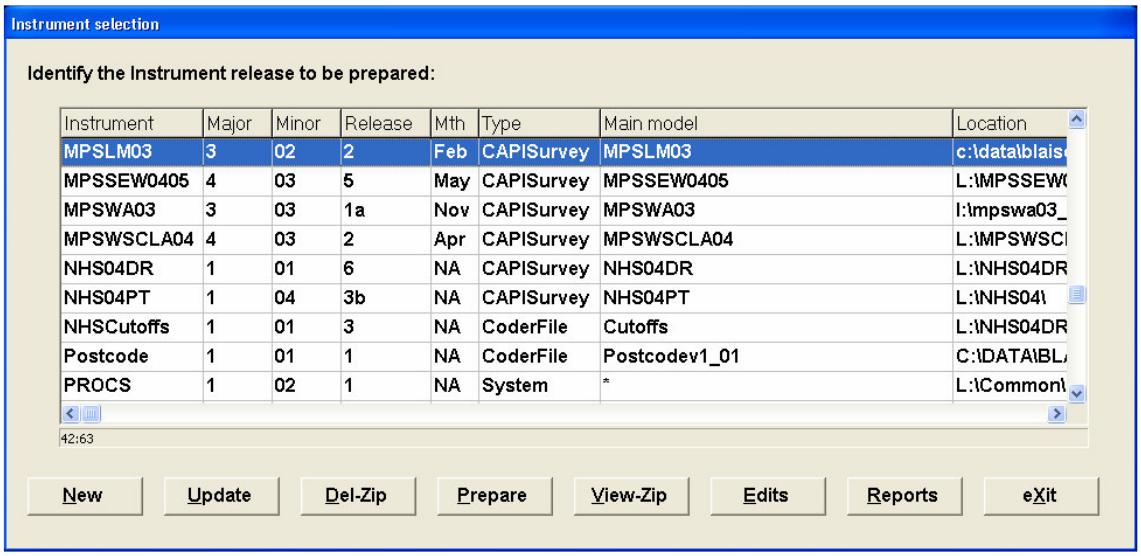


Figure 12. Interface for the preparation utility

6.3 Enforcement of version control

When an instrument is registered for preparation the utility requires details of the major, minor and release version numbers. Without these details, a release cannot be prepared.

Once supplied, the release number is checked against the release number found in the Blaise Project Definition file. If the version number is not the same then an error message is produced (see figure 13) and the preparation stops.

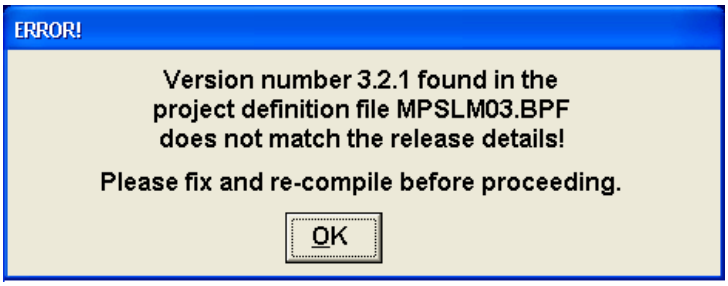


Figure 13. Error message for mismatching version number

6.4 Preparation of the Zip files

The preparation utility produces a list of the full names of all the required files and then activates Winzip to load two zip files:

- one containing the compiled programs along with any data files or text files that are needed;
- a second, containing the local (survey specific) contributing modules of source code for archival purposes.

The utility is programmed to take note of the instrument type (instrument, code list, template etc.) and adjust the lists of files accordingly.

The Zip files produced are given a name which matches the instrument name and full version number. This makes it very easy to align the Zip files with their corresponding entry in the instrument register.

Note that only the local contributing modules of source code are loaded into the source code Zip file because the common (shared) modules would have been released in their own right and would already have been loaded into their own source code Zip file.

6.5 Report on the contributing source code

The instrument preparation utility produces a report (see Figure 14) of the contributing modules of common source code. This list can be used for checking purposes and can assist any person who wishes to re-create the instrument from the source code.

```
ZIP file : MPSLM03_3_022_Feb_Source
-----------------------------------
List of shared components used

TYPE=Include
------------
..\COMMON\HFv7.03\
        Caidemog.BLA
        Caihhold.BLA
..\COMMON\LFv2.14\
        Bhourswork.BLA
        Blastjob.BLA
        Blastjobft.BLA
        Bunempnilf.BLA
..\COMMON\PROCS\
        Firstletteruc.BLA
        Getdatefills.BLA
..\COMMON\SYSv5.10\
        Householdid.BLA
        Iwmsflds.BLA
        Officefields.BLA
        Specs.BLA
        Stdtypes.BLA

TYPE=Modelib
------------
..\Config\
        Mode2_f14_g9x2_desc_tdesc.BML
```

Figure 14. Report of contributing modules

6.6 Report of the version history of contributing modules

The instrument preparation utility also scans all the local contributing modules of source code for the version history and produces a report of the latest changes (see Figure 15). This report, which shows the version number and persons who have made the latest changes, provides a simple way of identifying the instrument changes over time.

This report is only possible if the history entries have been formatted in accordance with the description given in Section 3.3.

```
ZIP file : MPSLM03_3_022_Feb_Source
-----------------------------------
List of modules and history

NAME                          LINES  VERSION     DATE     AUTHOR
==========================  ======  =======  ==========  ======

TYPE=BLA
--------
Education                       168  3.02     04-02-2003  mcnall
Initial_sequence                 83  3.02     07-04-2003  mcnall
Introduction                     84  3.02     04-02-2003  mcnall
Job_mobility_other              117  3.02     07-04-2003  mcnall
Mpslm03                        1502  2.13     01-08-2003  Doylsu
Occ_prev_job                    125  3.03     03-04-2003  MCNALL
Personqre                       950  3.12     06-08-2003  doylsu
Previous_job_status             170  3.02     29-01-2002  mcnall
Specsrules                       48  4.13     06-08-2003  Doylsu
                              ------
                               8900

TYPE=MAN
--------
Viewlm                          609  1.14     21-02-2003  Wensfr
                              ------
                                609
```

Figure 15. Report of instrument version history

6.7 Listing of the edits found in the source code

The instrument preparation utility can also produce a list of the edit statements in the local contributing modules of source code (see Figure 16).

This listing is produced using a SAS program (developed in 1996) which scans the source code for the Blaise keywords SIGNAL and CHECK and then lists the statements which follow. For this aspect of the utility to operate successfully it is important that these keywords accompany each edit.

```
MPSLM03_3_022_Feb_Source                            July 31, 2004

List of Blaise programs containing SIGNAL or CHECK
in C:\DATA\
                                                     Line
NAME                          SIGNAL    CHECK    count
Education.bla                    0         0       168
MPSLM03.bla                      0        20       283
...
SpecsRules.bla                   0         0        48
                              ======    =====    =====
                                 0       161      8900
*******************************************************
MPSLM03_3_022_Feb_Source
List of edits found
=======================================================
Module : MPSLM03.bla
---> Line 627
     CHECK
     Hhld.Qns.Scope[LCount].xPersTypePD <> Visitor
     "Visitors are not included in this survey - select UR or Remove"
---> Line 942
     CHECK
     Hhld.Qns.D3[LCount].Age <= 24
     "Boarding school pupils must be aged 24 years or less"
=======================================================
Module : PersonQre.bla
---> Line 556
     CHECK
     pYearArrived <= YEAR(xIntDate)
     "Cannot have arrived in Australia after this year"
...
```

Figure 16. Example of a list of edits found in an instrument

7. Conclusion

The systems described in this paper have been developed in order to ensure that instrument packages cans be readily and reliably produced. When an instrument package is produced, the local source code is checked for version and history information. The local source code for an instrument package is also collated into a separate Zip file for storage purposes. Summary documentation is then produced to identify the contents of the instrument and the current version information.

The success of these facilities is dependent on good practice in the management and sharing of code, a simple system for recording update history and the implementation of version control. Added to this is a special method of identifying or tagging lines of code which can then be updated by the system if required.

Using Arabic in Blaise

Shani Abel / Shifra Har, Central Bureau of Statistics, Israel

1. Introduction

Surveys conducted by Israel's Central Bureau of Statistics must accommodate a multilingual population. The main language of our surveys is Hebrew, however, Arabic is Israel's second official language and it was therefore crucial that we find a solution for displaying Arabic, particularly once we began using CAPI. With both a large Arabic-speaking population and a large Russian-speaking population a solution was sought for incorporating both languages into the surveys. In the past, this need was answered by providing paper-based questionnaires when necessary. Thus, while most interviews were conducted in Hebrew by CATI, at times surveyors were sent to a respondent's home with a questionnaire in either Arabic, Russian or English.

The problem of displaying all three languages in a single survey was complicated by the diverse nature of these languages. Both Hebrew and Arabic are written RTL, while Russian is written LTR and each language requires a different number of characters for its alphabet, with Arabic requiring at least hundreds of characters, if not thousands.

The problem of Russian was solved by employing an interpreter that converts Cyrillic characters to encoded text understood by Blaise, which then displays Russian in Bulgarian Courier font. We have been using Russian in our CAPI surveys since 2002.

Arabic meanwhile was presented in Hebrew transcription while we continued to work on a solution for presenting Arabic fonts in the DEP. The breakthrough occurred when we realized that we could combine characteristics unique to the Hebrew-operating system (such as RTL functionality) together with characteristics unique to Blaise, to achieve a complete solution. We are pleased to report that our efforts have been successful and as of 2004 our surveys will include all three languages in the original.

In this paper, we will outline some of the problems we encountered with Arabic fonts and we will present both our solution for displaying Arabic in the Blaise DEP and its mode of deployment.

2. Problems with Arabic Fonts

Several factors contributed to the complication of displaying Arabic in the DEP.

2.1 The Arabic Alphabet

Arabic script consists of 14 basic shapes, representing 28 consonants. These are distinguished by the use of diacritics: symbols above or below the letters that signify the correct letter and the vowels.
Here are some of the main differences between Arabic and Latin characters:

1. The characters are laid out RTL, with the exception of numbers, which are laid out LTR. This change in the writing direction is usually described by the expression bidirectionality of Arabic text. Bidirectional text also emerges when Latin and Arabic text is mixed.
2. Letters change shape depending on context. Each letter has *up to* four forms: the initial form, where it is the first letter in a word, the final form, where it is the last letter in a word, the

isolated form, and the medial form. This means that as a word is typed, the shape of previously-entered letters changes as well as a new letter being added.

3. Arabic includes diacritics. These are marks placed above or below letters which typically represent vowel sounds or other modifiers.
4. Arabic includes ligatures. This is the process whereby two consecutive letters printed are replaced by a single new character.

Figure 1 – The Arabic Alphabet

خ Xaa'	ح H`aa'	ج Jiim	ث Thlaa'	ت Taa'	ب Baa'	أ 'Alif
ص Saad	ش Shiin	س Siin	ز Zaay	ر Raa'	ذ Thaal	د Daal
ق Qaaf	ف Faa'	غ Ghayn	ع 'Ayn	ظ Th:aa'	ط Taa'	ض Daad
ي Yaa'	و Waaw	ه Haa'	ن Nuun	م Miim	ل Laam	ك Kaaf

Light Grey - Letters which cannot be joined on the left
Dark Grey - Letters which dramatically change shape according to their position in the word

Figure 2 – examples of four forms of characters ('ayn', 'baa', 'qaaf' and 'haa', respectively), according to their position in the word:

Initial	Isolated	Medial	Final
عرب	ع	يعرب	مع
برد	ب	يبني	حلب
قلم	ق	يقيم	دمشق
هالة	ه	يهيم	نبيه

The problem thus becomes the sheer amount of graphic representations needed in order to create a complete Arabic font. Since the number of graphic representations is dramatically multiplied when using vowels, and since Arabic (to a native speaker) is easily understood without the vowels, we chose not to include vowels at this stage.

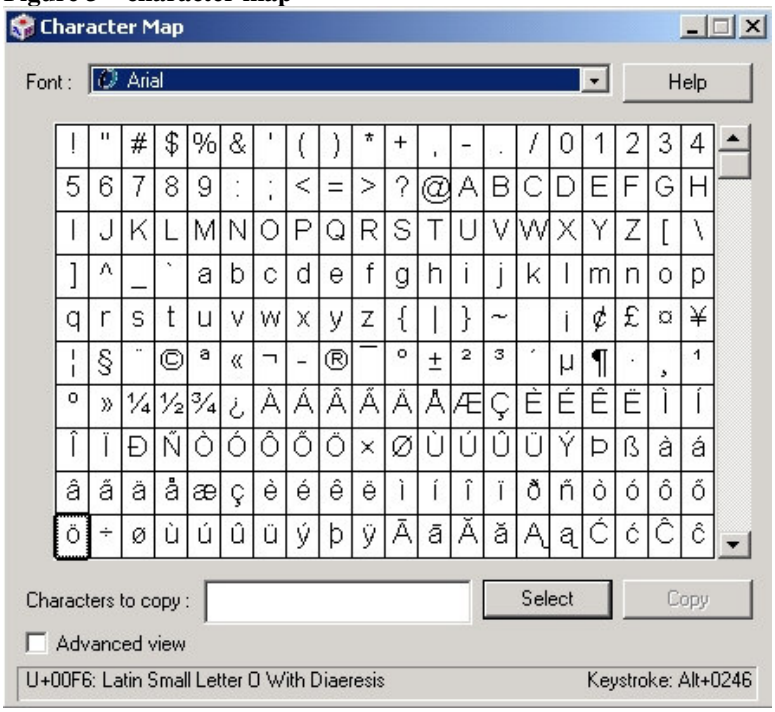
2.2 Arabic Encoding

Standard operating systems, such as MS Windows, use ANSI code for the Latin alphabet. This coding system uses a set of 8 bits to represent each character, which allows for 256 different symbols (2^8). Not all of these codes can be used, as several are reserved for special characters. Because of the special features of Arabic outlined above, the ASCII character set is not sufficient to display Arabic. It is therefore necessary to encode the characters into several character sets.

2.3 Hebrew-Enabled Operating-System

Since our primary survey language is Hebrew, we use a Hebrew-enabled operating system. On the one hand, this solves the problem of RTL writing in Arabic, as Hebrew is also written RTL. On the other hand, it further complicates the issue, by using a slightly different character set than that of a standard Latin alphabet. Some of the codes available in the Latin character set are reserved in the Hebrew character set and are therefore unavailable for use.

Figure 3 – character map



3. The Breakthrough

After careful analysis of the problems, we realized that some of the problematic features could in fact be used to our benefit. Since we anyway work with a Hebrew operating system there would be no need to account for Arabic's RTL direction. Furthermore, while Blaise cannot understand Unicode and requires an interpreter for such fonts, it has one unique feature that could aid us in our solution: Blaise allows mixing various font-types in a single line by enclosing each font in its correct font definitions respectively. Thus we were able to develop three separate fonts that together make up the complete Arabic alphabet and use the interpreter to translate Arabic texts into a string made up of all three fonts together.

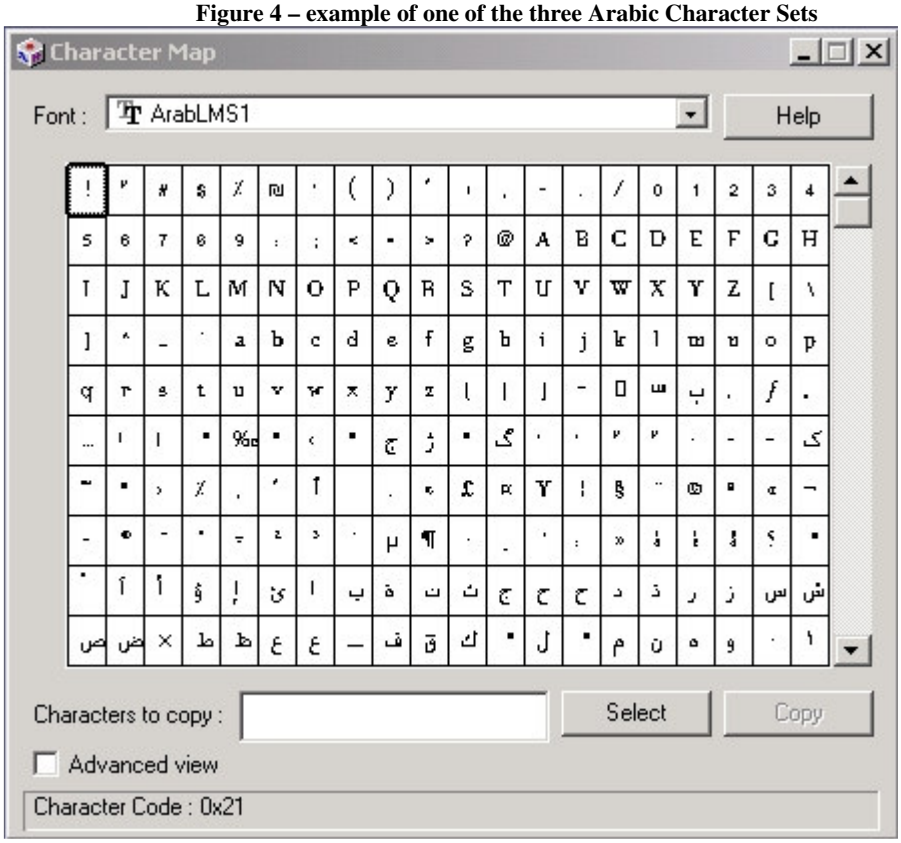
4. Our Solution for Displaying Arabic in Blaise

Our solution for displaying Arabic fonts in Blaise consists is twofold: First, designing three unique fonts that together make up a complete Arabic alphabet and, second, converting Arabic characters from MS Word into the new fonts designed for Blaise, including their Blaise font definitions.

4.1 Designing an Arabic Font

Since we are unable to use an Arabic operating system for the surveys, we required that a unique font be designed for Blaise. Over 500 separate symbols were needed (even without using vowels) and it was therefore necessary to assign 3 separate fonts to get a complete set of Arabic characters. The fonts also include numbers (Arabic numerals – 1,2,3…) and letters in English, since we sometimes have need for them. These fonts were designed uniquely for us and must be installed on all the computers that require Arabic in Blaise for both programming and surveying.

Figure 4 – example of one of the three Arabic Character Sets

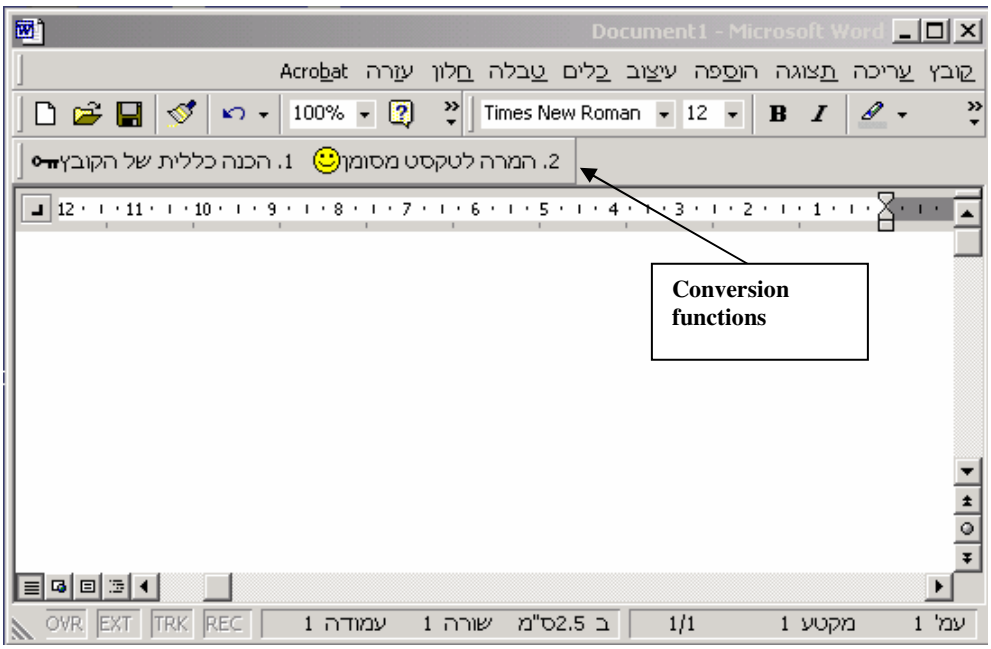


4.2 Converting Arabic from MS Word into the designated fonts for Blaise

In addition, it was necessary to develop an encoding program for Arabic, as it cannot be copied directly from MS Word into Blaise. This was achieved by developing a Word template that is able to convert the font as requested.
This file must be installed on all the computers that require Arabic for programming in Blaise. (Note that it is unnecessary to copy this file onto the laptop computers used for surveying.). The file is copied to:
C:\Documents and Settings*User*\ApplicationData\Microsoft\Word\STARTUP

Now, when we open a document in Arabic we have an extra toolbar with two functions necessary for converting the fonts.

Figure 5 – MS Word with conversion functions

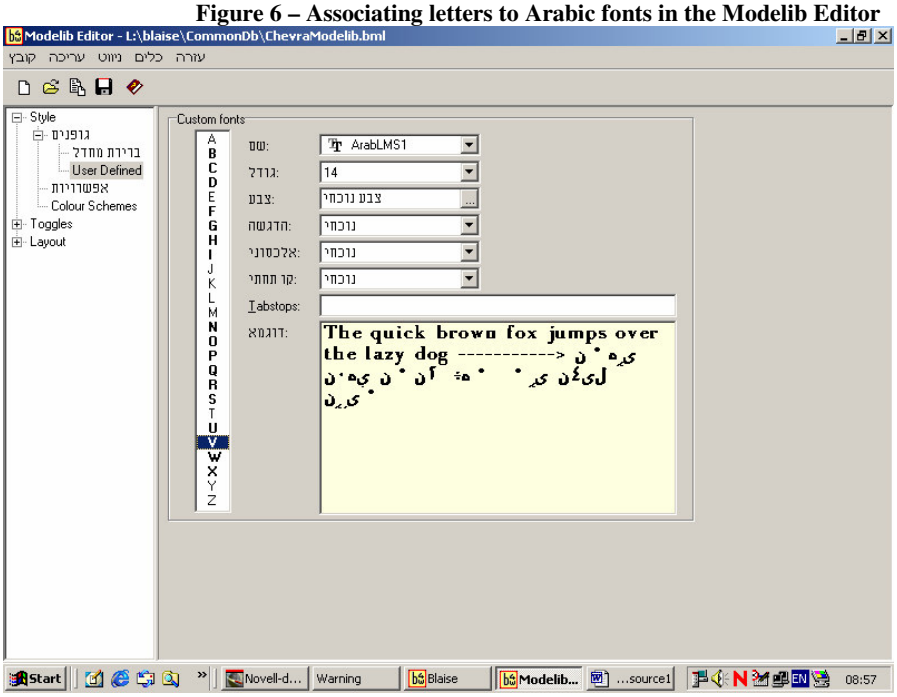


5. Using Arabic with Blaise

5.1 Defining The fonts in Blaise

We Associate the following three font letters in the Modelib Editor with the corresponding Arabic fonts:
X – ArabLMS1
Y – ArabLMS2
Z – ArabLMS3

Figure 6 – Associating letters to Arabic fonts in the Modelib Editor

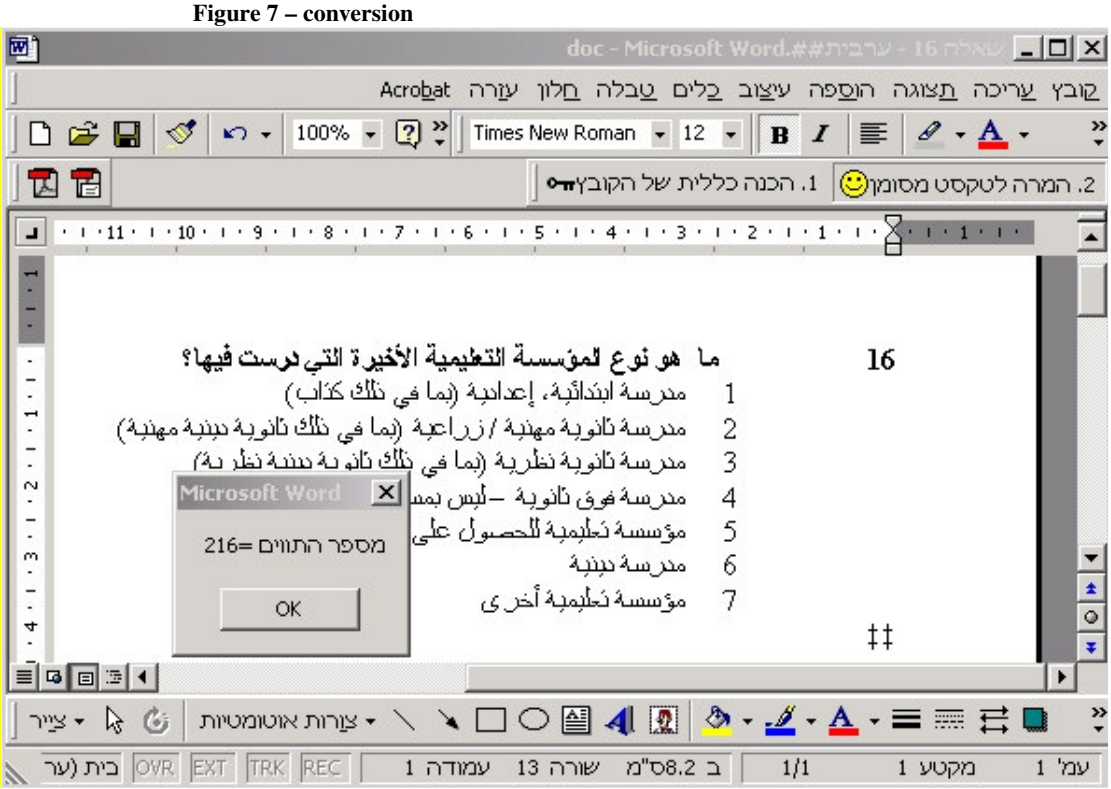


5.2 Copying Arabic text from Word to Blaise

In order to copy Arabic text from Word to Blaise we must use the special features developed for the conversion in word.
The first function *prepares* the entire text for conversion, so it is only necessary to perform this action once for any given document in Arabic.

The second function copies the selected sentence into the required font for Blaise. A popup window informs us of the length of the string:

Figure 7 – conversion

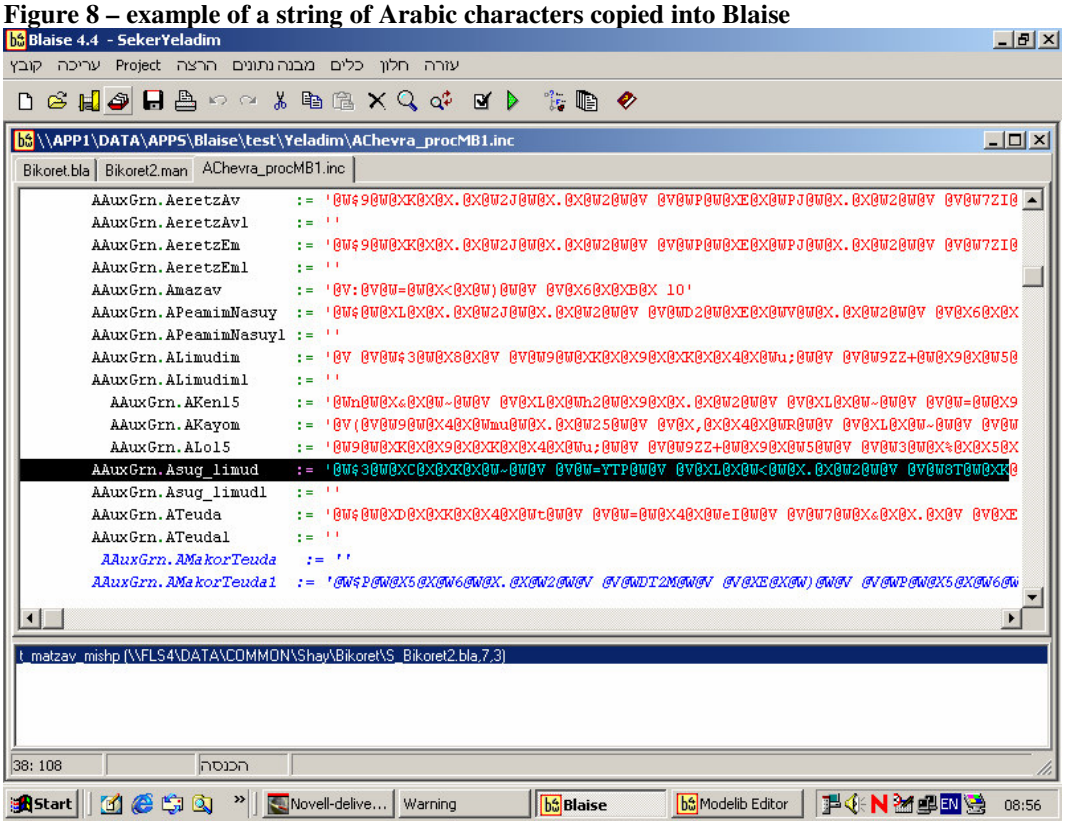


The string conversion in Word already includes not only the character itself but its corresponding font definitions for Blaise: @Xcharacter/s@X@Ucharacter/s@U@Wcharacter/s@W

Since the Arabic characters are split into 3 separate fonts, and each font requires a separate definition in Blaise, the output string resulting from the conversion is much longer than the original. Every section, or even a single letter at times, is surrounded by its definitions.

Next, the string is pasted into Blaise. There is no need to enclose it with font definitions as these are already part of the conversion. The length is defined according to the number displayed at the time of conversion.

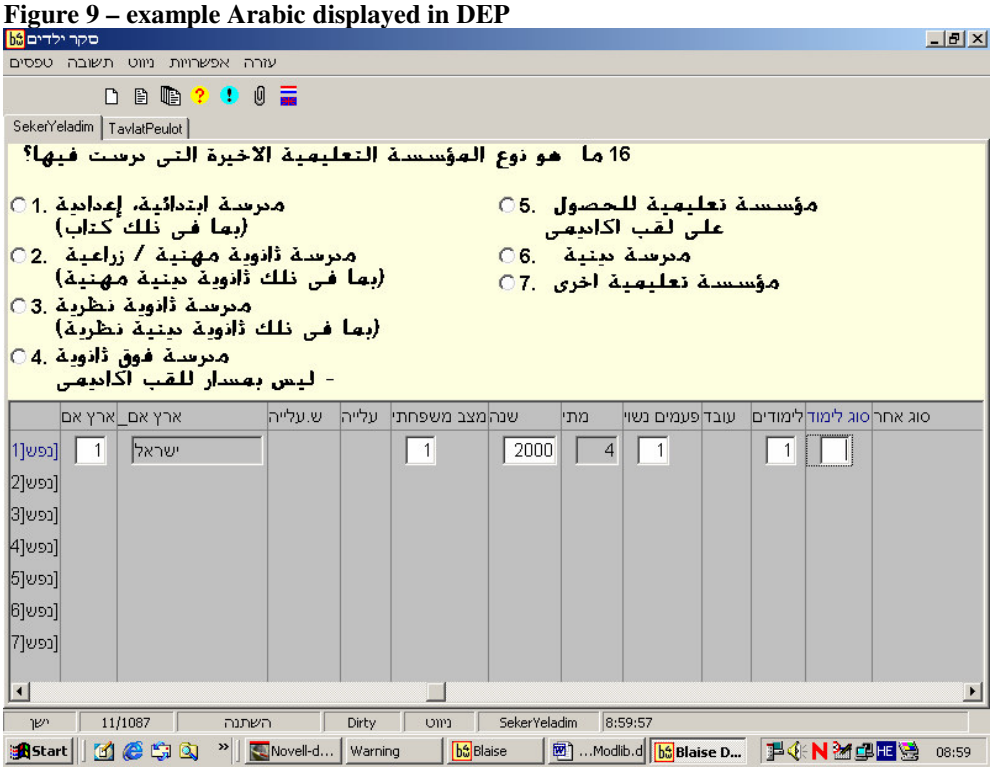
Figure 8 – example of a string of Arabic characters copied into Blaise



5.3 Displaying Arabic in the Data Entry Program

The above example of a string of Arabic characters will appear thus in the questionnaire itself:

Figure 9 – example Arabic displayed in DEP



Both Arabic and Russian are defined as parallel language blocks, so it is possible during the survey to switch between each of the languages at any given point.

6. Summary and Conclusions

Finding a solution for displaying Arabic in Blaise has been a top priority for Israel's Central Bureau of Statistics over the past few years. Arabic is one of Israel's two official languages and it became crucial that we be able to display it properly once we began using CAPI. The complications arose from the number of symbols required for encoding Arabic characters, which is greater than those available in a standard 256 ANSI character set. The difficulties, however, were overcome by designing b`oth a set of fonts and a converter from Word into Blaise. We have begun using Arabic characters in our surveys as of 2004 and are pleased with the results.

Currently we display Arabic with no vowels. Further development may be required in order to include vowels: a complete set of Arabic characters with vowels will require several *thousand* symbols. However, at present we find there is no need for displaying the vowels.

Another aspect of our present solution which might be problematic is the length of strings of Arabic characters. Since each string consists of three different fonts intermingled, and each font must be enclosed within its Blaise definitions, the strings end up being quite lengthy. The texts in Arabic, therefore, require more memory than the parallel text in our other survey languages.

7. References

Groeneveld, R., (2003) "Using non-Latin alphabets in Blaise", Proceedings of the 8th International Blaise Users Conference

Cheung, G.Q. and Liu, Y., (2003) "Displaying Chinese Characters In Blaise", Proceedings of the 8th International Blaise Users Conference

http://diwww.epfl.ch/w3lsp/conferences/ridt98/decotype.html

http://members.aol.com/gnhbos/meissues.htm

Codification automatique en ligne

HUMBERT Stéphane, INSEE, France

1. Introduction

L'ensemble des enquêtes auprès des ménages de l'INSEE sont réalisées grâce à une application CAPI qui permet d'intégrer les différents questionnaires écrits en Blaise. Certaines variables telles que la profession sont collectées dans des zones de textes qui permettent à un enquêteur de relever la réponse exacte donnée par l'enquêté. Elles doivent donc être codifiées systématiquement pour pouvoir être exploitées.

L'opération était jusqu'à présent réalisée à l'INSEE après la collecte en deux étapes :

- une codification automatique à l'aide d'un outil (SICORE) assurant d'une part la reconnaissance des libellés et d'autre part l'application de règles de codage s'appuyant sur d'autres variables appelées variables annexes,
- puis une reprise dans un service de l'INSEE pour les libellés qui ne peuvent être codifiés automatiquement.

Il y avait alors un nombre important de libellés envoyés en reprise et l'ensemble des variables annexes pouvant éventuellement servir à la codification devaient être collectés. Pour réaliser des gains de productivité et de délai, la codification automatique de ces variables a été insérée sur le poste de collecte.

Après une rapide présentation du fonctionnement de l'outil SICORE, nous verrons quelles fonctionnalités ont été développées sur le portable des enquêteurs. Nous expliquerons ensuite comment ces fonctionnalités ont été implantées d'un point de vue technique. Enfin, nous exposerons quelques applications de cette codification.

2. Présentation de SICORE :

SICORE est un outil qui permet de codifier dans une nomenclature des libellés saisis en toutes lettres en s'appuyant sur des bases de connaissance. Cette codification s'effectue en deux étapes. La première étape est une reconnaissance du libellé qui s'appuie sur une recherche en bigramme. En cas de succès de cette première phase, un précode est obtenu. La deuxième étape consiste alors à affecter un code final à partir d'une table de décision indiquée par le précode obtenu lors de la première étape. Cette table de décision donne les règles à suivre pour réaliser la codification à l'aide d'éventuelles variables annexes. Si toutes les variables annexes utiles ont bien été collectées, on obtient en final une codification.

Un "serveur SICORE" a été installé sur le portable de l'enquêteur pour permettre à Blaise d'accéder à ces fonctionnalités. Ce serveur fonctionne de la façon suivante : au démarrage, les bases de connaissance sont chargées en mémoire vive. Le serveur reste alors en attente des demandes de connexion de postes clients. Lorsqu'une telle connexion est établie, il reçoit les requêtes en provenance du client, effectue ces requêtes et renvoie le résultat au client. Du fait du chargement de l'environnement en mémoire vive la réponse est quasi-instantanée sur un portable enquêteur.

La requête et la réponse à la requête sont formées de chaînes de caractères aux formats prédéfinis. La chaîne de la requête contient notamment le nom de l'environnement SICORE à utiliser pour effectuer la codification, le libellé à codifier et la liste des variables annexes renseignées avec leur valeur. La chaîne de la réponse contient quant à elle le résultat de la codification, et éventuellement le précode, le code final et la liste des variables manquantes (variables dont SICORE a eu besoin et qui n'étaient pas dans la liste précédente).

3. Fonctionnalités de l'outil de codification en ligne

Avec la codification en ligne des variables, nous avons cherché à exploiter au mieux les possibilités offertes par SICORE. La codification en ligne permet ainsi de s'assurer de la reconnaissance du libellé, de vérifier la qualité du libellé, de ne poser que les variables annexes utiles à SICORE et enfin d'obtenir le code final.

3.1 Reconnaissance du libellé par SICORE

Lorsqu'un libellé est saisi ou modifié par l'enquêteur, une requête est envoyée au serveur SICORE afin de contrôler si le libellé est reconnu et pourra donner lieu à une codification. En retour, on exploite le code résultat SICORE pour savoir si oui ou non le libellé a été reconnu : si le libellé a été reconnu par SICORE, la codification pourra avoir lieu. Sinon, on indique à l'enquêteur que le code n'est pas reconnu et on lui propose de vérifier sa saisie.

3.2 Vérification de la qualité des libellés

Certains libellés peuvent être reconnus par SICORE mais peuvent ne pas être satisfaisants car ils ne donnent pas une information suffisamment précise pour aboutir à une codification satisfaisante.

Ainsi si on prend l'exemple des libellés de profession, « OUVRIER » est reconnu par SICORE mais il n'est pas très précis.

Pour éviter ce problème qui risque de faire perdre de l'information au moment de la collecte, une phase de vérification du précode a donc été introduite. Le principe retenu pour effectuer cette opération est de vérifier que le précode fourni par SICORE n'est pas dans une liste de précode à proscrire. Si le libellé conduit à utiliser une table de connaissance réputée peu précise, on demande à l'enquêteur de préciser le libellé qu'il vient de saisir.

3.3 Questionnement sur les variables annexes

Après la phase de reconnaissance du libellé, SICORE s'appuie sur une table de décision pour obtenir la codification finale à l'aide de variables annexes.

Ainsi, pour codifier la profession, SICORE a besoin de 14 variables annexes. En pratique, SICORE n'a pas toujours besoin de l'ensemble de ces variables pour effectuer la codification. Une seule de ces variables peut parfois suffire à obtenir le code final.

Une exploitation du résultat de la requête envoyée à SICORE qui contient notamment la liste des variables annexes manquantes, permet de ne poser que les questions nécessaires. Pour cela, on récupère la première variable manquante non posée et on pose à l'enquêté la ou les questions qui correspondent à cette variable SICORE puis, on effectue à nouveau un appel à SICORE. Cette opération est répétée autant de fois que nécessaire, c'est à dire tant qu'on a des variables annexes manquantes qui correspondent à une question présente dans le questionnaire de l'enquête.

Parmi les variables nécessaires à la codification, le traitement n'est pas toujours identique. En effet, certaines variables sont de toutes façons nécessaires pour l'enquête alors que certaines ne sont présentes que pour codifier la variable. Pour chaque variable, on peut donc décider au moment de la conception d'une enquête si elle est toujours posée ou si elle n'est posée que dans le cas où elle serait nécessaire à la codification.

3.4 Obtention du code Final

Lorsque le libellé est reconnu par SICORE et que toutes les variables annexes nécessaires ont été posées, on récupère le code final correspondant à ce libellé.

4. Réalisation technique

Outre l'utilisation du serveur SICORE, le développement d'un tel outil s'est appuyé d'une part sur des développements en DELPHI et d'autre part sur des développements Blaise. La partie développée DELPHI a en charge les interactions avec le serveur et la partie développée en Blaise permet d'assurer le questionnement, la mise en forme de la requête pour SICORE et l'exploitation de cette requête.

4.1 Développements DELPHI pour l'intégration de SICORE dans CAPI

L'interaction avec le serveur SICORE comprend 3 opérations : démarrer le serveur, l'éteindre et créer un poste client qui envoie une requête au serveur.

Le lancement de ces opérations est géré dans les programmes Blaise par l'intermédiaire de deux librairies nommées capi_sic.dll et Sicore.dll. Pour assurer la communication entre les programmes Blaise et la couche DELPHI, des fonctions et méthodes de la librairie BLDllObj.dll ont été utilisées.

Capi_Sic.dll contient les deux procédures de lancement et de fermeture du serveur SICORE. Ces deux procédures sont lancées par un manipula qui permet à l'enquêteur de réaliser la collecte. Ainsi au moment où l'enquêteur accède à la liste de ses questionnaires, la procédure de lancement du serveur est activée. Cette procédure renvoie un code retour pour indiquer si l'opération s'est passée correctement. De même à la fermeture du questionnaire, la procédure de fermeture est lancée.

Sicore.dll contient quant à elle la procédure qui crée un poste client, envoie le message de la requête et réceptionne le message indiquant le résultat de la requête. Cette procédure reçoit en entrée le libellé de profession, la liste des variables annexes déjà renseignées pour SICORE et le nom de l'environnement à utiliser. Elle écrit alors le message de la requête en respectant le format attendu par SICORE et envoie cette requête. Lorsqu'elle réceptionne le résultat de cette requête, elle renvoie au programme Blaise différentes chaînes de caractères contenant le résultat de la requête, le précode, la liste des variables manquantes et le code final s'il existe.

4.2 Développements Blaise pour l'intégration de SICORE dans CAPI

Du côté Blaise, différentes procédures ont été écrites pour assurer la communication avec la procédure DELPHI, la mise en forme des informations pour SICORE et l'exploitation du résultat renvoyé par SICORE. En plus de l'écriture de ces procédures, il a fallu adapter le questionnaire.

4.2.1 Communication avec DELPHI

Pour assurer l'échange entre DELPHI et Blaise, on utilise une méthode « **ALIEN** » dans une procédure spécifique. Cette procédure reçoit en entrée toute l'information qui doit être transmise à la procédure DELPHI et réceptionne en retour toute l'information que la procédure DELPHI doit retransmettre à Blaise. Elle assure donc la passerelle entre le datamodèle Blaise et le programme DELPHI. Elle s'écrit de la façon suivante :

```
PROCEDURE CallPrecodage
    PARAMETERS
        IMPORT
            LibProfession                : STRING

  ...

        EXPORT
            CodeSicore                : STRING
        EXPORT
            CodeFinal                 : STRING

        ALIEN('C:\Proggen\dll\SICORE.DLL','Precodage_CS')
ENDPROCEDURE
```

4.2.2 La mise en forme des données pour SICORE

Pour fonctionner, SICORE attend d'une part, une chaîne de caractères correspondant au libellé de profession et d'autre part, une chaîne de caractères lui indiquant quelles sont les valeurs prises par chaque variable annexe dont il a besoin.

La deuxième chaîne de caractères se présente sous la forme suivante : « variable1=valeur1, variable2 = valeur2… ». Dans le questionnaire de l'enquête, chaque variable SICORE correspond à une ou plusieurs questions pouvant prendre différentes modalités. Une procédure de transcodification a donc été écrite pour chaque variable annexe utilisée par SICORE. Ces procédures reçoivent en entrée une liste de variables et donnent en sortie la chaîne de caractères correspondante.

Ainsi, si on prend l'exemple de la profession, SICORE a besoin de savoir si la personne travaille dans une entreprise publique ou privée pour obtenir certains codes. Dans le questionnaire, cette information peut être obtenue par la variable Statut. La procédure s'écrit alors de la façon suivante:

```
PROCEDURE TransCodePUB

    PARAMETERS
        IMPORT
            Statut : StatuType
        EXPORT
            Libelle  : STRING

    RULES
            Libelle := ''
            IF STATUT = etat THEN
                Libelle := 'PUB=1,'
            ELSEIF STATUT = coll THEN
                Libelle := 'PUB=2,'
            ELSEIF STATUT IN [entr,part,fam,chef,indep] THEN
                Libelle := 'PUB=5,'
            ELSEIF STATUT = DK OR STATUT=RF THEN
                Libelle := 'PUB=*,'
            ENDIF
ENDPROCEDURE
```

Pour le libellé de profession, les bases de connaissance de SICORE ne prennent pas en compte d'éventuels accents. Une procédure a donc été écrite pour convertir la chaîne de caractère en une chaîne en majuscule sans accent. Cette procédure supprime également certains caractères spéciaux non pris en compte par SICORE.

4.2.3 L'exploitation des données fournies par SICORE

Une procédure générale gère les différentes opérations liées à l'intégration de SICORE. Elle permet ainsi la mise en forme de la liste des variables annexes et du libellé, l'envoi des informations à la procédure assurant l'interface entre Blaise et DELPHI, la vérification de la validité du libellé, la mise à jour de la liste des variables annexes à poser et l'affectation du code PCS obtenu.

Elle reçoit en paramètres le libellé de profession, l'ensemble des variables annexes ainsi que des variables « Pose_var » indiquant si une variable donnée est posée ou non. Après avoir effectué les différentes étapes, elle renvoie le résultat de la codification, les variables Pose_Var mises à jour et éventuellement le code final.

La première phase consiste en l'appel des différentes procédures de mise en forme décrites précédemment et la constitution des chaînes de ces chaînes de caractères pour l'envoi à SICORE.

La deuxième phase consiste à appeler la procédure assurant l'interface entre Blaise et DELPHI. Pour cela, on fournit le libellé de profession reçu en paramètre, la liste des variables annexes mise en forme, l'environnement à utiliser ainsi que les tailles des différents éléments renvoyés par SICORE. Pour une variable à codifier donnée, les valeurs de ces différents paramètres sont des constantes stockées au niveau de la procédure.

Les informations fournies en retour par la procédure assurant l'interface entre Blaise et DELPHI sont récupérées dans des variables locales afin d'être exploitées dans les étapes suivantes. Pour cela, la procédure vérifie d'abord la validité du libellé, puis exploite la liste des variables annexes manquantes et enfin renvoie le code final.

La vérification de la validité du libellé se découpe en deux parties : on interprète le « code retour SICORE » et on vérifie si le précode appartient à une liste de précodes « ambigus ». Le « code retour SICORE » permet de savoir si la reconnaissance du libellé a pu aboutir et s'il manquait ou non des variables annexes pour effectuer la codification finale. Lorsque le code retour SICORE indique un échec, on stocke dans la variable Res_Sicore la nature de l'échec. Si le résultat indique une reconnaissance du libellé par SICORE, on vérifie si le précode est satisfaisant ou non. Cette phase consiste à vérifier si le précode correspondant au libellé appartient ou non à une liste fournie dans un fichier externe. S'il est trouvé dans cette liste alors on affecte à Res_Sicore une valeur indiquant que ce libellé est ambigu, sinon le libellé est considéré comme bon. Cette variable sera retournée au bloc de questionnement et sera affichée après la saisie du libellé. Les différentes modalités de Res_Sicore sont ainsi :

- **LibOk** "Le libellé est reconnu et pourra donner lieu à une codification.",
- **LibAmb** "Le libellé est reconnu mais il est ambigu. Il ne pourra donc pas donner une codification précise.",
- **LibNon** "Le libellé n'a pas été reconnu.",
- **Pbxxx** "Un incident s'est produit durant l'opération. Par exemple, non connexion au serveur",

Lorsque le libellé est reconnu, on exploite la liste éventuelle de variables annexes pour mettre à jour la liste des questions à poser dans le questionnaire. Cette opération est la plus complexe à gérer. En effet, lorsqu'une variable est manquante, SICORE prend une valeur par défaut pour continuer sa

codification et trouver si d'autres variables sont manquantes. Ceci implique que, suivant la valeur qu'on donne à cette variable, le cheminement peut être différent et SICORE peut avoir besoin d'une variable qui n'était pas indiquée dans la liste précédente. Il faut donc consulter la liste fournie par SICORE et mettre à jour la liste des questions à poser en ajoutant une seule question supplémentaire à chaque appel. De même, en cas de modification de valeur, si une variable est déjà déclarée comme étant à poser mais n'a pas encore été renseignée, il ne faut pas rajouter une nouvelle variable.

Pour gérer la liste des variables à poser, une variable Pose_Var à été créée pour chaque question. Elle peut prendre 3 valeurs : « Oui », « Non » ou « déjà posé ». « Oui » indique que la question est à poser mais n'est pas encore renseignée. « Non » indique qu'elle n'est pas posée et « déjà posé » indique qu'elle est renseignée.

On commence d'abord par regarder si on a une valeur de Pose_Var à « Oui ». Si c'est le cas, on ne modifie pas les valeurs des autres variables Pose_Var. Sinon, on consulte la liste des variables annexes manquantes fournie par SICORE. On récupère en fait une chaîne de caractères contenant différents noms de variables séparés par des espaces. Il faut tout d'abord isoler chaque mot contenu dans cette chaîne. Ensuite, on analyse les variables manquantes dans l'ordre d'apparition : Pour la première variable rencontrée, si la question permettant d'obtenir cette information n'est pas posée et existe dans le questionnaire, on bascule la variable Pose_Var correspondante à « Oui » et on arrête l'exploitation de la chaîne de caractère. Sinon, on passe à la deuxième variable et on continue jusqu'à trouver une nouvelle variable à poser ou la fin de la chaîne de caractère.

Lorsque le code retour SICORE nous indique que la phase de codification a eu lieu sans ambiguïté (cas où la liste des variables annexes est vide), la procédure renvoie dans une variable le code obtenu. Ce code pourra alors être affiché dans le bloc ou simplement affecté dans une variable récupérée en sortie.

4.2.4 L'adaptation du questionnement à la codification

Il s'agit d'une adaptation d'un bloc de questionnaire Blaise classique permettant de prendre en compte les appels à SICORE.

Concernant la structure, plusieurs variables ont été ajoutées pour gérer le questionnement et les modifications.

Ainsi, les variables Pose_Var, indiquant si une question est à poser ou non, ont été ajoutées pour chaque question.

De plus, le lancement de SICORE s'effectue uniquement lorsqu'une variable utile à la codification a été modifiée pour ne pas alourdir les temps de réponses. Pour cela, chaque variable utile à la codification apparaît en double, la première variable étant la question posée et la deuxième une variable prenant la même valeur. Pour vérifier si on a eu une modification, on teste si ces deux variables sont égales : si elles ne sont pas égales, cela signifie que la question a été modifiée ; on lance alors un appel à SICORE avant d'affecter la valeur de la question dans l'autre variable.

Une variable SIC_INFO a également été ajoutée après la saisie du libellé pour indiquer le résultat de l'appel à SICORE. Elle permet de dire à l'enquêteur s'il doit ou non vérifier le libellé.

Enfin, une variable Code_var a été ajoutée pour stocker le résultat de la codification.

Concernant le déroulement du questionnaire, les règles de routage doivent également être adaptées.

En début de bloc, on commence par affecter toutes les valeurs de pose_Var à la valeur choisie par défaut : « non » dans le cas courant, « déjà posée » si cette variable doit toujours être posée pour les besoins de l'enquête.

Une fois le libellé et le premier appel à SICORE réalisés, la question SIC_INFO indique à l'enquêteur si le libellé a été reconnu ou non. Selon la valeur de cette variable, on se retrouve ensuite soit dans un déroulement classique (libellé non reconnu ou ambigu), soit dans le nouveau déroulement (libellé reconnu).

Dans le nouveau déroulement, chaque question est posée si la valeur de Pose_Var correspondante vaut « oui » ou « déjà posée ». Ces valeurs sont mises à jour par la procédure décrite précédemment.

De plus, un test est présent au niveau de chaque question pour vérifier si le libellé est modifié. S'il est modifié, la procédure d'appel à SICORE est lancée.

Le filtrage des questions par les variables Pose_Var implique que les questions sont posées dans l'ordre où SICORE en a besoin. Cet ordre n'est pas toujours le même que l'ordre « normal » du questionnaire. Il est donc possible d'avoir des « retours en arrière » dans le questionnement : on pose la troisième question du bloc puis la deuxième. Dans la pratique, les questions sont souvent posées dans le même ordre. Cependant, une variable Pause a été programmée pour indiquer à l'enquêteur un éventuel retour en arrière dans le questionnement.
Grâce à l'utilisation de procédures standards gérant tous les aspects de communication avec SICORE, l'adaptation du questionnement consiste simplement à faire appel à la procédure de codification et à filtrer le questionnement en fonction des besoins de SICORE. L'adaptation du questionnement est donc relativement simple.

5. Applications de la codification en ligne.

Le développement de la codification en ligne étant récent, les utilisations de ce procédé sont pour l'instant peu nombreuses mais elles permettent déjà de voir les possibilités qu'offre un tel outil. Le procédé a ainsi été appliqué à deux domaines : le domaine de la profession et le domaine des produits de consommation. Ces deux exemples sont intéressants car la problématique n'est pas forcément similaire : le premier domaine s'appuie sur un nombre important de variables annexes alors que dans le deuxième, aucune variable annexe n'est pour l'instant utilisée. Les efforts d'adaptation se concentrent donc uniquement sur le libellé et l'obtention d'un code le plus précis possible.

5.1 Les libellés de profession

Cette variable est présente dans toutes les enquêtes réalisées par l'INSEE. Pour les exploitations, un code CS ou PCS est utilisé. Ce code est obtenu à partir du libellé de profession et d'une liste de 14 variables annexes. Jusqu'à présent, la codification de masse en fin de collecte permettait d'obtenir un taux de codification automatique de l'ordre de 75 à 80%. Les libellés non codés étaient ensuite envoyés "en reprise" dans un service de l'INSEE qui pouvait traiter ces cas selon 3 grandes méthodes. Si l'échec de codification était dû à une erreur de saisie de libellé, il était modifié et on relançait un appel à SICORE. Si l'échec était dû à une incohérence dans la liste des variables annexes, celles-ci étaient modifiées et SICORE était également relancé. Enfin, si l'échec était dû à une "non-connaissance" de SICORE (le libellé est bon mais n'est pas dans les bases de connaissance), la codification était effectuée manuellement avec l'aide d'un expert qui relevait le cas pour une éventuelle mise à jour des bases de connaissance.

L'outil de codification en ligne a été utilisé sur l'enquête SILC pour la reconnaissance du libellé de profession. L'utilisation de cet outil a permis d'obtenir un taux de codification automatique de 90% et donc de diviser par deux les cas de traitement en reprise et les délais dus à cette opération.
Un bloc de questionnement standard a été développé pour l'ensemble des enquêtes et il est intégré dans la plupart des enquêtes à venir de l'INSEE. Suivant l'importance de la profession dans le thème de l'enquête, les concepteurs choisissent soit de poser toutes les questions sur la profession, soit de ne poser que les questions utiles à SICORE. Dans le premier cas, ils n'utilisent que le module de reconnaissance du libellé pour limiter la reprise alors que dans le deuxième cas, ils limitent également le temps du questionnement nécessaire pour obtenir le code CS désiré. Cette deuxième version est appréciée par les enquêteurs car pour une variable comme la NAF2 (NACE). En effet, cette variable est posée de façon standard dans les enquêtes sous forme d'une classification hiérarchique mais elle reste cependant assez compliquée à obtenir.

Les gains attendus de l'utilisation de SICORE dans le cas du libellé de profession sont donc d'une part une diminution du temps de reprise et d'autre part une diminution du temps de questionnement pour l'enquêteur.

5.2 Les produits de consommation et magasins

Pour l'enquête "Budget de Famille 2005", des carnets de consommation sont remplis par les personnes enquêtées puis saisis par l'enquêteur qui peut notamment leur demander des compléments d'information si nécessaire. Dans ces carnets de consommation, on note un libellé de dépense. Ce dernier est ensuite codifié par SICORE selon la nomenclature COICOP. Le problème rencontré lors de l'édition de l'enquête était que le niveau de précision donné par le ménage n'était pas forcément celui attendu par SICORE. Le libellé pouvait être riche tout en n'ayant pas forcément tous les éléments nécessaires. De plus, dans certains cas, le manque de précision pouvait obliger à remonter assez haut dans la nomenclature pour finalement obtenir un code très partiel.

Avec la codification en ligne, le code obtenu par la codification est exploité dès la collecte. Il permet de demander à l'enquêteur de préciser le libellé ou d'améliorer la codification.

Pour assurer une meilleure aide aux enquêteurs, des codes supplémentaires ont été rajoutés dans la nomenclature pour traiter des cas particuliers. Lorsque la codification en ligne aboutit à un code, on vérifie si ce code est présent dans un fichier externe. On peut récupérer dans ce fichier soit une aide spécifique, soit une liste de cas clairement identifiée permettant à l'enquêteur d'améliorer la qualité du code final qu'on obtiendra. Par exemple, si l'enquêteur saisit "boisson", l'aide va lui indiquer de préciser le type de boisson et notamment s'il s'agit d'une boisson alcoolisée ou non alcoolisée. L'enquêteur peut alors modifier le libellé en conséquence.

En cas de codification partielle, une autre solution est proposée à l'enquêteur : il s'agit de terminer la codification de façon manuelle sans modifier le libellé. Pour cela un programme DELPHI a été développé. Il permet de parcourir la nomenclature en rentrant au niveau du code partiel obtenu à l'aide du libellé et donc de compléter la codification. Par exemple, si l'enquêteur saisit "pantalon", le code partiel obtenu correspondra à "vêtement". Si l'enquêteur qui est en contact avec le ménage sait si ce pantalon est destiné à un homme, une femme ou un enfant ; il peut soit compléter le libellé soit terminer directement la codification.

L'utilisation de la codification en ligne dans ce cas nous permet donc d'envisager une amélioration de la qualité de l'information saisie et donc de limiter d'une part le travail de reprise et d'autre part le travail d'imputation. Pour l'instant, un seul test a été réalisé. Il ne permet pas de conclure sur les gains chiffrés d'un tel outil. On peut néanmoins déjà dire que les principes mis en œuvre sont appréciés par les enquêteurs.

6. Conclusion

L'utilisation de cet outil de codification en ligne nous permet donc d'envisager un gain de temps en limitant les questions posées d'une part et en diminuant les échecs de codage d'autre part. Il permet également d'améliorer la qualité du libellé saisi en donnant à l'enquêteur des consignes adaptées pour traiter des cas particuliers.

Cependant, cet outil modifie la façon de travailler des enquêteurs et il faut être vigilant sur les consignes données pour l'utilisation. En effet, le but n'est pas d'obtenir 100% de codage automatique. Un libellé n'est pas forcément faux parce qu'il n'est pas reconnu par SICORE : il peut s'agir d'une nouvelle dénomination non prise en compte par les bases de connaissance. Dans ce cas, il faut bien sûr laisser le libellé pour permettre d'enrichir ces bases. Il ne faut corriger le libellé que dans les cas où ce dernier est trop vague ou saisi avec une erreur. Le premier libellé saisi est pour l'instant conservé afin d'analyser la nature des modifications effectuées par l'enquêteur.

Enfin, l'utilisation d'un tel outil implique un investissement important pour la création des bases de connaissance. Cette opération est d'autant plus compliquée dans le cadre d'un nouvel environnement que les bases de connaissance doivent être réalisées avant la collecte.

Blaise Instrument Design for Automated Food Coding

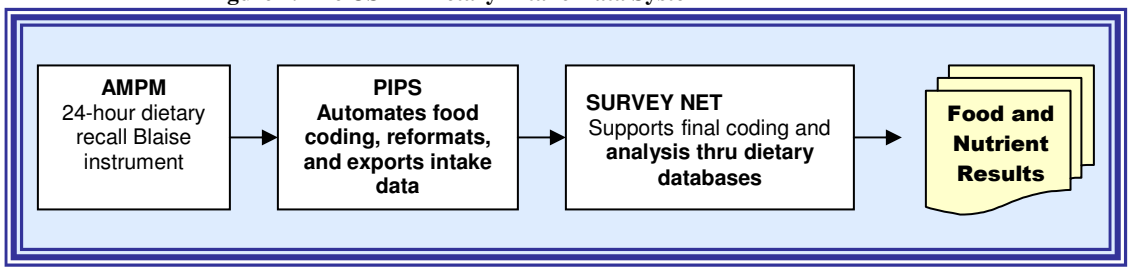
Ellen Anderson and Lois Steinfeldt, United States Department of Agriculture, Agricultural Research Service, Beltsville Human Nutrition Research Center, Food Surveys Research Group

1. Introduction

The Food Surveys Research Group of the United States Department of Agriculture has developed automated methods for collecting and processing food intake data. These methods are part of the Dietary Intake Data System (Figure 1), designed to efficiently collect and process high quality food intake data. The foundation of the system is the Automated Multiple Pass Method (AMPM) Blaise instrument, which is used to collect 24-hour dietary recalls. 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 as well as a description of the amount consumed. Information is also collected about the time of day the food was eaten, the name of the eating occasion, and where the food was obtained.

From the AMPM Blaise database, the intake data are extracted, reorganized, automatically coded, and reformatted by the Post Interview Processing System (PIPS). PIPS was developed with Visual Basic and the Blaise Application Programming Interface (API). These data are reformatted to be used by Survey Net, the computer-assisted food coding and dietary data processing system developed for use with USDA nationwide food consumption surveys. 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 U.S. population.

Figure 1. The USDA Dietary Intake Data System



Coding survey data via customary means (e.g. trained coding experts) often requires an inordinate amount of resources and may, at times, produce results that are subjective and erroneous (Speizer and Buckley, 1998; Biemer and Lyberg, 2003). Food coding can be especially challenging due to the variety of food items available in the U.S. marketplace and the diversity of the U.S. population. Automation of the food coding process may be a way to decrease data processing time and increase consistency and quality of food consumption data.

While the AMPM Blaise instrument was designed to facilitate the food intake interview, also considered was how the food detail data stored in the Blaise database would be extracted for manual and automated coding. With our method, automated coding of foods is accomplished as a data matching process between (a) the food question variable names and response values as reported in the AMPM interview, and (b) data in the Autocode table, which is part of an autocode pathways database. The series of questions and responses that describes a food establishes a food path for that food. If an exact match is found, then the food is automatically assigned a code. This method of automated coding was recently implemented as part of PIPS.

This paper will discuss how the design of the AMPM Blaise instrument facilitated the development and implementation of automated food coding in PIPS, as well as report on the successes and pitfalls of the automated coding process encountered thus far.

2. Overview of the AMPM Blaise instrument design

The AMPM Blaise instrument contains more than 2500 questions and more than 21,000 responses. Ninety-five percent of the questions are about specific food details, including the amount of food consumed. The types of questions asked for a food is determined by the food category assigned to that food. Foods were grouped into 132 categories represented by a unique 5 or 6 digit code. Each food category contains a series of questions designed to elicit specific details about that particular food. Certain questions are only asked of certain foods. In addition, certain questions may be skipped depending on the response to previous questions. Table 1 lists a variety of food categories and the number of food detail questions, not including questions about portion size, for each category. Collecting the proper level of detail for specific foods is an important first step in accurately coding food data. 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.

Table 1. Number of food detail questions (excluding portion size questions) associated with selected food categories

Food category	Number of food detail questions
Soda, pop, soft drinks	3
Milk	4
Yogurt	4
Cheese	8
Green salads	8
Infant formulas	8
Fruits, berries	10
Coffee	11
Mixed dishes	17
Meat sandwiches	45

Variable names for each question within a category were given a standard prefix. Initially, this was done to help the interviewer and food coder interpret the meaning of the variable names. For example, questions for the fruit category begin with the word “Fruit”. Across food categories, the same question was given a standard suffix. For instance, the question “What kind was it?” in the fruit category is represented by the variable name “FruitKind”. In the bread category, the variable name “BreadKind” represents the same question for bread. In addition to the variable name, each question in a food category is assigned an item number that indicates the order in which questions are asked. The item number consists of a three character mnemonic and a three digit number. Table 2 shows an example of these fields for food category 60060, yogurt. For the majority of food categories, item numbers were assigned systematically. Questions dealing with food details were assigned item numbers ending in 499 or less. The item number ending in 500 represents the standard question, “Did you add anything to the food?” Item numbers ending in 600 or greater represent questions about the amount of food consumed. The few exceptions to the item numbering order appear in the more complicated food categories such as sandwiches and salads. By standardizing the variable names and item numbers as much as possible, the process of identifying food detail questions that are important for automated coding was simplified.

Like the question variable names, the response values were written to be understandable to both the interviewer and the food coder. Open-ended responses were limited to “Other Specify” fields by the use of more than 90 lookup tables. The ready-to-eat cereal lookup table, for example, currently contains 289 possible responses. If the instrument required interviewers to enter responses into

open-ended fields, it is likely the same answer could be recorded in several different ways. For example, "Cap'n Crunch®" cereal could be entered as "Captain Crunch" or "Capn Crunch" or "Captain Krunch", etc. With one selectable option for "Cap'n Crunch®", the potential number of food paths is reduced. The standardization of questions and answers throughout the AMPM Blaise instrument proved to be important for implementing an autocoding system based on this exact text matching method. Limiting the variability of response entries helped to keep the food paths consistent and eliminated the need for complex parsing and decision making algorithms.

Table 2. Example of AMPM Blaise instrument field specifications for the yogurt food category

Item Number	Variable Name	Item Text
YOG005	YogurtFroz	Was it frozen yogurt?
YOG010	YogurtTypeMilk	Was it made from whole, lowfat, nonfat milk, or something else?
YOG015	YogurtLoCalSwtnr	Was it made with low calorie sweetener?
YOG020	YogurtFlav	What flavor was it? (Was it plain, vanilla, fruit, or something else?)
YOG500		GO TO STANDARD ADDITiON QUESTIONS, THEN CONTINUE WITH YOG600.
YOG600	YogurtAmt	How much yogurt did {YOU/SP NAME} actually eat?

3. The autocoding pathways database

The large number of questions and responses in the AMPM Blaise instrument produces an even larger number of skip patterns because the questions asked about a food depends on the responses to the previous questions. The number of possible food paths through the food detail questions is roughly estimated to be over 400,000. Identifying the most common food paths from the hundreds of thousands of possible pathways was the first step in setting up an automated coding system. The initial implementation of autocoding began in 2002 with the identification of commonly reported, simple foods and the creation of food paths linking to these foods. Using the 2002 AMPM Blaise instrument, 28 food paths were created and linked to 10 basic foods (whole, 2%, 1% and nonfat milk; diet and regular cola soft drinks; white sugar; catsup; apple; and banana). This initial implementation achieved an automated food coding rate of 11% of reported foods in the 2002 What We Eat in America, the dietary component of the National Health and Nutrition Examination Survey (NHANES). To pursue a more substantial rate of autocoding, a greater number of paths had to be identified and linked to food codes.

3.1 Development of the autocoding pathways database

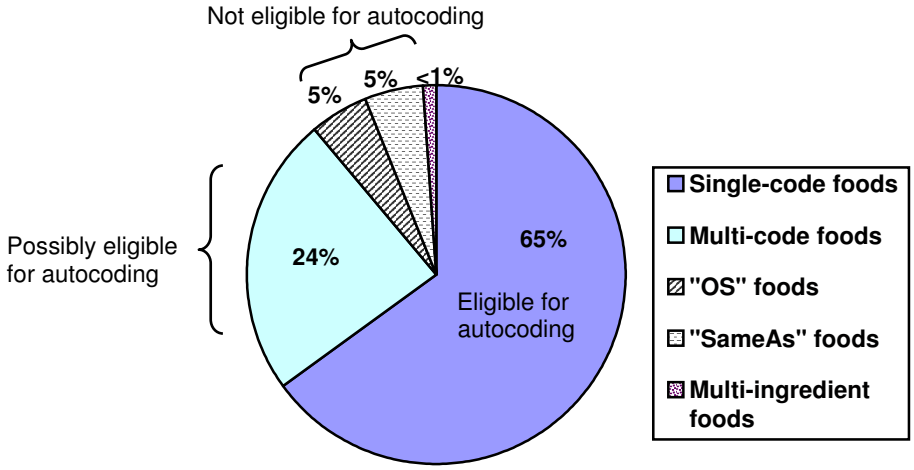
The development of the autocoding pathways database began with analysis of approximately 147,000 foods reported in the 2002 What We Eat in America survey and other food intake studies conducted by the Food Surveys Research Group. These data were collected using the AMPM Blaise instrument, coded by trained food coders, and thoroughly reviewed and monitored for quality control by nutritionists at USDA. A program was written in Visual Basic for Applications (Microsoft® Access 2000 (9.0.3821 SR-1)) that extracted food paths from the intake data and assigned a unique number to each food path within a food category. Extraneous information that does not contribute to the description of a food (e.g. the amount consumed) was not included in the food path identification process. Each combination of question and response that defines a food path was assigned a unique sequence number within that food path. Table 3 illustrates three food paths for the coffee category and the food codes that correspond to the food paths.

Table 3. Examples of food paths and food code links for the coffee food category

Food category		Sequence #	Question variable	Response value	Foodcode link
20010	1	1 2 3	CoffeeKind CoffeeCaffeine CoffeeForm	Coffee Regular Brewed	92101000 Coffee, made from ground, regular
20010	2	1 2 3	CoffeeKind CoffeeCaffeine CoffeeForm	Coffee Regular Instant	92103000 Coffee, made from powdered instant, regular
20010	3	1 2 3	CoffeeKind CoffeeCaffeine CoffeeForm	Coffee Regular Ready to drink	92100500 Coffee, regular, NS as to ground or instant

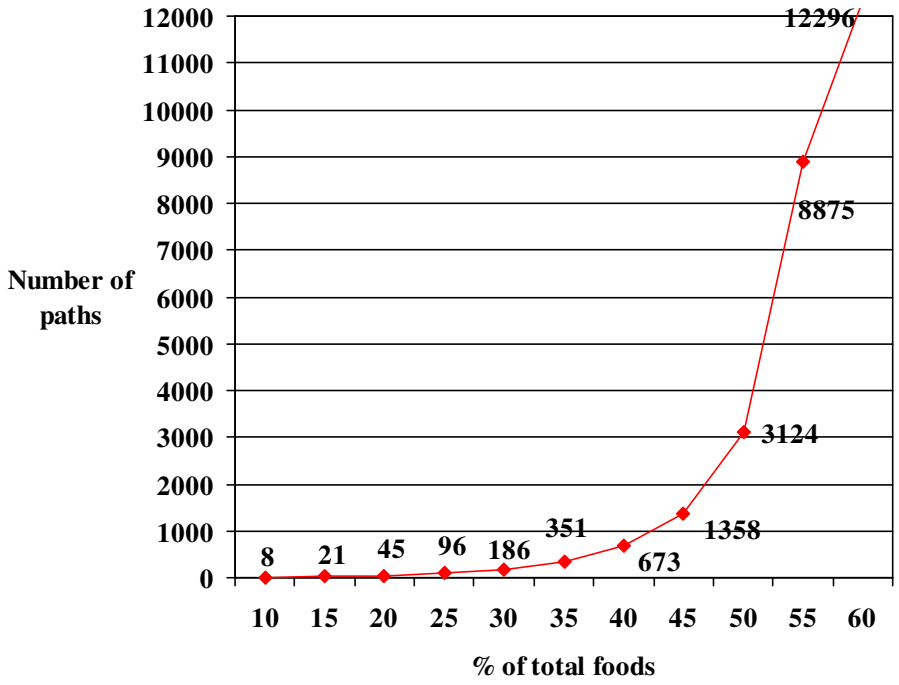
Certain foods were excluded from this initial construction of food paths. Complex foods that were coded with more than one food code were not considered for autocoding at this stage. These "multi-code" foods are items such as sandwiches or salads where individual ingredients (e.g. bread, meat, cheese, etc.) are coded as separate foods. Another category of foods that was not included during food path development was "Other Specify" or "OS" foods. During the AMPM interview, if a response is not listed as a choice in the Blaise instrument, then the interviewer may enter text in an open-ended "OS" field to record the exact response. "Same As" foods are foods that were previously reported by another member of a household. The AMPM Blaise instrument allows interviewers to record details such as "same as Mother's dinner, 7pm" in an open-ended "SameAsInstruction" field. A small percentage of foods have their main ingredients entered in an open-ended field during the AMPM interview. Homemade soups are an example of multi-ingredient foods which were deemed not eligible for autocoding at this initial stage. Accurate interpretation of these open-ended responses using a text matching method would be difficult. A breakdown of the approximately 147,000 foods analyzed during the autocoding developmental stage is shown in Figure 2.

Figure 2. Initial estimations of foods eligible for automated coding



From this initial analysis of 147,000 foods, we generated 12,304 unique food paths. Since the food intake data had already been coded, food codes that linked to the 12,304 food paths could be easily identified. We found that a relatively small number of food paths represented the majority of foods, as shown in Figure 3. Based on these data, we estimated that about 50% of total foods reported could be autocoded with approximately 3100 food paths.

Figure 3: Estimation of the number of food paths needed to achieve autocoding rate of total foods based on approximately 147,000 foods.



Before the food paths were incorporated into PIPS for autocoding, the data were reviewed by USDA nutritionists who verified that the food path was linked to the correct food code. Food paths were ranked by frequency to identify the paths that would capture the most autocoded foods. The food paths reported most often were reviewed first. Once a food path is approved as correct, it is added to the Autocode and FoodCodeLink tables used by PIPS for autocoding. The Autocode table contains fields for food category, path, sequence number, question variable and response value. The FoodCodeLink table links the food path to a food code. It contains fields for food category, path, and food code. These tables may be updated as necessary.

In addition to the review process, an adjudication test was performed before autocoding was implemented for the 2004 survey year. About 1500 dietary intakes were collected with the AMPM Blaise instrument and coded by trained food coders in 2002 and 2003 during a food intake study conducted by the Food Surveys Research Group. The Blaise intake data were re-processed using PIPS, which contained 1567 approved food path-food code links for autocoding. An autocoding rate of 50% of total foods reported was achieved for this dataset. The autocoded food items were then compared to the original food codes selected during manual coding. Out of the 12,224 autocoded foods, 98 food items (or less than 1% of all autocoded foods) did not match the original, manually-coded food code. Upon examination, some food path-food code links in the autocoding database were changed or removed. About half of the mismatched foods were due to additional interviewer comments that changed the outcome of the specificed food path. (The section titled "Limitations of

Autocoding", discusses these situations in greater detail.) This adjudication test verified that the autocoding process employed by PIPS produces accurate results in an efficient manner. The autocoding rate of 50% of total foods achieved with 1567 food path-food code links exceeded our estimates and expectations.

3.2 Implementation of the autocoding pathways database

Autocoding was put into operation for the 2004 What We Eat in America survey. In January 2004, we started with 1567 food paths in the autocoding database, which resulted in an autocoding rate of about 45% of reported foods. More food paths were added to the autocoding database during the course of the year. In June 2004, an autocoding rate of 59% was achieved. Currently, 2788 food paths are part of the autocoding database. In another food intake study conducted recently by the Food Surveys Research Group, we achieved an overall autocoding rate of 61% using the 2788 food paths in the current autocoding database. Again, these rates exceeded our estimations and expectations. Table 4 shows the 10 most frequently reported food paths since autocoding has been implemented.

Table 4: Top ten most frequently reported food paths being autocoded

Question Variable	Response Value
MilkKind	Whole milk
MilkKind	2% reduced fat milk
SugarKind	Granulated white sugar
CoffeeKind CoffeeCaffeine CoffeeForm	Coffee Regular Ground
FruitKind	Banana
SodaKind SodaCaffeineFree SodaType	Coke No (contains caffeine) Regular
FruitKind FruitType	Apple Fresh/raw
MilkKind	Skim/nonfat milk
ChipKind ChipType01	Potato chips (all flavors) Regular
MilkKind	1% lowfat milk

There is much agreement between the most frequently reported food paths and the most frequently reported food codes. However, there are some differences between their rankings. For example, lettuce and white bread are two of the most frequently reported food codes, but their food paths do not rank in the top ten. Because lettuce and bread are typically used as ingredients in sandwiches and salads, these foods are not autocoded very often with our current system. This is one of the limitations of our autocoding method that is discussed in the following section.

3.3 Limitations of the Autocoding Pathways Database

Although we have found that the AMPM Blaise instrument works successfully with the autocoding pathways database thus far, some limitations of this method exist. The autocoding process relies on text matching to link the food detail stored in the Blaise database to a food code. Changes to the AMPM Blaise instrument, such as added questions and responses and changes in the spelling of responses, may affect the autocoding food paths. Coordination between updating the AMPM Blaise instrument and maintaining the autocoding pathways database will be important. If a change occurs in the AMPM Blaise database, the autocoding pathways database will need to be reviewed to determine the effect, if any, on the food paths.

Open-ended responses are always a challenge to automated coding systems. The number of open-ended responses in the AMPM Blaise instrument was greatly reduced through the use of over 90 lookup tables. These lookup tables help to standardize the response options and facilitate the autocoding process. However, there are a few open-ended response variables ("Other Specify", "Same As" and multi-ingredient foods) that could not be handled with look up tables. These were mentioned previously in Section 3.1. The "Other Specify" or "OS" open-ended response is necessary when there is no response in the lookup table that corresponds to the answer given by the respondent. Text matching could be done on this type of open-ended response, but the variability of interviewers' entries would make pursuing this impractical with our method. We estimated that about 5% of total foods reported in a survey year contain an "OS" field (Figure 2). Frequently reported OS responses are added to the AMPM Blaise instrument. A similar situation exists for multi-ingredient foods where the ingredients in a mixed dish are listed in an open-ended response field. Only a small number of foods (approximately 0.4%) are handled in this way, so autocoding is not substantially affected. The "SameAsInstruction" open-ended field is an important feature of the AMPM Blaise instrument, allowing the interview to quickly record the same food consumed by more than one household member. Even though these foods cannot be autocoded with our method, this open-ended response saves the interviewer time and enhances the flow of the interview.

At any point in the interview, the interviewer may enter a text comment that gets stored in the Blaise database. Most often, these comments are entered to note time, eating occasion, or amount information volunteered by the respondent before the food details are collected. A small percentage of these comments contribute additional information to the food detail captured by the food path. Sometimes, these comments may contradict or change information collected in the food path. For example, a respondent reported consuming rye toast and the appropriate food path for rye toast was recorded. However, the interviewer entered a comment that said, "This was really wheat toast", which negates the food path recorded in the Blaise database. In this case, if there were a match in the autocoding pathways database for the rye toast, this food would have been inappropriately autocoded as rye toast. Fortunately, this type of situation does not happen very often. During the adjudication test described in Section 3.1, less than 1% of autocoded foods contained comments that would change the food path-food code link. The number of comments affecting food paths can be reduced further with additional interviewer training on when to enter comments versus when to change the response in the AMPM Blaise instrument.

Foods such as sandwiches and salads where individual components of the food are reported separately pose another challenge for the autocoding process. As it is currently designed, the Blaise database considers these multi-component foods as a single food. In order to capture the food paths that represent the individual ingredients, the multi-component food must be disaggregated into its component parts. For example, a tossed salad may disaggregate into iceberg lettuce, tomatoes, carrots, and croutons. Disaggregating these types of foods will increase the autocoding rate by an estimated 20-25%.

4. Conclusion

The creation of an autocoding pathways database and its implementation in PIPS for automated coding of food data has proven to be very successful. The success of this autocoding method is owed in large part to the design of the Blaise instrument and the ability to extract and reorganize data from the Blaise database. The amount of food data being captured with this automated method has exceeded our initial expectations. However, there is still room for improvement. In the future, we hope to explore other automated coding methods that would enhance the current system and contribute to our overall goal of high quality dietary data collection and processing.

Mention of commercial products in this article is solely for the purpose of providing specific information and does not imply recommendation or endorsement by the U.S. Department of Agriculture over others not mentioned.

5. References

Paul P. Biemer and Lars E. Lyberg, *Introduction to Survey Quality*. 2003, John Wiley & Sons, Inc.

Howard Speizer and Paul Buckley, "Automated Coding of Survey Data" in *Computer Assisted Survey Information Collection*, Edited by Mick P. Couper, Reginald P. Baker, Jelke Bethlehem, Cynthia Z.F. Clark, Jean Martin, William L. Nicholls II, and James M. O'Reilly. 1998, John Wiley & Sons, Inc.

Generating Blaise with Blaise

Gerrit de Bolster, Statistics Netherlands, July 26, 2004

1. Introduction

At Statistics Netherlands the first version of the IMPECT information system was implemented in 2001 to support the Business Surveys. This information system includes a questionnaire server linked with a variable server and a classification server. These servers, built with Visual Basic and SQL-server, are used by end-users to define their questionnaires interactively. Once finished, they can use the system to generate questionnaires from the same meta data for different media: paper (using Post Script), Blaise IS and Blaise to be used with the Electronic Data Reporter (EDR) as well as data entry machines for the processing of data from paper forms. The first version only supported the generation of paper questionnaires, the second version was extended to generate also Blaise IS and Blaise questionnaires. This version was limited yet to simple questionnaires like the Short Term Surveys on monthly and quarterly turnover. In this generation process Maniplus set-ups are used to convert the metadata, exported with Visual Basic programs from the SQL-server database as ASCII files, into the final product.

A Maniplus prototype has been developed to generate from this metadata larger Blaise questionnaires as used for the yearly Production Surveys. The questionnaires themselves are basically not complex but extensive. To deal with that the sections defined in the metadata are translated into parallels in the Blaise questionnaire. The questions are considered as rows in a table with a maximum of four answer columns following it. Besides that different symbols are supported for aggregation, subtraction and transport of results to other sections. So called 'table-questions' (e.g. value and quantity of sales per product) are featured too. The generation system supports all combinations of answer columns of different types and symbols by building a question row as a Block in a data model using pre-defined pieces of Blaise language in which parameters are substituted. It adjusts the height of the block on the screen depending on the number of text lines in combination with the size of symbols and answer fields. The explanations are generated as a Windows help file linked to the related questions and sections using the Blaise help facility. Rules are added to sum or subtract different sets of answers to a higher aggregation level, even throughout the different sections. As a special feature enterprise dependant fillings of 'table-questions' in the same type of questionnaire are supported without changing the structure of the data model so the data can be stored in the same data file (.bdb). This paper describes the generation process with all its ins and outs.

2. Questionnaire server

2.1 User-interface

The questionnaire server in the IMPECT system supports a hierarchical (tree) structure for business questionnaires. The metadata is stored in an SQL-server database. The end-users responsible for the maintenance of the questionnaire metadata are given access through a Visual Basic application. This application has a user-interface similar to Outlook. It contains 3 windows: one on the left-hand side of the screen and two on the right-hand side, one above the other. In the left-hand side window a tree-view of the hierarchical structure of the questionnaire is shown. The highest level consists of a list of all the questionnaires available. By clicking on this structure a part of it can be unfolded so detailed levels will be visible. Depending on the focus in the tree the two windows on the right-hand side show different information. In the top window auxiliary and linked information is shown, e.g. the list of possible question texts. By clicking on this information it can be used to define or alter

the questionnaire. In the bottom window detailed information about the focused item itself is shown.

2.2 Questionnaire structure

A 4-digit number uniquely identifies a questionnaire within the questionnaire server. This number is presented in the questionnaire preceded by the characters 'VL'. These characters are an abbreviation of the Dutch word for questionnaire: *vragenlijst* and are used to distinguish these questionnaires from those who are manually created by our repro-department. The latter ones start with the characters 'RZ', abbreviation of the Dutch word for repro-department. Besides its identification number a questionnaire has a title and is linked to a certain target group within the population of enterprises. A questionnaire is divided into so-called rubrics. Every rubric has a (capital) letter as identification and a header. A rubric is used to group questions related to one subject expressed in the header. They can be considered as the chapters in a questionnaire. On itself a rubric is divided into questions. The questions are identified by their position within the rubric. The most important element of a question is its question text. This text is taken from a separated table and can be re-used. A question can also be given a sequential number within the scope of the rubric. This sequential number preceded by the rubric identification character is used as a link to explanations. To every question 0 to 4 answer columns can be added. If no answer column is added the question text can be seen as a sub-header within a rubric followed by a number of real questions. On the rubric level each of this columns can be given a header. At every level (questionnaire, rubric, question) an explanation text can be added.

2.3 Variables

For internal identification every answer is linked to a variable. These variables are stored in a different database, the variable server. The variables are identified by their own identification code. Within the questionnaire server each variable corresponds with an answer column of a question. Of course, a variable can be used in different questionnaires. Disseminated questionnaires do not contain these variable identifiers. On the paper questionnaires there is no place for these 13 character long identifications. Besides, it has no use to send out information that is not useful for the respondents as it only increases the volume of the questionnaires. When a statement is received by our input systems the variable identifier is automatically added to the answers for further processing.

2.4 Classifications

A special type of questions is called "table-question". A table question consists of a set of rows related to a classification, e.g. the value of the purchases per product. Each row is connected to a code within the classification. A filled-in answer is thus identified by the variable identification (the column) in combination with a classification code (the row). The identification of the classification used is linked to the question in the questionnaire server. The text for each row is taken from the classification server. A question can consist of more than one column related to a classification, e.g. the quantity and value of the purchases per product. Within a classification a measurement indicator is available to be used in combination with a quantity. When applied, this indicator appears in a column preceding the answer column for the quantities. If the indicator is empty the answer box is not shown in the questionnaire. This is necessary because the quantities are only asked for a subset of the rows.

In the yearly Production Survey (PS), in the target group of the industry the classification for goods is very extensive (10.000 items). The list is too long to be presented to a respondent in an electronic questionnaire, it is even impossible to print it on a paper questionnaire. For this reason a function is added to select only a subset of the possible codes depending on the enterprise involved. As a

consequence individual different questionnaires (as far as it concerns table-questions) are supported by the IMPECT system.

3. Generation process

3.1 The metadata file

The metadata in the questionnaire server in combination with the variable server and the classification server are the source for the questionnaires. (This three-some is called the meta-server.) The metadata is exported from these databases by a query invoked by a Visual Basic program. Its output is an ASCII-file with an XML-like structure. This structure has explicitly been chosen to be flexible and extensible on one hand and to be read easily by a Manipula setup on the other hand. It is hierarchical as the metadata itself is hierarchical. The same file is used to generation the questionnaires for all the possible media: paper (PostScript), EDR (Blaise data model) and Blaise IS/form-based mode (Blaise data model converted finally to HTML). It is also used to generate the Blaise data-entry engines for the paper questionnaires (Blaise data model). It will also be used to generate the questionnaire for Blaise IS/page based mode (Blaise data model).

Every section in the file is starting with a tag between the characters "<" and ">", just like in XML. However, the ending tag is just a line with the characters "<>". The information lines are preceded by a two-character code, mainly two letters in uppercase (A-Z). Within a section several sub-sections can be present and within a sub-section several sub-sub-sections, thus representing the questionnaire structure. The lines are all starting in the first column to make it easier (and faster) for Manipula to read. Its general structure looks like this:

```
<questionnaire>                  *****************
AAquestionnaire information lines                 |
<rubric>                          *************   |
BBrubric information lines                  |     |
<question>                         ********   |     |
CCquestion information lines               |   |     |
<column>                            ***    |   |     |
DDcolumn information lines             |   |   |     |
<>                                  ***    |   |     |
<column>                            ***    |   |     |
....                                   |   |   |     |
<>                                  ***    |   |     |
<>                                 ********   |     |
<question>                         ********   |     |
....                                       |   |     |
<column>                            ***    |   |     |
....                                   |   |   |     |
<>                                  ***    |   |     |
<>                                 ********   |     |
<>                                *************   |
<rubric>                          *************   |
....                                        |     |
<question>                         ********   |     |
....                                       |   |     |
<column>                            ***    |   |     |
....                                   |   |   |     |
<>                                  ***    |   |     |
<>                                 ********   |     |
<>                                *************   |
<>                               *****************
```

3.2 Calculation rules

The metadata stored in the questionnaire server is not rich enough to generate an electronic version of the yearly Production Survey. As the first version of the questionnaire server was only intended to support paper questionnaires, no provision was built in to support calculations of answer values. Even in the second version this was not added, as it was not necessary for the questionnaires of the Short Term Surveys on monthly and quarterly turnover. This second version was extended to support these simple electronic questionnaires. There were not enough resources to upgrade the questionnaire server. It is also understandable that the questionnaire server could not be changed only for a prototype. Therefore a provisional solution was chosen creating some kind of supplement to the questionnaire server. This supplement, consisting of a Blaise database, contains the calculation formulas. After deriving the metadata file from the questionnaire server the calculation rules are added to it enriching the metadata without disturbing its structure. In this way it is very easy to add this supplementary information to the questionnaire server in the future without changing the process. Only the additional step, which adds the calculation rules to the metadata file, must be removed. These rules are based upon the variable identifiers. During the generation process they are substituted by the names of the actual fields in the Blaise data model.

3.3 The process steps

Every step in the generation process is implemented as a Maniplus setup. A main setup (GenEDR.msu) is used to invoke these steps as well as additional programs to parse the generated Blaise data model (B4CPARS.exe) and WinHelp file containing the explanations (HCW.exe). The main setup also takes care of copying the necessary auxiliary files.

The first setup (AddMeta.msu) is enriching the metadata file with the calculation rules as was described before. Additional it also replaces dotted lines in the question text (…. ….) by a text telling the respondent to add descriptions as a remark (Ctrl-M). These dotted lines are used in paper questionnaires to offer the respondent the opportunity to write down in free text the descriptions of goods not covered by the rows of the classification.

The next setup (GenVertMeta.msu) converts the metadata file into a series of 4 text files. The first text file contains a list of the blocks to be generated including their pattern. A block can represent a rubric-header, a question or a additional line containing e.g. symbols to be displayed. How blocks are constructed including the use of the pattern will be explained in the next chapter (From bricks to blocks). The second one contains the text lines to be used in the blocks. The text lines contain special characters for fonts and other layout features mostly the same as used in Blaise (rich text). The third file contains the calculation rules and the fourth the variable identifiers linked to the names of the Blaise fields in the data model.

The third setup (GenBla.msu) generates the actual data model using the 4 files created by the previous setup. A file containing templates of Blaise and Manipula source lines (GenBla.tfm) is read and, based upon the information in the 4 input files, the right set of lines are selected and the variable parts are replaced using text fills. Besides the data model a Maniplus setup is generated so the questionnaire can be used with the CBS data collection tool “Electronic Data Reporter”. The command line parser from Blaise (B4CPARS.exe) is used to parse the data model and the Maniplus setup.

A fourth setup (GenHlp.msu) uses the file containing the text lines to generate a help file with the explanations linked to the questionnaire, the rubrics and the questions. It can be used through the Blaise help mechanism for questionnaires. The setup creates a Windows help project file as well as an RTF-file according to the format of a Windows help file. To do so it selects the lines with the explanation from the text file in combination with a file containing templates with Project directives and RTF-lines (GenHlp.tfm). After generation of the output files the Windows help parser (HCW.exe) is invoked to create the final help file.

A last, small setup is used to create a XML-file that is used by the Electronic Data reporter. Finally the set of files are packed using the Windows cabinet program (CabArc.exe) to make the package file for Electronic Data reporter (.esf).

4. From bricks to blocks

4.1 The basic structure

The basic structure of the generated Blaise data model is a set of tables, one for each rubric, containing a block for every normal question or a block for each row in a table-question. Every table is defined as a parallel so the rubrics will appear as tabs. The main program appearing as the first tab contains a fixed set of questions related to contact information. This is standard within the IMPECT questionnaires. Each field in a table refers to a block. The fields are not created as an array as each block can be different. In stead, the fields are defined each with its own name: a letter with a 4-digit sequential number (e.g. B0006). The blocks they refer to are defined separately depending

on their pattern (see next paragraph). Questions with the same pattern are referring to the same type of block. The blocks themselves do not contain any text, this is past on to them through parameters. In the rules, where the fields are put on the route, the correct question text is added as well as the link to the Windows help file containing the explanations.

Example of fields and rules:

```
FIELDS
R0018 /"·":B_HHHHH1
B0019 /"·":B_TTTTT1
B0020 /"@bD.1*·":B_TTTBT1
B0021 /"@bD.2*·":B_TTTBT1

RULES
R0018('@bBedrijfsopbrengsten','·','·','@b@ebedragen x 1000','·','_rbr')
B0019('@bNetto omzet','·','·','·','·','_vrg')
B0020('Netto omzet uit de hoofdactiviteit van de onderneming','·','·','·','D.1')
B0021('Netto omzet uit overige activiteiten','·','·','·','D.2')
```

4.2 Block pattern

The pattern of a block (and thus of a question) is described as a set of 5 capital letters. Each letter represents a column in the table. The first one is a wide column in which the rubric header or the question text is placed. The other four are the answer columns in which an edit box or a text or a symbol can be placed. These symbols, like a line under a sum of fields, are made using normal characters and displayed as text. Each type of letter is linked to a special part of a block. The letter “T” e.g. causes that a part that shows a text is included, a letter “R” an edit box with a field of the type “real”. The 5 parts together form a complete block with a unique pattern: from bricks to blocks. The pattern serves as a part of its name as well.

Example of a block:

```
BLOCK B_TTTBB1
PARAMETERS
IMPORT ITekst0 : STRING
IMPORT ITekst1 : STRING
IMPORT ITekst2 : STRING
IMPORT IToelID   : STRING
AUXFIELDS
Tekst0 "^ITekst0" "^IToelID" : STRING[1]
Tekst1 "^ITekst1" "^IToelID" : STRING[1]
Tekst2 "^ITekst2" "^IToelID" : STRING[1]
FIELDS
Vraag3 "^Header @/@b* zie toelichting (Ctrl-F1)" "^IToelID" : -99999999..99999999
Vraag4 "^Header @/@b* zie toelichting (Ctrl-F1)" "^IToelID" : -99999999..99999999
RULES
Tekst0.SHOW
Tekst1.SHOW
Tekst2.SHOW
Vraag3.ASK
Vraag4.ASK
LAYOUT
AT Tekst0 FIELDPANE fptext1
```

```
AT Tekst1 FIELDPANE fpcolumn1
AT Tekst2 FIELDPANE fpcolumn1
AT Vraag3 FIELDPANE fpnumber
AT Vraag4 FIELDPANE fpnumber
ENDBLOCK
```

The number at the end of the pattern indicates the height of the layout of fields. The length of the question text determines this height. If it is too long it must be wrapped and the row on the screen must be higher to fit multiple lines. A “1” indicates that only one line is needed, a “2” two lines and so on. The different heights are defined in the standard modelib that is added to this application. This modelib contains a standard set of field layouts, one for each type of field. The calculation of the height needed must still be improved. It depends e.g. on the font size and attributes like “bold”. To complicate the case more: two lines as defined in the modelib can often fit 3 text lines. To conclude: a block is defined by a combination of its pattern and the height of the text lines to be displayed. Generating a block based upon its parts makes it possible to support all type of patterns.

Some times a question can lead to 2 blocks. In the questionnaire server it is possible to define a question column containing an edit box with a line beneath it, to symbolize the aggregation of fields. On a paper questionnaire it can be printed in the same space as any other question. On the screen this is not possible. Blaise does not allow putting an edit box together with text in one table cell. As a consequence in an electronic questionnaire two rows are needed: one for the edit box and one for the aggregation symbol.

5. One size fits all

5.1 Different look, one database

As was mentioned before the system has to support individual different questionnaires as well. These questionnaires are individual in the sense that in one or more table-questions the sub-set of rows is dependant on the enterprise where it is used. A simple solution would be to generate tailor-made data models. However, the IMPECT system demands that questionnaires with the same identification have the same data model. Not only because of the internal processing but also because of the use with the Electronic Data Reporter. Giving them all different identifications would create a chaos. Besides, it is quiet normal that an accountancy office is doing the statistical reporting for several clients. In that case the accountancy office can load the different questionnaires from the clients into its installation of the Electronic Data Reporter. If it is the same questionnaire it is stored once. The different clients are distinguished by their identification as part of the primary key of the data model. Furthermore, it will not be understood that questionnaires would be considered as different because some rows in a table are different. So the problem was to generate different looking questionnaires that were using the same basic data model and thus the same data base (.bdb) to store the data.

5.2 Maximum number of rows

In the questionnaire server in the case of these individual table-questions already a field was available containing a maximum number of rows. This was introduced because the data entry instruments faced the same situation: one instrument per questionnaire. For the instrument it was used to generate a table with "open" rows. "Open" means that the code of the row had to be typed in. As they were printed on the paper questionnaire this was possible. In CASI questionnaires as used in the Electronic Data Reporter such solution is out of the question. The questionnaire should contain a predefined set of rows according to the situation of the responding enterprise. The respondent only has to fill-in the requested values at the right row. To achieve this the maximum number of rows is used to generate a fixed set of blocks for the table-question involved. As the question defines its pattern they all refer to the same block type. The only difference could occur in the height of the text lines. In Blaise a same pattern of fields always produces the same database (.bdb). In other words, doing so individual different questionnaires can use the same database to store the data. The difference in text and number of displayed questions is stored in the rules and thus in the data model (.bmi) and the difference in layout (height) is stored in the layout file (.bdm). By activating the right data model and layout file as another enterprise is selected the problem was solved: *one size fits all*. This has been implemented by replacing the "current" data model and layout file in the working folder by the ones from the respondent stored in another folder. This replacement functionality is included in the generated Maniplus setup that invokes the form using the "edit" statement after the respondent has selected the keys through a dialog. For the use with normal, not individual, questionnaires a setup is generated without this extra functionality. Using the new Blaise feature of directives that are interpreted during preparation time creates the different appearances of the setup. Only by filling in the right directive {$DEFINE …} the matching setup is automatically selected.

6. Concluding remarks

Generating Blaise with Blaise is not difficult. The capabilities of Maniplus to manipulate text files in combination with the structure of Blaise sources offer a wide range of possibilities.

Concerning the process described in this paper: at this moment a pilot is prepared to test the questionnaires generated with the process described in this paper. It will used with Blaise IS, the page based mode. If successful, it will be the first step to on-line interviewing at Statistics Netherlands.

CodeBuilder2 – An Initial Coding Tool

Michael D. A. Egan, Operations Research and Development Division (ORDD),
Statistics Canada

1. Introduction

In February 2003, an informal post-development review of a survey determined that authors had spent too much time working on the static text during development. Other authors, working on other surveys, agreed. Looking at the specifications and the code, it struck us that a programmatic approach was available.

As a result, we created a set of Manipula scripts that we called *CodeBuilder*: it quickly became a favourite tool of developers. We added some functionality in the summer of 2004, resulting in *CodeBuilder2*. The name *CodeBuilder* will be used throughout this paper, for the sake of simplicity.

2. Genesis

The creation of an application to collect data for a survey is a multi-divisional project at Statistics Canada. Over the last several years, there have been many reviews of how these applications are created (in varying levels of formality). We are always looking for better ways to do things.

"Authoring" (the act of creating Blaise code from a specification) is an intra-divisional task, so improving this aspect of development is an internal initiative. When we look for an internal solution, we want to develop something that: has no impact on any of our clients; uses our existing software mix, and; can be created "in our free time."

"Specifying" is an extra-divisional task. Different client divisions have different histories of specifying, and have different levels of sophistication. Some clients have a metadata system that produces specification documents and other outputs; others use Microsoft Word. Some clients produce one document per block; most produce two documents per block (one per official language). While ORDD has no input on <u>how</u> clients create their specifications, we have documented a standard that defines the structure of specifications.

To summarize, we had an internal initiative to reduce the amount of time an author spent on static text during initial authoring. Our solution could have no impact on how clients create specifications, and so would actually have to handle any style of specifications.

2.1 The Burden of Static Text

Just how burdensome is static text? How can something which doesn't change be a drag on development? To answer these questions, one needs to know how text gets from the specifications to the code.

Normally, each block is specified through two documents: one for English, one for French. (Things do not go smoother when the client specifies both English and French in one document.) In the Blaise code, English and French mingles and weaves. Each field has English and French definitions; each answer item in a TYPE has English and French text. The following table is a mock-up of two specification documents (English, then French) and the associated code.

Q1	**QTextE11** **QTextE12** 0-90 Q2 **QTextE21** ITextE21 <1> ATextE11 <2> ATextE12 <3> ATextE13 <4> ATextE14	Q1	**QTextF11** **QTextF12** 0-90 Q2 **QTextF21** ITextF21 <1> ATextF11 <2> ATextF12 <3> ATextF13 <4> ATextF14	Q1 "Q1@|@|^piSubjectName @/@/@BQTextE11a QTextE11b@B @/@/@BQTextE12@B" "Q1@|@|^piSubjectName @/@/@BQTextF11a QTextF11b@B @/@/@BQTextF12@B" : 0..90 Q2 "Q2@|@|^piSubjectName @/@/@BQTextE21@B @/@/@SITextE21@S" "Q2@|@|^piSubjectName @/@/@BQTextF21@B @/@/@SITextF21@S" : (V01 (1) "@SATextE11@S" "@SATextF11@S", V02 (2) "@SATextE12@S" "@SATextF12@S", V03 (3) "@SATextE13@S" "@SATextF13@S", V04 (4) "@SATextE14@S" "@SATextF14@S")

(SOD standards require that the subject's name be displayed at the top of the screen on every question. This does not need to be specified, and is handled by "@|@|^piSubjectName". QTextE11a and QTextE11b show what would happen if the specified paragraph was long. ORDD coding standards place a limit of 104 bytes on a line of code. We would never put the type definition in the FIELDS section: they would be placed in the TYPE section. It is in the FIELDS section here so that you may have an underrstanding of the work required to manipulate text strings for enumerated types.)

To get the text for one field, an author would:

- Insert the field-id and header information in the code in the Control Centre;
- switch to the English specification;
- find, highlight and copy the English text;
- switch to the Control Centre;
- paste the English text into the code;
- add font-controls (@B, @S, @U) and align the code;
- switch to the French specification;
- find, highlight and copy the French text;
- switch back to the Control Centre;
- paste the French text into the code;
- add font-controls (@B, @S, @U) and align the code;
- insert the field definition (0..90 or tYesNo, for example).

This takes four or five minutes per field for normal question text. If there were 25 fields in the block, you're looking at two hours of copying, pasting and formatting. In a smallish questionnaire, where there are 500 questions, you're looking at one solid week of copying, pasting and formatting.

We did not want to spend that amount of time on static text. We wanted a program that would handle the default field definitions (field-id, standard header), insert the English and French text, handle font controls and format the code properly.

2.2 The Benefit of Standards

It was fairly easy to write the code that creates the FIELDS statements, and that is because of our coding standards and SOD presentation standards. ORDD standards require certain parts of a field definition to be in certain areas, and that is the sort of thing a computer program can do extremely well; repeatedly; without getting bored.

SOD presentation standards require that text which the interviewer must speak aloud be displayed in bold-black, while optional text and interviewer instructions are to be displayed in bold-sapphire. There are a few other colours that are used on rare instances. *CodeBuilder* assumes that any text should be bold-black. If a particular paragraph has a different requirement, the author just has to insert the appropriate font control in the *CodeBuilder* input file.

It was not possible to write code that could detect question text within the specification. How would the code differentiate between question text and field range descriptions, or programmer notes, or answer categories? That's something that the author can do with complete accurately. The way an author organizes the input files tells *CodeBuilder* what it is dealing with (field-id, question text, edit text, etc.).

2.3 Reduction, Not Elimination

Unfortunately, *CodeBuilder* does not reduce the amount of time an author spends on static text by 100%. It reduces it by about 80%. It takes a little time to organize the specifications, about one minute per field.

When using *CodeBuilder*, the author works on text at the block level, not at the field level, and not at the answer/item level. The organization work is largely deleting text that isn't needed, which can be done very quickly. What would have taken two hours can now be done in about 20 minutes.

A completely programmatic solution would require specifications that clearly identify their various component parts (English text, French text, answer text, flow instructions, etc.). This level of functionality describes a metadata system, not a couple of Manipula scripts.

3. When *CodeBuilder* is Used

CodeBuilder is an **initial** coding aid: it is used one time, when we do initial coding.

CodeBuilder is not an **iterative** coding aid: it cannot be used to make updates to a few text items in a few places. After initial coding is done, we will continue to update specific text items the way we always have done: testers will record a text problem; specifications will change, if necessary; authors will update the specific word or phrase in the code.

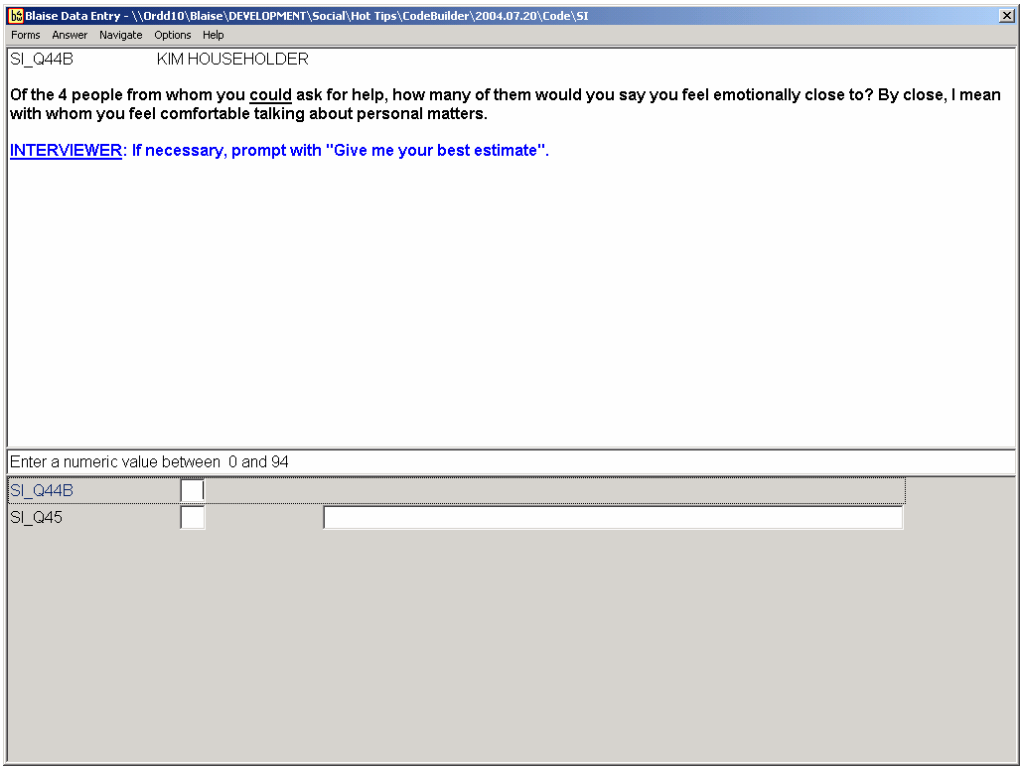
While "static text" is not supposed to change, it does. Despite all of the pre-coding reviews, mistakes will work their way into the code. There will be situations where testing will show that the specified text doesn't address all circumstances. Clients and sponsors will change their minds about the text they want. It is even possible that an author will make a mistake while using *CodeBuilder*! It is important to note that, while static text does change, it doesn't change a lot, and it is far faster to fix specific problems than it is to re-run *CodeBuilder*.

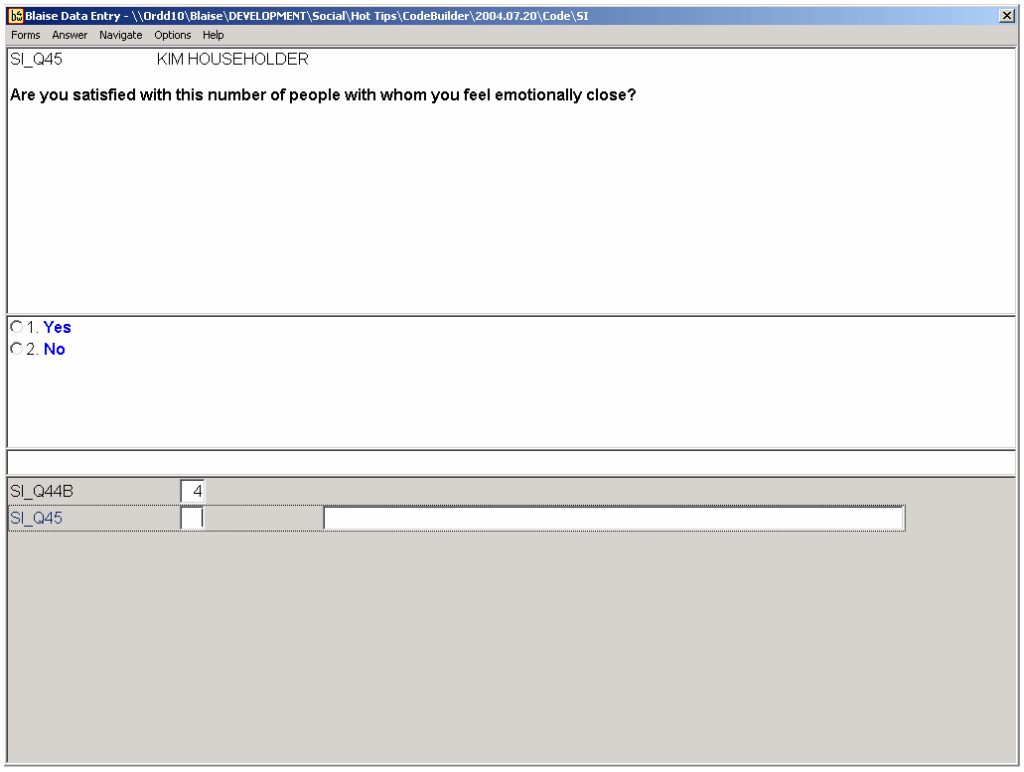
4. How *CodeBuilder* is Used

To better show how *CodeBuilder* works, some samples are included. First, we'll look at what the interviewer sees on a screen; then the specifications; then the *CodeBuilder* input files, and finally, the *CodeBuilder* output files.

4.1 Interviewing Screens

Here are two questions from one of our surveys, in English.





Here are the same two questions, in French.

Blaise Data Entry - \\Ordd10\Blaise\DEVELOPMENT\Social\Hot Tips\CodeBuilder\2004.07.20\Code\SI

Forms Answer Navigate Options Help

SI_Q44B KIM HOUSEHOLDER

Auprès de combien des 4 personnes à qui vous pourriez demander de l'aide vous sentez-vous proche émotionnellement? Par proche, j'entends avec qui vous seriez confortable de discuter de sujets personnels.

INTERVIEWEUR : Au besoin, demander au participant de donner sa meilleure estimation.

Enter a numeric value between 0 and 94

SI_Q44B 4

SI_Q45

Blaise Data Entry - \\Ordd10\Blaise\DEVELOPMENT\Social\Hot Tips\CodeBuilder\2004.07.20\Code\SI

Forms Answer Navigate Options Help

SI_Q45 KIM HOUSEHOLDER

Êtes-vous satisfait(e) du nombre de personnes avec qui vous vous sentez proche émotionnellement?

1. **Oui**
2. **Non**

SI_Q44B 4

SI_Q45

4.2 Specifications

This is the English specification for those two questions.

SI_Q44B **Of the ^Numhelp people from whom you could ask for help, how many of them would you say you feel emotionally close to? By close, I mean with whom you feel comfortable talking about personal matters.**
INTERVIEWER: If necessary, prompt with "Give me your best estimate".

[Min: 0 Max: 94]

<98> Refused ……go to SI_Q46
<99> Don't know……go to SI_Q46

SI_Q45 **Are you satisfied with this number of people with whom you feel emotionally close?**

<1> Yes
<2> No
<8> Refused
<9> Don't know

As previously mentioned, SOD standards require interviewers to speak aloud text that is displayed in bold-black. ORDD specification standards say that text which is to be displayed in bold-black must be specified in bold. Text that is specified in regular font (such as the interviewer instruction) will be displayed in bold-sapphire font.

Next is the French specification for those two questions.

SI_Q44B **Auprès de combien des ^Numhelp personnes à qui vous pourriez demander de l'aide vous sentez-vous proche émotionnellement? Par proche, j'entends avec qui vous seriez confortable de discuter de sujets personnels.**
INTERVIEWEUR: Au besoin, demander au participant de donner sa meilleure estimation.

[Min: 0 Max: 94]

<98> Refus ……passez à SI_Q46
<99> Ne sait pas ……..passez à SI_Q46

SI_Q45 **Êtes-vous satisfait(e) du nombre de personnes avec qui vous vous sentez proche émotionnellement?**

<1> Oui
<2> Non
<8> Refus
<9> Ne sait pas

4.3 *CodeBuilder* FIELDS Input

In the boxes below, bold-red text represents text that the author has to add or modify during the preparation of the input files. Everything else has been copied directly from the specification.

The text for the first paragraph of SI_Q44B has been truncated because it was too long to fit on one line. In the actual input file, all of the text for it is on one line. *CodeBuilder* recognizes each new line in the input file as a new paragraph in the question text. A blank line is used as a field delimiter. The type definitions are recorded in the French input file because our code places the field definition after the French text.

```
SI_Q44B
Of the ^NumHelp people from whom you @Ucould@U ask for help, how many of them would you say you feel…
@S@UINTERVIEWER@U: If necessary, prompt with ""Give me your best estimate"".

SI_Q45
Are you satisfied with this number of people with whom you feel emotionally close?
```

```
SI_Q44B
Auprès de combien des ^NumHelp personnes à qui vous @Upourriez@U demander de l'aide vous …
@S@UINTERVIEWEUR@U : Au besoin, demander au participant de donner sa meilleure estimation.
:   0..94

SI_Q45
Êtes-vous satisfait(e) du nombre de personnes avec qui vous vous sentez proche émotionnellement?
:   tYesNo
```

4.4 Sample – FIELDS Output

```
SI_Q44B         "SI_Q44B@|@|^piRespName
                @/@/@BOf the ^Numhelp people from whom you @Ucould@U ask for help, how many of them
                would you say you feel emotionally close to? By close, I mean with whom you feel
                comfortable talking about personal matters.@B
                @/@/@S@UINTERVIEWER@U: If necessary, prompt with ""Give me your best estimate"".@S"

                "SI_Q44B@|@|^piRespName
                @/@/@BAuprès de combien des ^Numhelp personnes à qui vous @Upourriez@U demander de
                l'aide vous sentez-vous proche émotionnellement? Par proche, j'entends avec qui
                vous seriez confortable de discuter de sujets personnels.@B
                @/@/@S@UINTERVIEWEUR@U : Au besoin, demander au participant de donner sa meilleure
                estimation.@S"

                :   0..94

SI_Q45          "SI_Q45@|@|^piRespName
                @/@/@BAre you satisfied with this number of people with whom you feel emotionally
                close?@B"

                "SI_Q45@|@|^piRespName
                @/@/@BÊtes-vous satisfait(e) du nombre de personnes avec qui vous vous sentez
                proche émotionnellement?@B"

                :   tYesNo
```

4.5 *CodeBuilder* TYPE Input

I'm introducing a new sample field at this point (row 1 below), with a longish answer list, to demonstrate how *CodeBuilder* handles TYPE text. Again, bold-red text in the input files indicate text that the author has to type during the formatting of the file; everything else was brought in during the paste from the specification.

SI_Q03 **What kind of information did you receive?**
INTERVIEWER: Mark all that apply.

<01> How to find housing
<02> How to look for a job
<03> How to access medical care services
<04> How to get language training
<05> How to find education or training
<06> How to get foreign education credentials or work experience assessed
<07> How to receive support for basic needs and services (e.g. food, clothes)
<08> How to obtain citizenship or a permanent resident card
<09> How to sponsor other immigrants
<10> How to contact immigration agencies for other business
<11> How to access other government services (excluding immigration issues)
<12> How to obtain financial and insurance products
<13> Other – Specify…….. go to SI_S03
<98> Refused ……..go to SI_Q06
<99> Don't know …….go to SI_Q06
Default Next Question: SI_Q04

@S tKindInfo
<01> How to find housing
<02> How to look for a job
<03> How to access medical care services
<04> How to get language training
<05> How to find education or training
<06> How to get foreign education credentials or work experience assessed
<07> How to receive support for basic needs and services (e.g. food, clothes)
<08> How to obtain citizenship or a permanent resident card
<09> How to sponsor other immigrants
<10> How to contact immigration agencies for other business
<11> How to access other government services (excluding immigration issues)
<12> How to obtain financial and insurance products
<13> Other – Specify

<01> Comment trouver un logement
<02> Comment chercher un emploi
<03> Comment avoir accès aux services de soins de santé
<04> Comment obtenir de la formation linguistique
<05> Comment poursuivre mon éducation ou ma formation
<06> Comment faire évaluer des attestations d'études ou de l'expérience professionnelle obtenue à l'étranger
<07> Comment obtenir du soutien pour pourvoir aux besoins fondamentaux (p. ex., aliments, vêtements)
<08> Comment obtenir la citoyenneté ou une carte de résident permanent
<09> Comment parrainer d'autres immigrants
<10> Comment entrer en communication avec les organismes d'immigration pour d'autres sujets
<11> Comment avoir accès à d'autres services gouvernementaux (excluant les questions reliées à l'immigration)
<12> Comment obtenir des produits financiers et d'assurance
<13> Autre – Précisez

4.6 Sample – TYPE Output

```
TYPE
    tKindInfo = (
        V01 (1)     "@SHow to find housing"
                    "@SComment trouver un logement",
        V02 (2)     "@SHow to look for a job"
                    "@SComment chercher un emploi",
        V03 (3)     "@SHow to access medical care services"
                    "@SComment avoir accès aux services de soins de santé",
        V04 (4)     "@SHow to get language training"
                    "@SComment obtenir de la formation linguistique",
        V05 (5)     "@SHow to find education or training"
                    "@SComment poursuivre mon éducation ou ma formation",
        V06 (6)     "@SHow to get foreign education credentials or work experience assessed"
                    "@SComment faire évaluer des attestations d'études ou de l'expérience
                    professionnelle obtenue à l'étranger",
        V07 (7)     "@SHow to receive support for basic needs and services (e.g. food, clothes)"
                    "@SComment obtenir du soutien pour pourvoir aux besoins fondamentaux (p. ex.,
                    aliments, vêtements)",
        V08 (8)     "@SHow to obtain citizenship or a permanent resident card"
                    "@SComment obtenir la citoyenneté ou une carte de résident permanent",
        V09 (9)     "@SHow to sponsor other immigrants"
                    "@SComment parrainer d'autres immigrants",
        V10 (10)    "@SHow to contact immigration agencies for other business"
                    "@SComment entrer en communication avec les organismes d'immigration pour d'autres
                    sujets",
        V11 (11)    "@SHow to access other government services (excluding immigration issues)"
                    "@SComment avoir accès à d'autres services gouvernementaux (excluant les questions
                    reliées à l'immigration)",
        V12 (12)    "@SHow to obtain financial and insurance products"
                    "@SComment obtenir des produits financiers et d'assurance",
        V13 (13)    "@SOther - Specify"
                    "@SAutre - Précisez" )
```

4.7 Sample – RULES Output

Lastly, here is an example of the RULES output that *CodeBuilder* generates. All of the fields are listed in the specified order, and edits are inserted as they are encountered.

```
SI_Q42

                {
                CHECK
                ( edit logic goes here )
                "@H@R@BSI_E42
                @/@/Please return to SI_Q36 and correct.@H@R@B"
                "@H@R@BSI_E42
                @/@/Veuillez retourner à SI_Q36 et corriger.@H@R@B"
                CHECK
                }

SI_Q43
SI_Q44A
SI_Q44B

                {
                CHECK
                ( edit logic goes here )
                "@H@R@BSI_E44B
                @/@/This number cannot exceed the number of people from whom the respondent
                could ask for help. Please return and correct.@H@R@B"
                "@H@R@BSI_E44B
                @/@/Ce nombre ne peut être supérieur au nombre de personnes sur lequel le
                répondant peut compter pour de l'aide. Veuillez retourner et corriger.@H@R@B"
                CHECK
                }

SI_Q45
SI_Q46
SI_R47
```

5. Conclusion

The *CodeBuilder* Manipula scripts described in this paper have led to a significant reduction in the amount of time authors spend on static text. *CodeBuilder* has removed a time-consuming, repetitive and boring development practice, allowing the authors to spend a larger portion of time on their 'real' task: getting the flows and edits to work properly!

CodeBuilder source code and documentation are available to any agency.

Generic System Interface Instrument

Curtis Ziesing, U.S. Census Bureau, Author/Presenter

1. Background

When the U.S. Census Bureau's legacy Computer-Assisted Interviewing (CAI) case management systems were designed back in the early-to-mid 1990's, one of the most problematic areas was the interface between the systems and the instruments. For our early surveys, system developers programmed their systems concentrating on the desired functionality for the end-users. At the same time the survey instrument authors programmed their questionnaires, concentrating primarily on the subject matter part of the instrument. The two pieces didn't come together until late in the design process and interface issues had to be resolved at the last minute. This caused problems in trying to get surveys fielded on time. Because there was no instrument available for system programmers to test while their system was being developed, interface issues could only be addressed once a "working" instrument was made available to them. (A "working" instrument is defined as one that includes the system interface pieces.)

As more surveys were converted to automated questionnaire data collection, it became more and more apparent that there was a desperate need for standards. These standards did evolve over time

These included:

- Standard Record Typed Input files for loading control and previously collected dependent data for the sample cases.
- Standard Interface fields to share data between the instrument and case management systems
- Standard File naming conventions so the systems would know how to process the file based on its file name.

One thing that was still lacking was a way to test the instrument interface with any newly developed systems until an automated instrument was available. As the Census Bureau was in the process of creating the next generation of systems to manage CAI collections at the same time it was creating its first Blaise instruments, it was decided to build a skeleton (or bare bones) instrument that would address the interface issues and allow system developers to test their systems early in the process so that the majority of the interface issues would have been resolved by the time the actual instrument was ready.

Some of the system changes that were occurring were:

- Converting from the DOS to Windows 2000 environment in CAPI
- Converting CATI from DOS to Browser-based with a central database repository for all telephone centers
- Converting survey instrument software from CASES to Blaise

- Storing case-level data as a BLOB field in databases for case transfer and access

- Running remote shell (rsh) from UNIX to NT platform for processing Blaise data

This "skeleton" instrument was designed to address these changes and allow system developers to test their systems while in the early stages of development. As new functionality has been added to the systems, the skeleton instrument has been used to prototype these changes. This paper will describe how the skeleton instrument was designed and constructed and the functionality it addresses.

2. Overview

The skeleton instrument was developed to provide the Census Bureau's CATI and CAPI case management systems with a means to test their interfaces with an instrument before the actual instrument is ready for testing. A computer assisted telephone interview (CATI) requires a different set of tests to be performed than a computer assisted personal interview (CAPI).

The instrument itself is quite simple. It is a shell, or one might say "a skeleton," that collects data only for the variables that the case management systems care about, since its purpose is to test the interfaces with the CAI control systems. The more complex piece is the process that was developed to automatically create the instrument, its associated scripts, and an input file (or preload data) based on the requirements of the particular survey instrument. There are six basic steps to this process:

1. Run a Blaise instrument that prompts the creator of the skeleton instrument for test-specific requirements. The instrument gets the name, survey period, and record types specific to the survey.

2. Generate the skeleton instrument appropriately based on the answers to the prompts. All the record types and CAPI or CATI code are added to a standard source file called **Inst.bla** and then processed into the skeleton instrument to be used for the specific testing purposes.

3. Generate an input file appropriately based on the answers to the prompts. The input file, if needed, may have four people in a single household or it may contain twenty people in several families. Based on the number of records requested these input files will be duplicated until enough records are created for the test.

4. Generate Manipula scripts for the interfaces with the case management systems. Because there are CAPI or CATI surveys, the Manipula scripts are different to handle the different interfaces with the control systems. Each of the Manipula scripts begins with a known set of standard lines of code, and then some specific lines of code are added to create scripts specific for the testing being done.

5. Create Blaise databases using the input file. This Blaise database will contain enough blocks, fields, and questions to allow for testing the control system interfaces.

6. Package the instrument for delivery to the appropriate case management system.

3. Collect Information

The process begins with the user running the **"getinfo"** Manipula script, which in turn runs a Blaise questionnaire named **"getsurveyinfo"** that collects answers to a series of questions to determine how the skeleton instrument and associated files are to be created, what the testing environment will be (CATI or CAPI), and how the input file is to be created. Although the same types of scripts are used for CATI or CAPI, the structure of each script is considerably different. The skeleton instrument will create the scripts needed based on the answers to the questions.

3.1 Instrument Name and Interview Period

First the study name (which is unique for each survey collection period) is captured. Figure 1 shows a screen where the name of the instrument is collected.

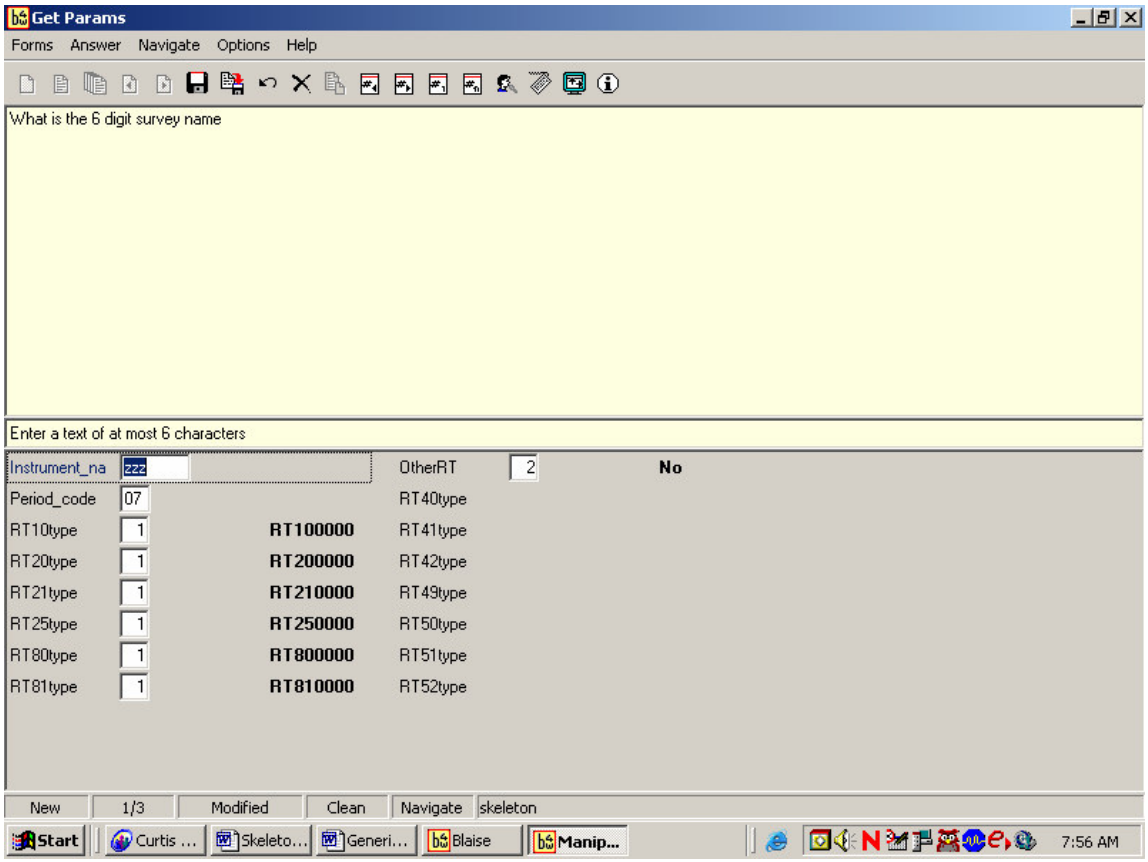


Figure 1. Collecting the name and record types to use

The case management systems use this information to create the appropriate directory structure when this instrument is installed. The skeleton instrument process also uses it to create the appropriate directory structure and to name the packaged file for delivery to the designated system. The answers to these questions will determine the files structure of the specific survey instrument. In Census Bureau systems, the study name consists of a 3-character survey acronym, a 1-character field that identifies whether it a production or re-interview survey, a 2-character panel code; and a 2-character interview period code. For example, a study name like acs_ag09 would indicate that this is for the ACS (American Community Survey), the "_" indicates it is an

original data collection instrument for production, the "ag" is a unique panel code assigned to the survey (may indicate the calendar year of the survey), and the 09 would indicate the month of the survey. The Census Bureau convention also refers to the last 4 characters of the survey name, which is made up of the panel code and the survey period, as the survey_id and it is used as an integral part of the file naming convention for the various files used by the systems.

3.2 Record Types

The next step is to identify the various record types that are required for the survey's input file. The Census Bureau uses a standardized set of record layouts for the various types of data that are typically included in an instrument as preload data. Each unique record layout is assigned a "record type," and the input file for a particular instrument is made up of a set of these record types for each case. The record types fall into three categories:

1. Those that contain data used only by the case management systems.

2. Those that contain data used only by the instrument.

3. Those that contain data used by both the case management systems and the instrument.

The skeleton instrument process is only concerned with record types that contain instrument data (categories 2 and 3).

Besides defining what record types are needed on the input file, this step is also defining some of the blocks that are needed in the skeleton instrument. Because the Census Bureau has pre-defined record layouts for input, we also have standard record type "include" files that define the standard variables on those record types. For each record type used by an instrument, that instrument will have a corresponding block of data. For example, if the record type for the address is 200100, the instrument will include an Rt200100.inc file, which will define the variables in a block named BRt20. By prompting for the record types in the getinfo script, the skeleton instrument process is then able to know which standard include files to put into the instrument. An example of a standard record type is provided in figure 2.

```
EMBEDDED BLOCK BRt20{00}
{SCIF Record Type 2000}
{
  Name:          RT2000.Q
  Record Type 20  Subtype 00  DEMOGRAPHIC ADDRESS DATA - Type 1}
    FIELDS
        {
        RecType        :  STRING[2]      { = 20 }
        RecSubType     :  STRING[2]      { = 00 }
}
        ADDR1          :  STRING[54]     {  [inputloc  2/5]     [# Address Field 1                }
        ADDR2          :  STRING[54]     {  [inputloc  2/15]    [# Address Field 2          }
        PO             :  STRING[22]     {  [inputloc  2/170]   [# Locality                   }
        ST             :  STRING[2]      {  [inputloc  2/192]   [# State Abbreviation          }
        ZIP5           :  STRING[5]      {  [inputloc  2/194]   [# Zip code                    }
        ZIP4           :  STRING[4]      {  [inputloc  2/199]   [# Zip 4                       }
 ENDBLOCK {BRt2000}

   FIELDS  {Instrument  Level}
        {ADDR1          :  STRING[54]     {  [inputloc  2/5]     [# Address Field 1                }
        ADDR2          :  STRING[54]     {  [inputloc  2/15]    [# Address Field 2          }}
        PO             :  STRING[22]     {  [inputloc  2/170]   [# Locality                   }
        ST             :  STRING[2]      {  [inputloc  2/192]   [# State Abbreviation          }
        ZIP5           :  STRING[5]      {  [inputloc  2/194]   [# Zip code                    }
        ZIP4           :  STRING[4]      {  [inputloc  2/199]   [# Zip 4                       }
```

Figure 2. Typical record type block

It shows the definition of a block with the group of fields defined, and then after the block is defined the fields are repeated. This process is to make sure any changes to the data are done outside of the BRT20 block definition. The data stored in this block will remain unchanged throughout the interview process.

3.3 Environment

The next set of questions helps establish the environment in which the instrument will be run. The Blaise version is collected because Census Bureau case management systems allow the use of different versions of Blaise for different instruments.

The next question asks whether the instrument is for production, systems testing, or training. The Census Bureau has included a field on one of the input file record types for this information, and it can be used in various ways by the instruments and the case management systems. Instruments have used this information to follow a path in a training case that differs from the path in a production case. For example, for training cases you might want a reference period to remain static while it is dynamic for a production case. This can be controlled through the use of the "survey type" field.

Next, the paths to Blaise and WinZip on the user's computer are captured for later use. This is done so that any user on the network can create a skeleton instrument. The Census Bureau generally sets up a users system with Blaise and WinZip in the same place but this can be adjusted for individual uses. In a Windows environment the default Blaise can be one version and the skeleton instrument can be created in a completely different version. The Census Bureau also uses different versions of WinZip to accomplish different tasks. One version is used to run command line parameter execution and another is used to run batch processes. The reason for establishing environment parameters is to allow flexibility in the use of different versions of Blaise running in different graphical user interfaces with different directory structures.

3.4 Input File

The questionnaire then collects more information about the input file that is to be created. An input file contains a set of sample data for each case. For first contact cases the sample data may be all that the input file for a case contains. For subsequent contacts to a household, the input file may also contain information obtained from previous contacts. This may include people's names and data. First the user must specify whether an input file is needed or not. The user may opt out of creating an input file if one will be obtained from another source, although the skeleton instrument is normally used early enough in the development cycle before test input files have been created. If an input file is needed, then the user needs to specify the following things:

1. The number of cases to create in the input file and how many people in each case? When creating an input file, the user should consider the type of test the input would be used for. The skeleton instrument allows for a 4 person per household input or a 20 person per household input file. Since the size of the initial .bdb file for a case can be relatively small, for load tests where we want to simulate the collection of data and increase sizes of the .bdb file, by selecting the 20 person household, one can more closely approximate the transfer of completed data (this was used primarily in the testing of our CATI environments).

2. If the instrument is for CAPI testing, the number of Regional Office codes to include in the file and how many cases to create for each Regional Office (RO) code. The testing done for CAPI may require the use of two or more RO test databases to test for such things as respondents moving between regions.

3. The caseid number to start with. The Census Bureau uses an 8-character caseid number as the primary key for the Blaise databases. It is generally assigned sequentially starting with 0000001, but there might be reasons a different sequence is needed. If multiple input files are desired, this allows the user to select blocks of case IDs for each input to avoid duplication of case IDs.

4. Whether the resulting Blaise database files should be zipped or not. The case management systems maintain the Blaise data for each case separately, zipped and stored in a database BLOB field, but depending on what the skeleton instrument is being used for, the user may or may not want the files zipped. Some testing is only concerned with importing or exporting data into and out of the instrument. The full process of unzipping a BLOB field may not be ready for testing until other tests are complete.

5. The name of the output file. The output file that is generated from the manipula script is actually the input file used to setup the Blaise databases. If the skeleton instrument is used for a full systems test, the user can specify the name of the file that the systems are expecting for the test. An input file will have the data in ASCII format ready to be loaded into the Blaise database by the setup manipula script also generated by the skeleton instrument process and ready for testing.

3.5 Output Documentation

Information is then collected about what type of output documentation is required. The user may choose to receive data descriptions for ASCII output or SAS dataset definitions for ASCII relational output by running the appropriate Cameleon scripts.

3.6 Root Directory

Finally the user is prompted for the root directory for where the process is to build the skeleton instrument and related files. The root drive letter will be added to the beginning of directory paths. The Census Bureau establishes a set of directories where the instrument will be loaded and run from. This directory structure can be created on any drive letter within the network environment. This way the generic instrument can be created in a network environment that can be shared by several users.

4. Build the Files

After all the questions have been answered it is time to begin building the files. At the conclusion of the **"getsurveyinfo"** questionnaire control is returned to the **"getinfo"** script, which in turn calls the **"skeleton_build"** script. This script constructs all of the files and prepares them for delivery to the various systems. A directory structure that corresponds to the directory structure on our case management systems is created beneath the root directory indicated above. An additional sub-directory (named "build") is also created. This is the directory where all the supporting files of the build process are copied or created. This is also the directory in which the input file will be built.

4.1 Create the Input File

If an input file is required, the script will create one using the information previously collected about which record types are required, how many cases to create (by RO, if CAPI), and the starting caseid. See Attachment A for the section of script that creates the input file. The starting input file for the skeleton instrument is a predefined set of people's names and data that would normally be collected during an interview. Based on the number of cases to be generated, one predefined file would be used to have four-person households or another predefined file would be used having twenty-person households represented.

4.2 Create the Instrument

Now it's time to create the instrument. The starting point is the record type include files discussed previously. It then pulls in other include files specific to the collection mode of the survey. The instrument, when executed by case management, allows the user to enter some demographic data as well as values for various control items (such as outcome code) that will be passed out to the case management system for it to act upon.

4.3 Create the Scripts

Although the specifics of the scripts may vary from survey to survey, there is a standard set of scripts that all our CATI and CAPI instruments use. The following is the basic processing flow for CATI and CAPI:

Step 1: (CAPI only) The CAPI Case Management system creates a **transfile**, which contains parameters used by the CAPI Transaction script.

Step 2: Case Management program (CATI or CAPI) creates a **<caseid>.in** file, which contains information collected or updated in the case management system that needs to be updated in the instrument data (or Blaise database).

Step 3: Case Management program runs a Transaction script (**cati_trans** for CATI or **capi_trans** for CAPI). The Transaction script controls the remaining instrument-related steps. See Attachment B for an example of a CAPI Transaction script. Different "transaction codes" cause the Transaction script to perform different steps. This allows the Transaction script to be the sole communication point between the case management systems and the instruments. See Attachment C for a list of the various transaction codes the Census Bureau use.

Step 4: The Transaction script runs a Casemgt_In script (**cati_casemgt_in** for CATI or **capi_casemgt_in** for CAPI). The Casemgt_In script updates the Blaise database with the data from the **<caseid>.in** file.

Step 5: The Transaction script calls the instrument, using information from the **transfile** for CAPI and using a standard run string for CATI.

Step 6: When the interviewer exits the instrument, the Transaction script runs a Casemgt_Out script (**cati_casemgt_out** for CATI and **capi_casemgt_out** for CAPI). The Casemgt_Out scripts creates a **<caseid>.out** file, along with some other output files, which contain information collected or updated in the instrument that needs to be updated in the case management system.

Step 7: Control passes from the Transaction script back to the case management system, which will then update its data using the information in the **<caseid>.out** and other output files and put the case away.

Thus, the standard set of scripts includes **<mode>_trans**, **<mode>_casemgt_in**, and **<mode>_casemgt_out,** where **<mode>** is for a CAPI or CATI instrument type. Each is different in the way the instrument is loaded for the interview and what data is passed into and out of the instrument.

In addition to the scripts used in the interviewing process, a **setup** script is also required. This script is run on the Census Bureau's Master Control System (MCS) instead of at the CATI or CAPI case management level. This is the script that creates the Blaise databases using the data in the input file. Each of these scripts is created using a predefined file, which is read one line at a time, and then the new instrument specific script is written out, one line at a time, using the correct record types.

The skeleton instrument process will create each of the appropriate scripts, based on whether the user indicated this was to be a CATI or a CAPI instrument. See Attachment D for the code that creates "*CAPI_CASEMGT_IN.MAN*". To create this script the skeleton build process reads a line of text from a generic file and writes a line of text to the instrument specific file, adding record type include statements where needed. This is where answering questions at the beginning of the process is now being acted upon. See Attachment E for the code that creates the Setup script. The setup script is a little more complicated but basically is uses a predefined generic setup script and then adds the instrument specific include statements for the record types. Additionally many new fields are added based on some newer record types. This is one of the great features of the skeleton instrument is it's adaptability. There are about nine of these instrument scripts that are created by the skeleton build process.

The process then compiles all the scripts. This process changes to the working directory based on the defined directory structure. Then each of the generated scripts is prepared using Blaise b4cpars.exe. This process will create the MSU files, which are the prepared versions of the source. Later these files may be added to the zip file for distribution.

4.4 Create Blaise Database

The process then runs the setup script against the input file to create the individual Blaise databases. The setup script uses the specific record types for the skeleton instrument. The setup script is similar to the one that will eventually be used to load data into the database during production. If the user requested that the Blaise files be zipped, the individual zip files are created at this point. This is where the path to the WinZip application is used.

4.5 Zip the Files for Distribution

The instrument and scripts are zipped and named appropriately for delivery to whichever system is using it for testing.

4.6 Create the Supporting Structure

In addition to the standard files that are generated, a number of batch files are created which are used to run the various scripts when required. These batch files simulate the areas of the process that may still be under construction. The following batch files are created:

1. Setcase.bat — This batch file will run the Setup script, which loads the input file into the Blaise database. This is the process where names and phone numbers and other supporting data are loaded for the interview.

2. RunCAPI.bat — This batch file simulates the Blaise scripts that CAPI case management executes to run the instrument and receive the data returned from the instrument. This is the process that actually runs the instrument for the interview. Data collected during the interview is then extracted from the Blaise database and prepared as an ASCII file to be loaded into the CAPI case management system.

3. RunCATI.bat — This batch file simulates the Blaise scripts that CATI case management executes to run the instrument and receive the data returned from the instrument. This is the process that actually runs the instrument for the interview. Data collected during the interview is then extracted from the Blaise database and prepared as an ASCII file to be loaded into the CATI case management interface.

4.7 Generate Output and Descriptions

Finally, the process will generate the data descriptions or the SAS definitions depending on the user's selection for output documentation. Depending on how many cases were created this process will compile each of the parts of the SAS output relational files and generate a Blaise database structure. The process will then use Cameleon.exe to run the sas.cif script file to make the SAS dataset with specific data from each case.

5. Uses of the Skeleton Instrument

The following describe some of the ways the skeleton instrument has been used at the U.S. Census Bureau:

5.1 GUI Laptop Case Management Testing

This was the initial system that prompted the need for a skeleton instrument. We were converting from a DOS based (Clipper) Case Management system running CASES interviews, to a Windows 2000 environment running Blaise. We were trying to use the same type of data exchange files between case management and the instrument that we used for CASES. Instead of a consolidated data structure stored in folders on a laptop, we were changing to one set of data files per case zipped up and stored in a BLOB field in an Oracle database. More than just testing the interface at this point we were also coming up with a standard processing strategy for handling the Blaise surveys. Many of the transaction scripts from the skeleton instrument were used as the starting point for production survey scripts for the initial surveys.

5.2 WebCATI Development Testing

Since the WebCATI system was the first system to use a browser for control, we needed to test the handoff between the control system and the Blaise instrument. They also required a tagged output file to be fed from the Blaise instrument to the WebCATI control system. As with the laptop case management testing, the scripts were initially developed for the skeleton instrument and then refined and deployed with the real instruments.

The skeleton instrument was also used in the initial load testing of the system. For the load testing we were not using live interviewers conducting live interviews; instead we were simply timing how long it would take to extract cases from the BLOB fields in the database, launch the instrument, and put the case down again. We were able to create the input file with varying size cases and conduct the load test without having to wait for input files from another division or a finished survey instrument.

5.3 New Feature Testing (Telephone Questionnaire Assistance)

The WebCATI system also introduced a new dilemma for handling incoming calls for a survey that employed telephone questionnaire assistance as one of its collection methodologies. In all of our other surveys, the case was called up by some identifying field – case ID or control number and the data already existed in the Blaise database format. This new feature required that a case be generated on the fly (instead of being pulled out of a database) and the control information for that case was then obtained during the interview. The skeleton instrument was adapted to allow for this type of transaction, and the skeleton instrument scripts were modified to handle a “TQA” transaction. Once again, the changes were made and tested using the skeleton instrument, and then they were moved into the production instrument once they had been proven.

6. Conclusion

The skeleton instrument was developed to provide case management with an way to test interfaces with an instrument and to test process control before and after the survey instrument is loaded. Many times the amount of time to develop an automated survey can be shortened if several processes are developed at the same time. Having a general survey set of interfaces can provide control systems a way of testing their interfaces with an instrument long before the actual instrument is complete. This skeleton instrument can provide some very specific instrument characteristics for testing. Only those fields that need to be passed back and forth with the control systems for any type of instrument can be created. Adding to that a generic set of features that can be used to load test the systems process improves the quality of our surveys.

Acknowledgements

I would like to acknowledge Mike for writing the skeleton instrument. I would like to thank Tom, Annetta, Mike, Sue, and Karen for writing, editing, and providing valuable input into this 9th International BLAISE Users Conference presentation paper.

Michael Haas, U.S. Census Bureau, Author/Developer
Thomas Spaulding, U.S. Census Bureau, Author/Manager
Annetta DePompa, U.S. Census Bureau, Author/WebCATI
Susan Galeano, U.S. Census Bureau, Editor/Version management
Karen Bagwell, U.S. Census Bureau, Author/Manager

Attachment A – Create Input File

```
Process CreateInput

SETTINGS
  DESCRIPTION = 'Replicates a one case SCIF into many cases '

USES

DATAMODEL NewSCIF
 FIELDS
  I_RecType       :  STRING[2]     {=  10 }
  I_RecSubType    :  STRING[2]     {=  00 }
  I_ROSTERINFO_10 :  STRING[12]
  I_CASEID        :  STRING[8]
  I_CTRLNUM       :  STRING[24]
  Blank1          :  STRING[2]
  I_SITE          :  STRING[2]
  Blank2          :  STRING[86]
  I_SURVEY        :  STRING[4]
  Balance         :  STRING[859]
ENDMODEL

INPUTFILE
   infile : Newscif ('orig.inp', ASCII)

OUTPUTFILE
  OutFile : NEWSCIF ( 'new.inp', ASCII)

  SETTINGS
    OPEN = YES
    CONNECT = YES

AUXFIELDS
  I      : INTEGER  {Counter}
  N      : INTEGER
  SW     : INTEGER
  R      : String[2]
  D     : INTEGER
  Surv   : String[4]

MANIPULATE
 R := Parameter(2)   {Site code}
 N := Val(PARAMETER(1)) {Number of cases to be created}
 Surv := Parameter(3)  {Survey ID}
 SW   := Val(PARAMETER(4)) {Starting caseid}

IF R = '' THEN R := '00' ENDIF
 D := N + SW - 1
IF D < 10 THEN
   For I := SW to D DO
```

```
    Infile.OPEN
    Infile.READNEXT     {Reads the first record of the input file
}
    WHILE NOT (Infile.EOF) DO
          { display (str(N) +'= N '+ str(SW) +'= SW '+ str(D) +'=
D ',wait)}
        IF Infile.I_RecType = '10' THEN     { Determine record
type and load appropriate date }
           Outfile.I_CASEID := '00' + R +'000' + STR(I)
```

… {more code here}

```
           OutFile.WRITE
        ENDIF
      Infile.READNEXT
    ENDWHILE
   INFILE.CLOSE
   ENDDO
{ENDIF}
{  okay to here}
ELSEIF D < 100 THEN
                IF SW < 10  THEN
                       For I := SW to 9 DO
                            Infile.OPEN
```

… {more code here}

```
ELSEIF

N < 1000  THEN

 For I := 1 to 9 DO
   Infile.OPEN
   Infile.READNEXT     {Reads the first record of the input file
}

   WHILE NOT (Infile.EOF) DO

      IF Infile.I_RecType = '10' THEN    { Determine record type
and load appropriate date }
```

… {more code here}

```
        OutFile.WRITE
      ENDIF
   Infile.READNEXT
   ENDWHILE
   INFILE.CLOSE
 ENDDO
```

```
For I := 10 to 99 DO
   Infile.OPEN
   Infile.READNEXT      {Reads the first record of the input file
}

   WHILE NOT (Infile.EOF) DO

      IF Infile.I_RecType = '10' THEN     { Determine record type
and load appropriate date }
```

… {more code here}

```
    ENDIF
   Infile.READNEXT
   ENDWHILE
   INFILE.CLOSE
ENDDO
```

… {more code here}

```
ELSEIF

N < 10000  THEN

 For I := 1 to 9 DO
```

… {more code here}

```
 {uncomment out the next section to handle up to 1,000,000 cases}

{ ELSEIF



 N < 1000000  THEN
```

… {more code here}

```
{ENDIF }
```

Attachment B – Transaction Script

Capi_Trans.man

```
Process Run_Interview

SETTINGS
  DESCRIPTION = 'Transaction processing'

USES

 INPUTMETA 'inst'
 INPUTMETA2 'outcomes'

 DATAMODEL TRANSACTION { Meta file (i.e. bmi file) for TRANSACTION
record }
                       { Also used for logging transactions}
   FIELDS
     CASEID         : String [8]
     BLANK1         : String [1]
     CONTROL_NUM    : String [24]
… {more code here}
     Time           : TIMETYPE     {time of transaction}

 ENDMODEL



 DATAMODEL ERRORFILE {Writes out ERROR messages to the file}

    FIELDS
      message     :  String [70]
      ERRORNUM    :  String[3]

… {more code here}

 ENDMODEL

UPDATEFILE   {The survey database}

  UpFile : INPUTMETA ('00000001', BLAISE)    {314}
   SETTINGS
    CONNECT = NO
    OPEN = NO {Gets opened in the manipulate section}

INPUTFILE    {The TRANSACTION file that case management creates}

… {more code here}


AUXFIELDS

  Verify_ID    : STRING[8]  {Used for comparison of case's CaseID and
TRANSACTION CaseID}
```

```
  Verify_CTRL  :  STRING[24]   {Used  for  comparison  of  case's  control
number and CONTROL_NUM trans rec}

  AUX_TRANS   : STRING
  AUX_ERRFLG  : STRING[3]
  AUX_DBINFO  : STRING[35]
  RO_FR_ID    : STRING[5]

  Reslt1             :  INTEGER    {Stores result of call to script to
update instrument}
```

… {more code here}

```
PROCEDURE RunDep    {Conduct the interview}
   PARAMETERS
        IMPORT
            THECASE,THEINSTRUMENT : STRING     {this is the caseID data
file used as input}
   INSTRUCTIONS
             Reslt2:=  edit  (THEINSTRUMENT  +  '  /Mskeleton.bwm  /f'  +
THECASE + ' /k' + THECASE + ' /g /x')
            IF Reslt2 <>0     THEN   { The DEP failed to execute }
               Prob.OPEN
               Prob.Reslt_no := '2'
               Prob.Reslt_val := Reslt2
               Prob.message := 'could not interview case'
               Prob.WRITE
            ELSE
               Flag9.OPEN(thecase +'.inst.flg')
               Flag9.write
            ENDIF
   ENDPROCEDURE
```

… {more code here}

```
MANIPULATE
   InTRANS.OPEN
   InTRANS.READNEXT    {Read the TRANSACTION file }

   WHILE NOT (InTrans.EOF) DO

   THECASE        := InTrans.caseid           {CASEID}
   THEINSTRUMENT   := InTrans.instrument         {Blaise instrument .bmi
file}
   UPINSTRUMENT     := InTrans.update_case         {Manipula program to
update instrument}
   UPCASEMGMT        := InTrans.update_casemgt     {Manipula program to
update case mgmt}
   refinst        := 'outcomes'
```

… {more code here}

```
    ELSEIF InTRANS.TRANS_CODE = '09' THEN

          Caseidin(THECASE,UPINSTRUMENT)
```

```
          IF Reslt1 = 0 THEN     {If update from case management was
successful, run DEP}
          {Upfile.release}
{LoadCATI}          Runexternal




{{RegularCATI}                       Reslt2   :=   RUN('note_editor.exe
'+THECASE+',P,E,'+RO_FR_ID+' ',wait)}

{{RegularCATI}                  IF Reslt2 = 0 THEN   {If interview was
successfully run, update case management}   }
                            Caseidout (THECASE,UPCASEMGMT)
{{RegularCATI}             ENDIF      }
          ENDIF   {reslt = 0}

      ELSE {unrecognised transaction code send message}
         AUX_ERRFLG := 'T02'
      ENDIF   {InTRANS.TRANS_CODE = '01' THEN }

      IF Intrans.log_trans = '1'  THEN
         Log.WRITE
      ENDIF
 ENDIF
{ display ('stop 1 ',wait)}
 IF SUBSTRING(AUX_ERRFLG,1,1) = 'T' THEN

    Prob.OPEN {added by Mike Haas}
    Prob.initrecord
    {format the TRANSACTION error message}
    AUX_TRANS  := InTrans.CASEID       +
                  InTrans.BLANK1        +
                  InTrans.CONTROL_NUM  +
                  InTrans.TRANS_CODE   +
                  InTrans.OUTCOME_CODE +
                  InTrans.AGENDUM_CODE +
                  InTrans.FLDREINT_FLG
```

… {more code here}

```
  InTRANS.READNEXT    {Read the TRANSACTION file }
ENDWHILE
```

Attachment C – List of Census Bureau CATI and CAPI Tranasction Codes

CAPI

01 – Post-interview action code update
02 – Late mail return - update action and outcome code
03 – SFR observed case - set FLDREINT from trans file
09 – Run the interview

WebCATI

17 – Case is a TQA case
18 – Run the interview - process <caseid>.in file
19 – Run the interview - no <caseid>.in file

Attachment D – Create CAPI_Casemgt_In Script

Capi_casemgt_in.man is a Manipula program for reading in the ASCII file CASEID.IN data for the instrument to use during the interview.

```
   Outfile4.OPEN(substring(start_dir,1,len(start_dir)) + '\' + study +
'\build\e-inst\capi_casemgt_in.man')
   Outfile4.INITRECORD
   WHILE NOT (Infile4.EOF) DO {capi_in}

         IF Infile4.datatext = ' INPUTMETA ''ske_le''      ' THEN
         Outfile4.datatext :=   ' INPUTMETA ''inst'''
         Outfile4.WRITE

         ELSEIF substring(Infile4.datatext,1,6) = '{RT 10'  THEN
            IF RT10 = '100000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  100000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 20'  THEN
            IF RT20 = '200000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  200000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ELSEIF RT20 = '200600' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  200600}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif

… {More types can be added here}

            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 21'  THEN
            IF RT21 = '210000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  210000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
         ELSEIF substring(Infile4.datatext,1,6) = '{RT 80'  THEN
            IF RT80 = '800000' THEN
                if  substring(Infile4.datatext,1,11)  =  '{RT  800000}'
THEN
                    Outfile4.datatext :=   Infile4.datatext
                    Outfile4.WRITE
                endif
            ENDIF
```

```
        ELSE
        Outfile4.WRITE
        ENDIF
 Infile4.READNEXT
 ENDWHILE
outfile4.close
```

Attachment E – Create Setup Script

{create the setup script with the proper RT files}

```
   Outfile2.OPEN(substring(start_dir,1,len(start_dir)) + '\' + study +
'\build\e-inst\setup.man')
   Outfile2.INITRECORD
   WHILE NOT (Infile2.EOF) DO

      IF Infile2.datatext =   '    INCLUDE  "Rt100000in.inc"' THEN
         Outfile2.datatext := '    INCLUDE  "Rt' + RT10 +'in.inc"'
         Outfile2.WRITE
      ELSEIF   Infile2.datatext   =               ('                  IF
In1000.Rt1000in.I_RecType  +  In1000.Rt1000in.I_RecSubType  =  ''1000''
THEN')   THEN
         Outfile2.datatext :=    '       IF In1000.Rt1000in.I_RecType +
In1000.Rt1000in.I_RecSubType = ''' + substring(RT10,1,4) + ''' THEN'
         RT10vers :=substring(RT10,1,4)
         Outfile2.WRITE
      ELSEIF Infile2.datatext =       ('                OutFile.SEGMENT
:= In1000.Rt1000in.I_SEGMENT      {comment out for RT1080}') THEN
            IF RT10vers <> '1080'  THEN
            Outfile2.WRITE
            ENDIF
      ELSEIF      Infile2.datatext      =                            ('
OutFile.RT1000.SEGMENT                    :=  In1000.Rt1000in.I_SEGMENT
{comment out for RT1080}') THEN
            IF RT10vers <> '1080'  THEN
            Outfile2.WRITE
            ENDIF
      ELSEIF Infile2.datatext =       ('                 Outfile.LISTID
:= In1000.Rt1000in.I_LISTID       {comment out for RT1080}') THEN
            IF RT10vers <> '1080'  THEN
            Outfile2.WRITE
            ENDIF
      ELSEIF Infile2.datatext =     ('            Outfile.RT1000.LISTID
:= In1000.Rt1000in.I_LISTID       {comment out for RT1080}') THEN
            IF RT10vers <> '1080'  THEN
            Outfile2.WRITE
            ENDIF

      ELSEIF Infile2.datatext = '    INCLUDE  "Rt200000in.inc"' THEN
         Outfile2.datatext :=   '    INCLUDE  "Rt' + RT20 +'in.inc"'
         Outfile2.WRITE

… {More types can be added here}

        ELSEIF RT21 = '218000' THEN
           if substring(Infile2.datatext,1,11) = '{RT 218000}' THEN
            Outfile2.datatext :=   Infile2.datatext
            Outfile2.WRITE
           endif
        ENDIF
      ELSE
        Outfile2.WRITE
      ENDIF
```

```
 Infile2.READNEXT
 ENDWHILE
outfile2.close
```

Randomly Generating Interviews to Validate a Complex Survey

Lindsay Walton, National Research Bureau, Auckland, New Zealand

1. Introduction

The New Zealand version of the worldwide mental health survey was an extremely complex instrument (> 3000 questions and a seemingly unlimited number of routings) being a combination of the original US survey together with many changes made by New Zealand. The instrument was divided into 22 reasonably self contained modules, each representing a different area of mental health. After finishing the initial coding changes we tried a number of ways to validate the instrument. These included:

- A walkthrough "translating" the Blaise code to the hard copy (which served as our specifications)
- Having testers try to go through as many routes in each module as possible and check the routings were consistent with the hard copy.
- Ran a pilot of around 200 interviews, noting any problems and then examining the number of responses to each question to see if some questions had been incorrectly missed out or if a particular question had an inappropriate number of responses when compared to another.

All of these methods were successful in finding problems which we were able to correct. The problem was that we were finding errors on a regular basis and with the sheer complexity of the routings, it became obvious that we would never be able to say with reasonable certainty that the instrument was clean.

We needed to find an alternative approach but at the same time, we were very aware that time was starting to become critical and we needed to get the survey into the field as soon as possible.

2. Overview of the problem

The main problem was the complexity of the routings which were virtually impossible to test using just a trial and error basis. Some examples are:

***NZSU73.0** INTERVIEWER CHECKPOINT (SEE *SU72 SERIES, ONLY
MARIJUANA SYMPTOMS (NZSU72..2, 72a.2,…72i.2)
ZERO TO TWO RESPONSES CODED '1'..... 1
ALL OTHERS..... 2

***SU73**. INTERVIEWER CHECKPOINT: (SEE ***SU72 SERIES – IGNORING NZ MARIJUANA QUESTIONS**)

ZERO TO TWO RESPONSES CODED '1'.....1 **GO TO *SU87**
ALL OTHERS.....2

---

*SU73.1 INTERVIEWER CHECKPOINT: (SEE ***SU47a** , ***SU47b, *SU47c, *NZRSU47e, *NZRSU47f**)

ONE OR MORE RESPONSES CODED '1'..... 1
ALL OTHERS..... 2 **GO TO *SU81**

PH100. INTERVIEWER CHECKPOINT:** (SEE ***D24a** - ***D24c, *D24e** - ***D24f, *D26a** - ***D26e, *D26g** - ***D26i, *D26j,*D26m, *D26o, *D26p, *D26r** ,D26s, *D26u-*D26w, *D26aa-*D26ee, *M1, *M8, *M5,** ***M7a-*M7o, *PD17a, *PD14, *SP16, *SP17, *SO2, *SO16, *SO17, *AG13, *AG14, *AG15, *AG16, *G17.1, *SD4, *SD6, *SD17, *SD19, *SR1, *SR2, *SR9.1**, ***SR10**, ***SR11**, ***SR12**, ***SR13**, ***SR14**, ***SR17** SERIES)

THE SUM OF THE COUNT EQUALS FIVE OR MORE BASED ON INCREMENTING THE COUNT BY ONE FOR EACH OF THE FOLLOWING NINE SETS OF QUESTIONS IF ONE OR MORE QUESTIONS IN THE SET ARE CODED '1': ONE OR MORE '1' CODES IN ***D24a** - ***D24c**, ONE OR MORE '1' CODES IN ***D24e** - ***D24f**, ONE OR MORE '1' CODES IN **(*D26a, *D26b, *D26c, *NZRD26d, *D26e)** , ONE OR MORE '1' CODES IN ***D26g** - ***D26i**, ONE OR MORE '1' CODES IN ***D26j**, ONE OR MORE '1' CODES IN (***D26m** OR ***D26o**), ONE OR MORE '1' CODES IN (***D26p** OR ***D26r** OR ***D26s**), ONE OR MORE '1' CODES IN ***D26u-*D26w**, ONE OR MORE '1' CODES IN ***D26aa-*D26ee** AND (***D9** EQUALS '1' OR ***D12** EQUALS '1') 1

M1** EQUALS '1' AND **M8 EQUALS '1'** AND (M9** EQUALS '4' or '5' OR

***M9a** EQUALS '1' OR '2' OR ***M33** EQUALS '1')..... 2

***M5** EQUALS '1' AND SUM OF '1' RESPONSES IN ***M7a-*M7o** IS '4' OR MORE

AND (***M9** EQUALS '4' or '5' OR ***M9a** EQUALS '1' OR '2' OR

***M33** EQUALS '1')..... 3

***SP16** EQUALS '4'-'5' OR ***SP17** EQUALS '1'..... 6

SO2** EQUALS '3' AND (SO16** EQUALS '4'-'5' OR ***SO17** EQUALS '1')..... 7

***AG13** EQUALS '1' OR ***AG14** EQUALS '1' OR ***AG15** EQUALS '4'-'5' OR

***AG16** EQUALS '1'..... 8

***G17.1** EQUALS '1'..... 9

***SD4** EQUALS '1' OR ***SD6** EQUALS '1' OR ***SD17** EQUALS '1' OR

***SD19** EQUALS '1'..... 11

***SR1** EQUALS '1' OR ***SR2** EQUALS '1'..... 12

ALL OTHERS..... 16

GO TO *PH101

*PH100.1 (SEE ***PH100**)

***PH100** EQUALS '1'-'15'..... 1 **THESE ARE SOME OF THE "LONG GROUP." GO TO *PT1**
ALL OTHERS 2

While most questions weren't this complex, there were many checkpoints throughout the survey, often with some complexity, and when considered together represented an instrument that was

difficult to manage.

3. Overview of the approach

The complexity of some of the modules meant that it wasn't realistic to do manual checking. Our solution was to automate the process using a 2 pronged approach.

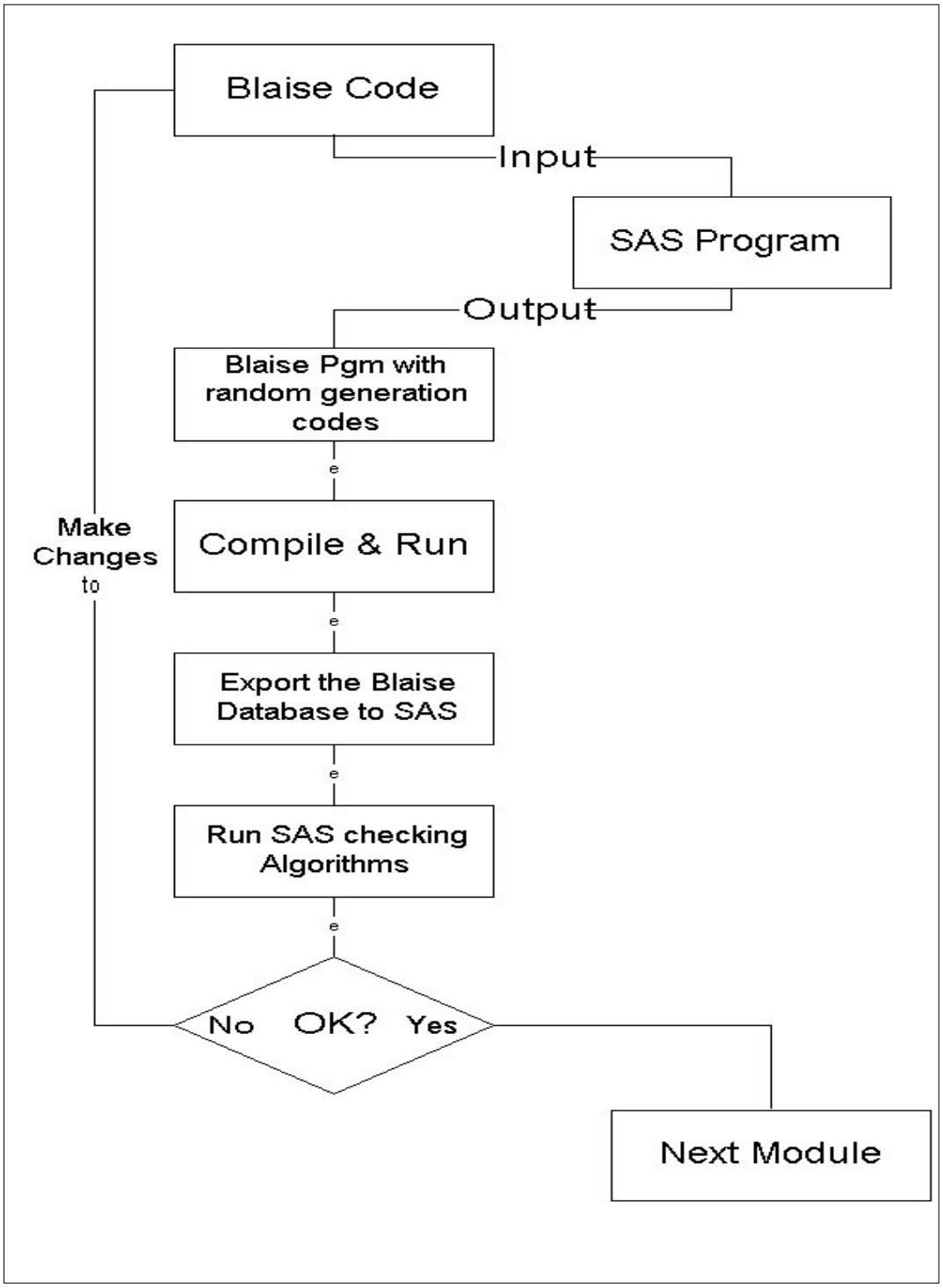
3.1 Generate random surveys

We felt somehow we needed to be able to generate a large number of surveys, using a random walk technique, so that we could feel reasonably confident that we had covered all or virtually all combinations of answers. To do this, we decided to try to convert all the questions in the survey into assignment statements using random generation. For example we needed to change a standard yes/no question into an assignment statement that would give yes 45% of the time, no 45%, and don't know and refused 5% each.

3.2 Check the surveys

The second part of the approach was to find a way of confirming these “interviews” were correct based on the logic of the paper version of the survey. For this to be successful, we felt it was important to change the emphasis of the checking from the routing based Blaise approach to a question-by-question analysis and using a different language, SAS. This was to reduce the chance of making the same mistakes in checking as were in the survey itself. For example, if the answer to question 1 is yes you ask question 2 otherwise you ask question 3. In our approach we would focus on each question and for question 3, decide that it should be populated if and only if question 1 is not yes, otherwise it must be empty.

3.3 Flow Chart of System



The above diagram shows that we needed to have 2 locations containing Blaise source code – one for the "normal" code and one, with exactly the same routing logic, but containing the random functionality. However it was vital that we only maintained one lot of source code. The SAS program at the top is how this was achieved. Essentially it read the "normal" Blaise code as an Ascii file, made some changes to the code and output it to another Blaise program containing

only assignment statements and no actual questions. This program is then compiled and run, producing a Blaise database containing anything from 200+ records. This database is then exported to a SAS dataset and then checked by the specially written SAS programs. In this way we could keep making changes to the "normal" Blaise code and only have to run a SAS program to produce the code to generate the random interviews. It was very quick and simple and meant we didn't have to make manual changes to both versions.

4. Randomly generating interviews – an example

The change in code for a simple Yes/No question was something like:

Original Module

```
LOCALS
...
FIELDS
...
SU2b2 "..." : TYesNo
...
RULES
...
SU2b2
...
```

After Random Generation

```
LOCALS
Rnum : INTEGER
...
FIELDS
...
SU2b2 "..." : TYesNo
...
RULES
...
{SU2b2}
 Prandom(101, Rnum)
 if RNum <=45 then SU2b2 :=C01
 elseif RNum <= 90 then SU2b2 :=C05
 elseif RNum <= 95 then SU2b2 :=DK
 else SU2b2 :=RF
 endif
...
```

Note that C01 is "yes" and C05 is "no". All that the SAS program did was comment out the original code and add the new code segment as well as adding the "RNum" field to the locals section.

The SAS program relied on certain consistencies in the programming standards of the Blaise modules. In general the tidier and more consistent the Blaise code, the easier the SAS job is. Probably the most important programming standard was "one question on one line" in the RULES section. This made it much simpler to identify the questions.

So how did the SAS program know that question SU2b2 is a Yes/No question? Essentially it made two passes of the Blaise module, the first pass read just the FIELDS section and matched the question code with its answer. In other words, it found SU2b2 and then skipped the text of the question and found its answer was the type TYesNo. This was written to a SAS Format and used in the second pass when it was changing the contents of the RULES section. The TYesNo answer was linked to a SAS macro that wrote the code in the box above.

This was the simplest situation: a question in which we were quite happy to have 45% of answers to be "Yes" and 45% to be "No". However there were other questions which needed a slightly different approach.

Ultimately the random answers had to reflect the logic of the survey and our desire to go down as many routes as possible. For example if the first question in a module is a Yes/No type and a "No" answer skips to the next module, then it is better to change the random amounts so that "No" has a very low chance of being selected. This is easy to do in SAS by a statement like

```
If ans='TYesNo' and Question in('SU1') then %yesno(85,5,5,5)
else if ans='TYesNo' then %yesno(45,45,5,5)
```

What is happening here is that for question SU1, the probability of it having an answer of "Yes" is 85% with "No", "Dont Know" & "Refused" at 5% each.

Another reason to change the randomising were for situations where 20 yes/no questions were asked and if the respondent answered "Yes" to 3 or more then they would go a different path to those who had fewer than 3. In this case the ratios can be changed so that "yes" has a lower chance of being selected, so ideally half the "interviews" would have 3 or more "yes" responses and half less than 3.

Multiple choice questions: These needed to be considered on a question by question basis. Some questions had upwards of 20 possible answers and generating all possible combinations of one question together with other long multiple response questions, together with all the other questions in the survey was not practical – and not necessary. For these questions, it was matter of assigning a small number of responses (5-10) that would be appropriate for any logic resulting from the answers to these questions. For example question SR105 had 14 answers:

```
SR105    "What kind of healer did you see?
            Record all mentions!"  :
       TSpecialist
```

Where TSpecialist is:

```
TSpecialist = SET OF
        (C01 "Acupuncturist",
         C02 "Biofeedback specialist",
         C03 "Chiropractor",
         C04 "Energy healing specialist",
         C05 "Exercise or movement therapist",
         C06 "Herbalist",
         ...
         C13 "Dietician",
         C14 "Other (specify)" )
```

The SAS macro to generate the “randomised code” was

```
%macro mult14;
 %commentc;
 code=' Prandom(101, Rnum)'; output;
 code=' if Rnum <=15 then ' ||qst|| ':=[C01]'; output;
 code=' elseif Rnum<=30 then ' ||qst|| ':=[C02]'; output;
 code=' elseif Rnum<=45 then ' ||qst|| ':=[C14]'; output;
 code=' elseif Rnum<=60 then ' ||qst|| ':=[C07,C03]'; output;
 code=' elseif Rnum<=75 then ' ||qst|| ':=[C08,C04,C14]'; output;
 code=' elseif Rnum<=90 then ' ||qst||
':=[C14,C12,C13,C07,C06,C05,C04,C03,C02,C01]'; output;
 code=' elseif Rnum<=95 then ' ||qst|| ':=DK'; output;
 code=' else ' ||qst|| ':=RF'; output;
 code=' endif'; output;
%mend mult14;
```

resulting in this Blaise code:

```
{SR105}
 Prandom(101, Rnum)
 if Rnum<=15 then SR105 :=[C01]
 elseif Rnum<= 30 then SR105 :=[C02]
 elseif Rnum<= 45 then SR105 :=[C14]
 elseif Rnum<= 60 then SR105 :=[C07,C03]
 elseif Rnum<= 75 then SR105 :=[C08,C04,C14]
 elseif Rnum<= 90 then SR105
:=[C14,C12,C13,C07,C06,C05,C04,C03,C02,C01]
 elseif Rnum<= 95 then SR105 :=DK
 else SR105 :=RF
 endif
```

If the survey logic after this question takes people who have only seen one specialist down one route whereas those who have seen multiple go a different route, then this will work fine with 45% going the single specialist route and 45% the multiple with 10% of “Don't knows” and “refused” which also must be routed correctly.

4.1 Some Technical Issues

4.1.1 Compile problem (number 1)

When we finally had all the questions changed to random assignment statements, we found we were unable to compile the Blaise code as it requires a survey to have at least one question!! The way around this was to add a dummy question which would accept “empty” as a response. This enabled the program to compile but meant that couldn't run completely by itself. However this was easily overcome by placing a coffee mug on the space bar – it was all Blaise needed to complete an interview and restart a new one (but see 3.1.4 below)! The dummy question was:

```
intro2     "Press Enter" : STRING[1], EMPTY
```

4.1.2 – Compile problem (number 2)

When we ran a SAS program to rewrite a Blaise module and we already had a Blaise session open with the module open, Blaise would warn us that the contents of the module had changed and we would simply hit the “Ok” button. If the module wasn't open then Blaise gave no warnings. Then when we tried to compile, we got an error:

RULES or LAYOUT section terminated incorrectly or too long.

Initially this was very confusing because the code shown in the window had no obvious error. Although it took some time to find the problem, the solution was very simple – you just had to manually save the module in the Blaise command center. Then it compiled.

4.1.3 Adjustments to the controlling .BLA module

The survey had a key field which was called "IVNum" and was a string of length 5. Generating multiple interviews with unique keys would have required either writing a Manipula program or doing something with the date/time. We took the easiest option and temporarily commented out the PRIMARY and SECONDARY statements in the controlling .BLA program and just assigned the key field with the same constant for each interview.

4.1.4 Modelib Settings

In the modelib editor, under toggles --> Interviewing, ensure the following items were ticked:

Auto create file
Auto save when finished
Prompt save when finished
Auto enter

4.1.5 General

As the layout of the questions in the Blaise programs was not always consistent, the SAS program to read and make sense of them did get more complex than we were hoping.

5. Checking the interviews

This was done by running the SAS checking algorithms against the data that had been extracted from Blaise. In checking an "interview", the algorithm would search for an error, and if it found one, would report it and then exit to the next "interview". In this way it would only report the first error, and thus any subsequent errors in an interview would not be reported. Although this meant we took a few more iterations to clean a module, it meant that we didn't have to worry about the situation of errors in previous questions, incorrectly causing a later question to error.

Although there were a few different types of questions we had to deal with, the majority were handled by the three SAS macros below. If a problem is found, the macro “Fail” is called which outputs the record.

```
%macro fail(field, msg);
        do; errfield="&field"; errmsg=&msg; output; return; end;
%mend fail;

%macro check(field, z);
if (&z) then do;
        if (&field) eq . then %fail(&field, 'no value');
end; else do;
        if (&field) ne . then %fail(&field, 'has value');
end;
%mend check;

%macro value(field, val, z);
if (&field) ne . then do;
        if (&z) and ((&field) ne (&val)) then %fail(&field, 'bad value');
end;
%mend value;
```

an example of how these were used are:

```
data findErrors;
 set <randomly generated interviews from Blaise>;
 ...
 %check(m1, sc24 eq 1);
 %check(m2, m1 in(5,8,9));
 ...
 count=sum(m7a=1,m7b=1,m7c=1,m7d=1, m7e=1,m7f=1) ;
 %value(m8, 1, count ge 3);
 ...
run;
```

The first statement concerns question “M1” and requires that it must have a value (any value) if question SC24=1 otherwise it must equal missing.

Question “M2” is only asked if the answer to “M1” is either “No”, “Dont Know”, “Refused” (“Yes” in this survey was recorded as C01, “No”=C05).

The third example here is “M8” which is an assignment statement rather than a question. Its value must be 1 if the number of “Yes” responses to questions “M7a”-”M7f” is 3 or more.

6. Summary

The project was completed by 3 people in 3-4 weeks. Initially we thought we would run the random surveys in one hit for all modules and then see where the problem were. However it soon became obvious that it was preferable to do one module at a time. While the ultimate goal was to be able to run a random generation of many thousands of surveys, practically running 200-500 interviews for a module was generally enough although we would usually finish testing a module by running a survey with 2-3000 interviews. By the time it came to run the entire survey, 18000 interviews ran without error.

In the end we found 20-30 new routing errors. These were fixed and at the time of writing, 2000 real interviews had been sent to the USA for checking. The results were that the interviews had no routing errors and we just had some inevitable data cleaning work to do – which is an ongoing process.

This is a project that you would only consider for surveys with very complex routings. For simple surveys where one question flows into the next question, which can be checked manually, there is little need to do something like this to validate it.

Moving from 43 Generated Unique Instruments to One National Instrument

Roger Schou, National Agricultural Statistics Service, USA

1. Background

At the National Agricultural Statistics Service (NASS), we have been generating instruments for our Crops/Stocks Survey for over ten years. It is the only national survey that NASS conducts, where the instruments are unique for each state. The Crops/Stocks Survey instrument collects crop acreage and production data, stocks data, and general livestock data of farmers across the country. It is conducted quarterly. It has been a very successful concept.

The generation of instruments results in 43 unique instruments generated in our field offices. It allows each state to ask crops pertinent to their state in their preferred order. For example, some states prefer to ask soybeans before sorghum, while others prefer sorghum first. Also, within a given crop, some states prefer to ask the irrigated questions before the non-irrigated questions, while others prefer non-irrigated first.

The ability to generate their own instruments from pre-tested headquarters-written code gives the states the luxury of having it their way. Since each state field office collects their own state's data, this is the preferred method, at least from the field office perspective. There are some maintenance issues from the headquarters' point of view.

1.1 Generator Maintenance Issues

The generation process is driven by a specifications database, which is a Blaise data set. It contains one record for each quarter for each state. It keeps track of things like which crops are asked, in what order they are asked, any special file extensions, whether irrigated questions precede non-irrigated questions, and much more information needed to assemble and generate an instrument from a library of pre-tested code.

There are many files that are specific to a given quarter. The developer must remember to change the extensions of a number of files to the "live" version for that quarter, and distribute them to the states.

Many of the similar crops share a rules section. Making a change for one crop may adversely affect many other crops if the change is not implemented correctly. Sometimes there is extra routing that pertains to one or two crops. This additional routing can be confusing when reviewing the rules for a crop that does not use that particular routing. Also, the error checks are often more complex because all the crops sharing the rules file must be taken into account.

Since there are 43 unique versions running in the field offices, testing the instruments becomes a tedious process. Testing may be tedious with one national instrument as well, but getting to the point of testing is greatly reduced. Each version must be generated in headquarters in order to test the instruments and to support the questions posed by the field office staff. The process of generating all 43 of the unique instruments has been automated, but it still takes 1.5 hours to complete when all of the code is error free. In order to duplicate a reported problem during the

survey, the instrument must be generated (or copied from an earlier generation process) and sometimes a test sample must be initialized.

2. Changes in NASS

As survey management and technology continue to evolve, changes are needed to adapt to the evolution. Two changes in NASS that have a direct impact on the way we conduct the Crops/Stocks Survey are the introduction of Data Collection Centers, and the Question Repository. Both of these changes have been driving forces in the decision to move from the 43 generated unique instruments to one national instrument for the Crops/Stocks Survey.

2.1 Introduction of Data Collection Centers

A few of the NASS field offices have begun to function as Data Collection Centers, because hiring and maintaining a steady crew of telephone interviewers is much easier in states that have less competition in the work force. In other words, field offices in larger metropolitan areas have more difficulty hiring interviewers than offices in smaller cities.

As the Data Collection Centers increase the number of states for whom they are collecting data, the survey management of the Crops/Stocks Survey is becoming overwhelming. Since the Crops/Stocks Survey has a unique instrument for each state, the Data Collection Centers must setup each state's survey as a separate entity. They have quickly maxed out with the number of states that they can manage for a Crops/Stocks Survey. Since the scheduler looks at one particular survey, and each state is setup as a separate survey, call management is quite convoluted.

2.2 Question Repository

NASS is currently building a Question Repository. This will be used to create both paper and WEB versions of questionnaires for NASS surveys. One of the requirements of such a system is standardization. After reviewing the generation of unique Blaise instruments and the number of years spent developing this generator, it was determined that the Question Repository would create unique paper and WEB instruments for each state, but the crop order would need to be chosen from one national standard order. In addition, within a given crop, we would need to ask the cropping practices (irrigated and non-irrigated questions) in the same order.

By not allowing the states to uniquely customize their paper forms (as far as question order is concerned) an estimated two to four years of development time of the Question Repository will be saved. This estimate is based on the amount of time invested developing and testing the generator in Blaise.

3. A Decision Was Made

An executive decision was made to create one national Crops/Stocks Survey instrument that would only ask crops, stocks, and livestock data that are relevant to each state, with the crops and cropping practices asked in one national standard order. It was important that the states were made aware of what they would have to give up as well as what they would gain.

There were open lines of communication between headquarters and the states, and we found that the states were not as attached to maintaining their crop orders as they once were. From survey administrators, to developers, to sample designers, to analysts, to upper management, the idea was passed around and discussed, which led to total buy-in and finally the executive decision. The projected date for this standard order to be in production in all states (in Blaise and on paper) is March 2005. There will be smaller sample tests (exclusively on paper forms in September 2004 and December 2004) to measure any impact of changing the existing order of crops.

4. From Building Specifications to Driving Specifications

The specifications database is still a big part of the Crops/Stocks Survey, but its role has changed from containing "data needed in order to build an instrument" to "data needed to run the instrument." Instead of Manipula reading the specifications database in order to generate code, the data model reads it as an external file to determine which crops need to be asked in a particular state and whether cropping practices data need to be asked.

The specifications database still contains one record for each quarter for each state. Many of the fields that are required for building an instrument no longer have any value, because the instrument code already takes that into account. Similarly, new fields have been added to drive the instrument's routing of questions.

5. One National Instrument

In order to minimize the maintenance impact on a small headquarters development staff, we decided to create one national instrument that would collect data for any given state on any given quarter. The cost of the original development of this complex instrument should quickly pay off with the lower maintenance projected for the future of the instrument.

To understand the complexity of the one instrument concept, one needs to first understand the crop data being collected. There are over 50 program crops collected over the year. Some of these crops are further broken down into varieties. Not all crops are collected in all states, because all states don't grow all crops. On certain quarters, some states collect crop data by cropping practice, while other states do not. In some quarters only planted and/or harvested acreages are collected. In other quarters planted and/or harvested acreages and production numbers (either total production or yield) are collected. When a crop is broken down into varieties, not all states collect the same combinations of the varieties, and some states don't even break them down at all. Some states collect crop data for the varieties by cropping practice.

Between the generator system and the national instrument concept, there are differences in the individual crop files. The following two sections highlight these differences.

5.1 Individual Crop Files in the Generator System

In the generator system, every crop has one file for every structure it may be involved in over the year. Some examples of crop structures follow: 1) planted acreage only, 2) planted and harvested acreage, 3) planted and harvested acreage, production and yield, 4) planted and harvested acreage, production and yield for grain and silage, and so on. The structures for a particular crop change as the quarters change.

All of these crop files are created by the generator, and consist mainly of an "assignment block" and the fields section for the crop and its structure. The assignment block computes some locals and auxfields for things such as error limits, crop names, production and yield units, and conversion factors. The assignment block then is used as a type for one of the fields in the crop block. Sometime the unit of production differs from state to state on certain crops. In this case, the assignment block is different and the crop file is saved with a different extension.

The rules for the crop block are more generic and are shared between crops that have the same structures. Many times there is extra code in the rules file that pertains to one or two crops that are sharing it. This can cause some confusion when looking at the block from another crop's perspective. Also, when making a change to the rules for a particular crop, it may affect a crop that the developer is not aware of at the time.

5.2 Individual Crop Files in the National Instrument

The new national instrument has a maximum of two files for every crop. The first file is for the crop itself and the second file is for any cropping practices in which the crop may be involved. The individual crop block accounts for all possible structures of the crop. As in the generator system, the crop file consists of the assignment block and the fields section for the crop.

There is one corresponding rules file for every crop file, making a maximum of two rules files per crop. Since the rules are specific for each crop, there is no extraneous information or routing for other crops. A crop may now be updated without the risk of the change affecting other crops. However, if the update is required to a specific structure, then the change needs to be made in multiple crop files. This is bit more labor intensive than the generated process, but in the long run, the single national instrument is easier to maintain.

It is very important to the editors in the field offices to be able to use the jumping capabilities of the field tags in Blaise. Therefore, even though many of the crops' field sections are the same, none of them are shared. The majority of the fields have a field tag that is equivalent to an item code which corresponds to the paper form. This allows the editor to jump to any portion of the questionnaire without paging through numerous screens to get to their data.

5.3 Prepare Directives

Obviously, the number of extra fields needs to be kept to a minimum. Instead of defining all possible fields for the entire year and only routing the needed ones in any given quarter, prepare directives are used to only include the fields needed for the quarter. There are still a number of extra fields that some states do not need, but they must be present in case the Data Collection Center is calling for multiple states. However, the total number of extra fields is much less than it would be without the use of the prepare directives. This instrument works for any state in any quarter, as long as it is newly prepared each quarter.

The use of the prepare directives has greatly reduced the size of the instrument. For example, in quarters where only planted acreage is collected, the fields for harvested acreage and production do not exist in the instrument. This is not a big issue for CATI, but is a factor in Interactive Edit, as there will be fewer "empty" screens through which to page.

Prepare directives also eliminate the need for the developer to change extensions on files that are different from quarter to quarter. The prepare directives allow the appropriate file to be included in the appropriate quarter. Many of the tedious steps from the generated process have been eliminated.

Great care was taken when assembling the code to insure that 1) it would prepare for each quarter, and 2) the correct crops and cropping practices were routed for the appropriate states. It is very easy to have the correct code in place, and yet have it ignored because of the prepare directives. However, once the code is in place and tested, maintenance is projected to be much lower in the national instrument than it is in the generated instrument.

The developers have adopted an indentation standard with respect to the prepare directives, in order to more easily detect where they start and stop. This becomes even more helpful when the prepare directives are nested.

6. Conclusion

Converting from the generated instrument schema to the one national instrument schema is a large undertaking, but the end result will be a more easily maintained instrument that meets the requirements of NASS. The generated instrument scenario worked very well for over ten years as a proven method for creating customized instruments for the states. It no longer meets the needs of the environment to which NASS is moving.

The one national instrument allows the Data Collection Centers to collect data for the Crops/Stocks Survey for multiple states, keeping the data in the same dataset. Each state's survey will no longer need to be treated as a separate entity in the Data Collection Centers. The call scheduler will handle the call backs for all of the states for which they are responsible. Data Collection Centers may add additional states in a given quarter much more easily than with the generated instrument. Maintenance and survey preparation time by the limited headquarters staff will be greatly reduced.

The Question Repository has motivated NASS to move to a standard crop and cropping practice order for the Crops/Stocks Survey. This will require fewer resources in developing the Question Repository. The standardized question order opened the door to move to one national instrument for the Crops/Stocks Survey in Blaise, since the original driving force behind the generator was the ability to customize the questionnaire in the states.

With the idea of only maintaining one instrument for all states in all quarters, we wanted to minimize the size of the data model as much as possible. This is preferable by our data editors in the field offices, because they don't have so many "empty" pages to page through to get to their data.
The prepare directives in Blaise provide for a much more streamlined instrument, while continuing to use the same instrument for all quarters.

Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey

Robert Bumpstead, Office for National Statistics, UK

1. Introduction

The Continuous Population Survey (CPS) is an innovative project to integrate the five key government surveys on which the UK Office for National Statistics leads into a single modular survey system. The proposed new survey involves a comprehensive integration of the entire survey process: from the creation of a unified field force of interviewers administering a common modular questionnaire, to the processing and production of outputs from a single common source.

The proposal to integrate Office for National Statistics (ONS) household surveys builds on a successful and ongoing programme of harmonisation and instrument standardisation which began in the 1990s. This paper first sets the context for the CPS by exploring the drivers to further integration, and then the costs, benefits and challenges entailed in an integrated household survey.

The paper also describes the main features of the proposed survey and discusses the implications of these for building an integrated questionnaire and modular Blaise instrument. Development of the instrument itself is still at a relatively early stage, but issues already scoped by the project and examined here include: harmonisation and standardisation; modular structure; routing; dependent interviewing; concurrent interviewing; interview mode, and; use of audit trails to inform design.

This paper also considers aspects of the wider programme to modernise ONS statistical tools and systems of which the CPS is a fundamental part and is, in turn, dependent upon. The paper concludes with an overview of the next steps in the survey's development.

2. Context

2.1. Background

The widespread development and application of Computer Assisted Interviewing (CAI) over the last two decades is one factor which has prompted consideration of comprehensive survey integration. The existence of electronic questionnaires and the computerisation of other elements of the data collection process has removed some of the practical difficulties in designing and administering combined surveys.

Among National Statistical Institutes (NSIs), perhaps the most notable development was the creation of an integrated household survey, *POLS*, by Statistics Netherlands in the late 1990s. Proposals for integration were previously investigated in the UK around the same time. One result was the successful launch of a merged Expenditure and Food Survey (EFS), bringing improvements in quality, coherence, and efficiency by replacing two separate surveys with one integrated survey.

However, for a variety of reasons, no further major integration took place. Since then, a stronger base for integration of government household surveys has developed, together with the

availability of improved technologies with which to deliver it. In addition, the drivers for integration have significantly increased.

2.2. Drivers

At the heart of a proposed Continuous Population Survey is a recognition of the need for UK National Statistics to produce better information on key social and economic variables between decennial censuses, for a range of policy purposes, and to meet the increasing demand for regional and sub-regional information.

In common with many other NSIs, ONS has experienced a rapid growth in the demand for small area statistics in recent years. In the UK there has been particular pressure for information about income and ethnicity, but also a range of other outputs. Cross-government initiatives to improve public access to official statistics, and to improve the availability and precision of local and regional statistics, has added to this demand.

There is also increasing demand for a range of new surveys, including those under European Union (EU) regulations which Member States are obliged to fulfil. Furthermore, there is a need to maximise value from ONS continuous surveys and improve the coherence of National Statistics. These demands cannot be met within the current survey arrangements.

2.3. Existing survey arrangements

Summary descriptions of the five surveys to be integrated are provided in Annex A. They are:

- the Labour Force Survey (LFS)
- the Annual Population Survey (APS);
- the General Household Survey (GHS);
- the Expenditure and Food Survey (EFS), and;
- the National Statistics Omnibus Survey (OMN).

An ongoing programme of harmonisation of questions, concepts, and classifications has improved the coherence of survey outputs in recent years. Household surveys have increasingly adopted common standards across not only the ONS but also the wider UK Government Statistical Service (GSS). Standardisation of Blaise instruments among the ONS household surveys has established a range of common authoring practices from naming conventions to screen layouts. Harmonisation of questions and classifications has increased the overlap between survey content. There is now more consistency between the surveys, but scope for improvement remains.

For example, estimates of the same variables across the different surveys cannot be combined and, despite the use of common questions, small but statistically significant differences between those estimates sometimes occur. Different fieldforces of interviewers serve different surveys, with differences between training programmes, work patterns and practices. The existence and maintenance of separate fieldforces, together with separate Blaise instruments, and a range of different survey processes from sample selection to weighting and imputation, imply a duplication of effort and sub-optimal use of limited resource.

2.4. Objectives

A Continuous Population Survey aims to build on and improve standardisation. As a coherent, effective survey system the CPS will include a range of components which will, together, best describe society as a whole and the individuals within it. Moreover, ONS believes that only through the combined resource of an integrated survey can demands for improved reporting at national, regional and local levels be addressed.

The central objectives of the CPS are to:

- develop a world class modular survey system and accompanying modular Blaise instrument, better able to meet the information needs of the 21st century;
- provide more coherent, better quality information on which Government, stakeholders and the wider user community can base their decisions;
- increase the precision of statistical outputs at national, regional and sub-regional levels;
- create a range of new outputs, including inter-censal estimates of key socio-demographic variables at the sub-regional level;
- develop a survey system with the flexibility to accommodate other surveys at a later stage;

And, to achieve these objectives:

- while maintaining continuity of outputs and preserving the integrity of key time series;
- without the need for resource additional to the combined budgets of the component surveys, and;
- while delivering further efficiency savings from economies of scale and increased value of statistical outputs.

2.5. Current position

The development of an integrated household survey to replace the existing ONS-led continuous household surveys is at a relatively early stage. Following an initial broad scoping study and informal consultations with stakeholders, funding was agreed in 2003 for a development project to establish more detailed proposals. At this time the project was known as the Integrated Social Survey (ISS). Later, it was renamed the Continuous Population Survey to better reflect its purpose and outputs.

ONS is currently engaged in a round of formal user consultation which started with publication of consultation document at the end of July 2004. [1] Work on the overall modular design of the survey is ongoing, while an extensive programme of development and testing has been proposed. On satisfaction of a range of agreed quality measures, the CPS would proceed to a start date no earlier than January 2008.

[1] http://www.statistics.gov.uk/about/consultations/ons_consultations/downloads/CPS_Consultation.pdf

3. A Continuous Population Survey

3.1. Principal features

The proposed form of integration involves a single sample of addresses and a modular survey instrument comprising all the existing topics to meet the information needs currently met by the separate surveys.

A short core module will be administered to the whole sample, while different topic modules will be administered to selected parts of the sample. Interview combinations will be composed of both core and selected topic modules. Interviewers in an integrated field force will each be responsible for delivering all the required interview combinations in their area.

From these main survey features stem a series of potential benefits and risks. These make the integrated survey highly attractive to data users in terms of increased value of statistical outputs, but also present a series of challenges in terms of managing change and minimising its unwanted effects. These elements are central in informing the objectives and implementation of a modular questionnaire and Blaise instrument for the CPS.

3.2. Benefits

There are large potential gains in the precision of many estimates from an integrated survey without the need for additional cost. These result from a series of factors, including improved weighting and better local representation. Two principal factors are further discussed here - sample size and design. In addition, improvements in flexibility and coherence are also considered.

3.2.1. Sample size

The CPS sample will be, in principle, composed of the cumulative total of addresses sampled by the existing surveys. Based on the survey reference year 2003/04, the CPS would have an annual independent achieved sample of 270 thousand households and more than 500 thousand adults, making it the largest ever continuous survey to be conducted in Great Britain. A full breakdown by survey of the number of achieved interviews in the survey reference year 2003/04 is given in Annex A.

All outputs derived from the core module will benefit from gains in precision associated with the new, larger sample. In comparison with the current LFS family of outputs these gains will be modest. In comparison with outputs derived from the other surveys, they will be very large indeed.

3.2.2. Sample design

The CPS will adopt an unclustered design, similar to that used by the LFS in Great Britain. Currently, all other Government household surveys use clustered designs, where addresses in a sample are selected from particular areas grouped in small ‘clusters’. Traditionally, this was necessary for all but the very largest surveys for reasons of economy. Clustering reduces interviewer travel time and costs, and makes fieldwork practical and affordable. However, a clustered sample leads to less reliable estimates.

By combining all the survey samples and fieldwork into one overall design the CPS will, for the first time, deliver unclustered samples for all interview combinations. An unclustered design will lead to substantial precision gains for many of the outputs currently produced from the GHS, EFS and Omnibus surveys.

3.2.3. Coherence and flexibility

A single modular survey approach also aims to improve coherence by delivering a range of outputs from a single data source. It reduces the incidence of 'competing estimates' where different surveys sometimes produce different estimates for the same variable. The process of harmonisation would be furthered as core module questions will be, by definition, common across the survey.

A flexible modular design would enable the requirements for new surveys to be met more readily and cost-effectively than at present. The expense and development time associated with launching a new stand-alone national survey are considerable, adding a module to an integrated survey much less so. The CPS also provides the flexibility to plan topic modules on the basis of sample size for the level of precision required, rather than constrained by whichever existing survey vehicle happened to be available, as at present.

As well as developing an appropriate design for the planned 2008 launch, the new survey is being built with future change in mind. The expectation is that new modules will be added as policy needs change over time, and that requirements for some new surveys will be met through the CPS. In addition, sponsors of other surveys may opt to switch their existing continuous surveys to this vehicle. Each survey joining the CPS would contribute to the overall sample size and further boost the precision and analytical power of the survey data.

3.3. Risks

There are a number of risks to the development and successful implementation of a Continuous Population Survey. Principal areas referred to relate to aspects of quality and continuity.

3.3.1. Quality

It is essential to a cost-effective CPS that each interviewer is responsible for all the interviews in their area. All interviewers, including new interviewers, will need to cover a large range of topics; larger than many will have covered before. Rather than work on a single survey, or a different survey each month, interviewers will be required to cover all interview topics on a weekly basis. New e-based learning methodologies and techniques must be devised as part of a programme to develop an integrated field force, and be validated in field trials.

Equally, the proposed new fieldwork design and modular questionnaire will need to be tested in practice to ensure they are acceptable to respondents and interviewers, help optimise response rates, and minimise respondent burden. Field trials must demonstrate that outputs can be delivered within the timescales necessary and to the required quality.

3.3.2. Continuity

Ultimately, any decision to implement the CPS rests on a demonstrable ability to deliver the benefits anticipated while maintaining the integrity of key time series, for example the UK

headline rate of unemployment. [2] In some instances time-series data span decades. An unduly large or unexplained step change in estimates would be unacceptable. A large scale experiment will be conducted to run a model CPS alongside the existing surveys. Benchmarked outputs from the two sources will be published and compared, and subsequently used to inform the decision regarding the go-ahead of the CPS, further development, or discontinuation.

3.4 Instrument requirements

Good questionnaire and Blaise instrument design will be central to the success of the project. To realise the statistical benefits implied by the new sample design, the questionnaire must be modular in structure, flexible, and consist of all the topics necessary to maintain fulfilment of information needs.

At the same time, the questionnaire must minimise and simplify the requirements placed on the interviewer (and respondent) in as far as is possible. Given the increase in the diversity of interviewer workload, improvements in the usability of Blaise instruments are highly desirable. Equally, a reduction in the time necessary to conduct each interview is also a priority, together with any other innovations which might help maximise response rates.

These improvements need to be delivered while maintaining sufficient comparability between the old and new designs so that unacceptable discontinuities to key outputs do not arise.

Finally, an essential element of CPS instrument design involves 'future-proofing' the survey. The modular structure of the survey instrument must be designed so as to readily accommodate new topics, and ensure that a range of survey types and features can be incorporated within the CPS survey system. These include:

- interview combinations with cross-sectional, quarterly or annual panel designs;
- diary components and other self-completion elements;
- telephone or personal interviewing, and;
- provision for interviewing with all household members, or with individuals sub-sampled within a household, or combinations of both.

Meeting these various requirements within a single instrument are further discussed in the following section 4.

[2] The headline unemployment rate as defined by the International Labour Organisation is a key output of the quarterly Labour Force Survey.

4. A modular questionnaire and instrument design

The CPS questionnaire will be designed as a single modular survey instrument comprising:

- a **core module** administered to the whole sample providing information on key variables for all CPS households and persons;
- **topic modules** administered to parts of the sample providing information on variables for which sufficient precision to meet policy needs can be obtained from a portion of the CPS sample;
- a small number of viable **interview combinations** formed from combining core with selected topic modules so that all topic modules are covered.

The CPS will involve design of the new core module and a re-examination of the combinations of topic modules represented by the current surveys, with the aim of improving coherence in reporting and optimising the overall use of modules.

4.1. Core module

Core module questions will collect information on census-type variables and other key socio-demographic and labour market variables. For example, household composition, tenure, employment status and economic activity, educational attainment and income, ethnicity and national identity.

The core questionnaire needs to be relatively short and straightforward so that total interview length for core and topic modules remain viable. Indeed, it is essential to the success of the CPS that the core questionnaire does not become unduly burdensome.

There are advantages to fixing key core questions over a number of years to provide an uninterrupted time series to a high level of precision. However, continuous reporting is not essential for all outputs. To allow greater flexibility and scope for inclusion of a wider range of questions, it would be desirable to include some questions periodically. Therefore, it is proposed to sub-divide the core module into two parts:

- A **fixed core** module with questions normally included for at least a five year period (or longer);
- A small **rotating core** module with questions normally included once every three years. For example, where a question might be asked first in 2008, then in 2011, and so on.

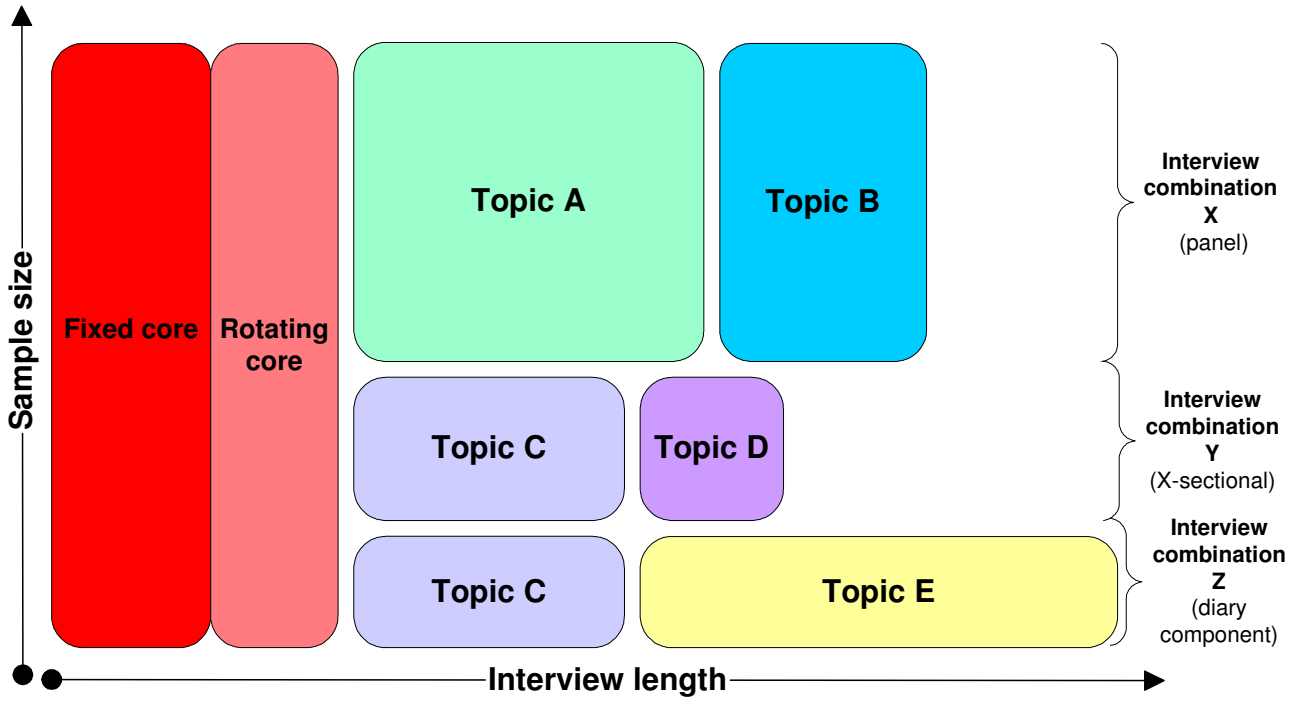
4.2. Topic modules

Questions in topic modules will collect detailed information on the various subjects covered by existing surveys where a continuing need for this information is identified. For example, health, education, labour market, income, and expenditure.

It is possible to run some modules across more than one interview combination to generate a sufficient sample size. It is also be possible to use the core questionnaire to identify rare or hard-to-find groups so that additional questions could be administered, or respondents followed-up at a later stage. A modular design would enable the quicker and more effective introduction of new modules, and amendment of existing modules, as customer output requirements change.

Figure 1 below shows a simplified abstract representation of how core and topic modules can be combined to form interview combinations.

Figure 1: Abstract representation of a modular survey instrument



A detailed illustrative diagram of a modular CPS questionnaire is contained in Annex B.

4.3. Interview mode

The CPS will be a mixed mode data collection instrument, combining elements of Computer Assisted Personal, Telephone, and Self Interview (CAPI, CATI, CASI).

Where an interviewer records an initial non-response, households may be 're-issued' to an alternative interview mode for a further attempt to secure an interview. For example, a field interview (CAPI) may be re-issued to the telephone unit where a telephone number can be obtained, or a telephone interview (CATI) re-issued to a fieldforce interviewer.

An integrated fieldforce, combined with new data handling systems, will improve the flexibility and speed with which re-issuing of interviews between modes can occur. [3] An efficient and systematic approach to mixed mode interviewing will help maximise response.

Different topic modules will have different rules for the use of personal, telephone, or self-interview - as with the current surveys. However, it is proposed that all questions in the core

[3] A proposed new Survey Case Management System (SCMS) is further discussed in section 5.

module should be suitable for both CAPI and CATI. Therefore, at a minimum, in the event of interview non-response in the field, the core module could be re-issued to the telephone unit.

This element of the survey demands an improved instrument design more sensitive to the particular mode of interview. Whereas current survey instruments are designed with a primary mode in mind, the CPS requires a mode-sensitive instrument that routes question type according to interview mode.

For example, in CAPI visual 'show cards' can be used to help respondents select an answer from a long list of options. In CATI the same information is best collected via a series of branched questions, each offering fewer answer options. Potential benefits include reductions in item non-response and respondent and interviewer burden, and, improvements in data quality.

4.4. Harmonisation and standardisation

All ONS computer assisted household survey instruments are written using Blaise, based on blocks sharing common standards. These are comprised of harmonised modules of questions (e.g. employment, ethnicity, education) and standard code (e.g. a block for the interviewer to record details about the administration of the interview, such as outcome and number of calls). Survey researchers also play the role of instrument developers, building their questionnaires using these standard blocks, together with blocks of survey specific questions.

These standard blocks are revised on an annual basis and are ready for inclusion on questionnaires at the beginning of the survey year, although essential minor changes may be made throughout the year. Instruments also conform to other common standards (e.g. screen design and layout) and are authored to common programming standards (e.g. naming conventions, datamodel structure). The ONS Blaise Development and Support Team (BDSS) monitors application of these principles.

This structure provides an excellent basis for full survey integration which may be seen, itself, as a logical extension of the harmonisation project. Indeed, without a preceding programme of harmonisation and standardisation, any move to any integrated survey would involve a step change of such magnitude that large discontinuities in outputs would be inevitable.

An important aspect of CPS development will be to take this process a stage further. Differences between survey specific questions can be minimised and, where duplication exists across surveys, blocks and questions will be removed. All blocks within the survey will, in essence, become standard. So for example, where it is necessary to collect detailed income information for different purposes in two interview combinations, the two income blocks will be developed to maximise similarity. Reducing the number of different questions (and rules of application) across the CPS will also reduce the interviewer learning burden.

4.5. Datamodel structure and routing

The CPS instrument is being developed as single datamodel, with elements of the existing survey instruments grouped together under an additional level of hierarchy. This approach minimises disruption to established survey specific naming conventions, but amalgamates those elements of blocks where a single survey approach is sufficient (e.g. top-level blocks for defining answer types, auxiliary variables, local variables, coding libraries etc.).

The routing structure currently being developed involves a linear progression through the instrument, in the sense that the entirety of the core module precedes sequential topic specific modules for an individual interview. This approach eliminates differential order effects in the core module, as core questions are always asked in the same sequence regardless of the interview combination which follows. It also provides a good working basis for topic specific researchers to develop blocks of code which can be inserted into the overall modular structure, overseen by a central design authority with responsibility for the instrument as a whole.

4.6. Audit trails

In recent years ONS has made greater use of the Blaise Audit Trail facility. Using the ATLAS and WesAudit tools, provided by Statistics Canada and Westat respectively, developers have been better able to analyse the vast array of data that an audit trail produces.[4]

The CPS will collect audit data in field trials planned for 2005 and 2006 to produce accurate timing information for individual questions, question modules and interview combinations. This will help inform design and optimise interview length across the different interview combinations. However, audit data from the existing surveys already provides CPS developers with useful information. By adding block and question times together from the current surveys to resemble the intended interview combinations in the new survey, good assessments of likely interview length can be made before the CPS instrument even enters the field. This helps to improve and accelerate the design process itself.

Audit trail data will also be used to validate the success of the CPS in reducing interview length compared with the current surveys. It could also be used to assess other aspects of the new survey's design. For example, the extent to which interviewers make use of extended context sensitive help functions or greater flexibility to use concurrent interviewing (see 4.7 below).

4.7. Other aspects of instrument design

Other on-going aspects of CPS instrument design include the standardisation and extension of dependent interviewing, concurrent interviewing and use of context sensitive help.

Dependent interviewing occurs when questions are asked which are dependent on a respondent's answer from a previous interview. The CPS will include at least three interview combinations which involve panel designs and elements of dependent interviewing. A standardised approach to designing and administering these questions, together with the processes used to 'feed forward' data from a previous interview is, therefore, required for the CPS.

The CPS will also incorporate wider and more flexible application of concurrent interviewing. This permits an interviewer, for example, to ask the same question of all household members in turn as an alternative to asking the whole questionnaire from start to finish for each respondent. Where appropriate, allowing individuals in households to be interviewed consecutively or concurrently (or in combinations of both) across all modules will improve standardisation and operational flexibility. It could also help to reduce interviewing time and respondent burden, particularly where information is collected from a single proxy respondent on behalf of all the household members.

[4] We gratefully acknowledge the help of colleagues at Westat and Statistics Canada

Finally, the CPS instrument proposes to expand the use of context sensitive help. This occurs where context sensitive information is made available in the Blaise questionnaire to the interviewer (or respondent) to deal with particular problems or queries as they arise during the interview. Benefits include improvements in data quality, reduced training costs, and reduction in paper instructions and documentation.

All changes in questionnaire design will be thoroughly tested and validated as part of the development programme outlined in section 5.

5. An evolving social survey system

5.1. CPS and ONS statistical modernisation

Delivery of the CPS is dependent on the development of a modular questionnaire and single Blaise instrument. However the CPS should also be understood in the wider context of the ongoing modernisation of ONS statistical tools and computing infrastructure. The CPS is dependent on the standard tools and systems that this programme will deliver.

The transition from a complex portfolio of surveys to a streamlined household survey system is illustrated in the diagrams below, where process 1 might represent the organisation and training of interviewers, process 2 the design and administration of questionnaires, and process 3 the editing, imputation and weighting of data.

Figure 2 provides an illustration of the extensive system of separate, but related, processes involved in each survey.

Figure 2 : Process model of existing surveys

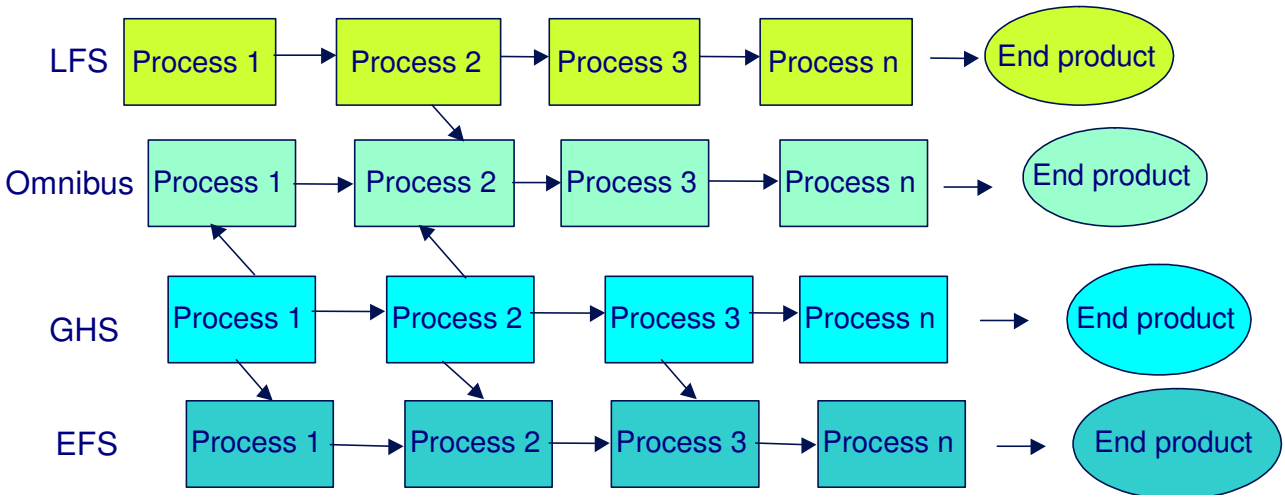
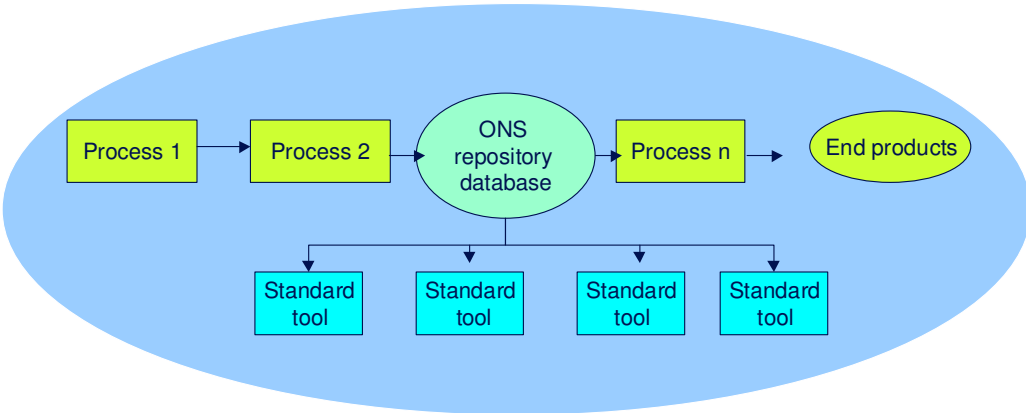


Figure 3 illustrates how the above model could be replaced with a single and efficient production chain. Data are derived from one questionnaire administered by a single field force, and stored in a Common ONS Repository Database (CORD). Processes like editing, imputation and weighting

are carried out using common tools to consistent standards. A range of end products are still derived, but from a common source.

Figure 3 : Process model for Continuous Population Survey



The CPS will depend on the delivery of these standard tools and the central database. In addition, it requires the delivery of a new Survey Case Management System (SCMS) to handle the flow of survey information and manage field operations more effectively.

5.2. Survey Case Management System (SCMS)

The new SCMS will support the modernisation of fieldwork operations that are necessary to provide successful data collection for the CPS. It will replace a range of older tools, with the aim of providing a more responsive and integrated system, together with improved information about data collection.

The SCMS will start with the assigned survey sample and cover every step in the process of information handling up to delivery of data to the central database, CORD. It will deal with allocation and re-allocation of addresses to all interviewers across all modes of data collection, and will manage the consolidation of disparate sources of data post-collection. It will also provide new systems to support the management of interviewers and, possibly and most significantly, real-time case management information. The design of the system will also ease the adoption of newer technologies, particularly remote connectivity, as they become available. Initially, the system will support household surveys but will be later expanded to support business surveys and the 2011 census too.

5.3. Development programme

The CPS development programme is designed to bring all these elements together to produce a comprehensive redesign of UK government survey taking and new a social survey system.

Currently, work is ongoing to consider a range of methodological issues regarding question development and data collection. Combinations of qualitative and quantitative techniques are being employed to investigate new questions, assess output quality and comparability, and, inform instrument design. A prototype Blaise instrument will be produced before the end of 2004.

A series of trials in 2005 and 2006 will test, develop, and validate all aspects of the proposed survey. These will begin with a small scale feasibility study involving a few hundred interviews,

and culminate in a large ‘parallel run’ involving many thousands. This trial will be central in taking the decision to go ahead with a CPS and discontinue the component surveys. For the CPS to proceed in January 2008 this decision can be taken no later than April 2007.

Annex A : Surveys for integration

The General Household Survey (GHS), the first UK multi-purpose household survey, started in 1971 and covers a wide range of social and socio-economic topics. The main aim of the survey is to collect data on core topics including housing, employment, education, health and family information.

The Expenditure and Food Survey (EFS) started in 2001 bringing together two surveys, the Family Expenditure Survey (FES) and National Food Survey (NFS), that were both well established and important sources of information, charting changes and patterns in Britain's spending and food consumption since the 1950s.

A Labour Force Survey (LFS) has been carried out in the UK since 1973 and in its present form since Spring 1992, providing a wide range of data on labour-market statistics and related topics such as training, qualifications, income and disability. In recent years the quarterly LFS has been supplemented by a series of annual boost samples in first England, then Wales and Scotland, known collectively as the Annual Local Area Labour Force Survey (ALALFS).

The Annual Population Survey (APS) is a boost sample in England which, when combined with results from the Labour Force Survey and LFS boost samples, will provide better local authority district estimates for key social and socio-economic variables than is possible from existing survey sources. The APS started in January 2004.

The National Statistics Omnibus Survey (OMN) is a regular, multi-purpose survey that started up in 1990 in order to provide quick answers to questions of immediate interest and information on topics that do not require a full, in-depth survey.

All the surveys attempt to conduct interviews with every household member, with the exception of the Omnibus survey where interviews are administered to one randomly selected household member only. The surveys differ in their use of Computer Assisted Personal and Telephone Interview (CAPI and CATI) and the extent to which proxy responses are permitted.

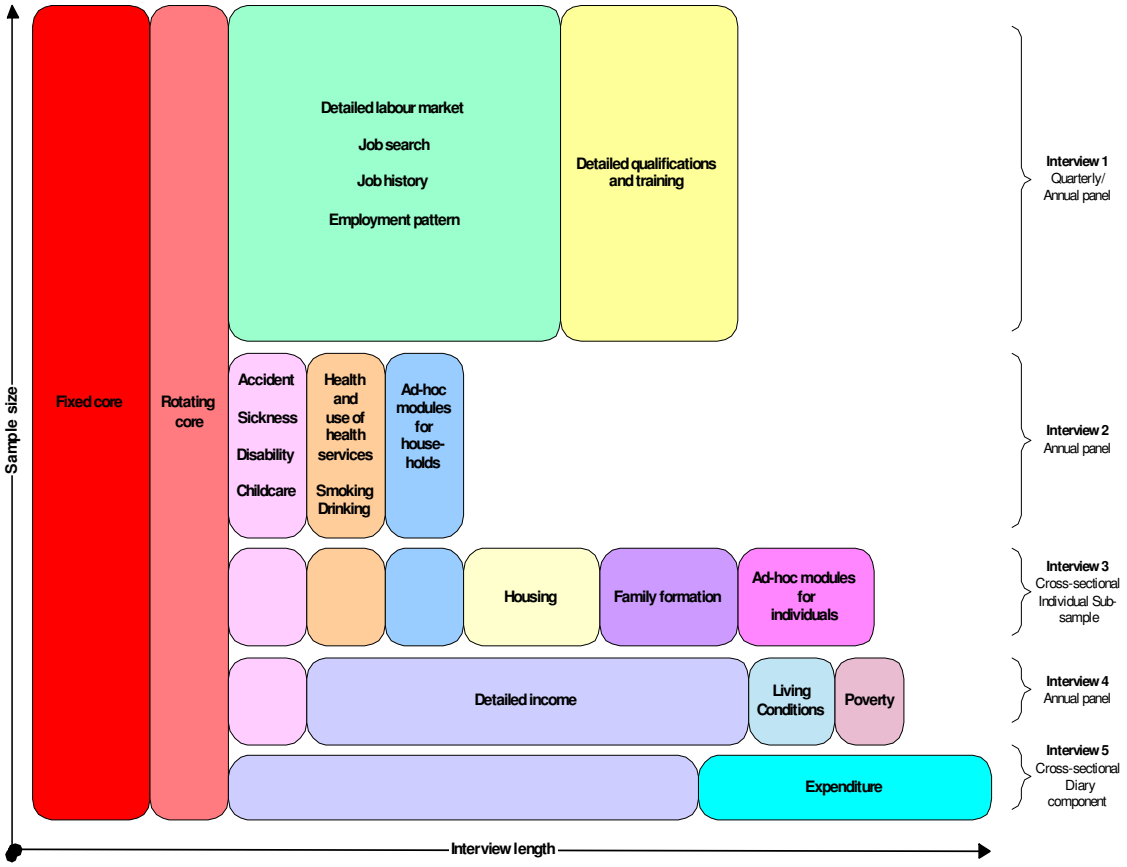
A breakdown by survey of the number of achieved interviews in the survey reference year 2003/04 is shown in Table 1 below.

Table 1 : Number of households interviewed in 2003/04

	thousands
Labour Force Survey	**86.5**
Annual Local Area Labour Force Survey Boosts	**87.0**
Annual Population Survey Boost [1]	**66.4**
Expenditure and Food Survey	**6.4**
General Household Survey	**8.6**
Omnibus Survey [2]	**14.0**
All	268.9

[1] Estimated [2] Currently one selected adult per household

Annex B : Illustrative diagram of a modular CPS



Replaying Blaise Audit Trail Files for Data Verification

Jason Langfahl, Survey Research Organization Technical Services Group, Institute for Social Research, University of Michigan

1. Introduction

In survey research, a lot of effort goes into ensuring the quality of the data that is gathered. After all, it is the data that researchers use and refer to when writing up papers for grants, proposals, theories, and a host of other technical documents for their fields. If the data is of poor quality, then the foundation on which their work is based is also of poor quality by extension.

This paper will concern itself with a quality assurance utility that was devised to take an audit trail file that is generated by an interview and using it to "play back" a case on an in-house reviewer's computer while listening to a tape. The goal is to approximate what the interviewer sees while entering the data. This allows the reviewer to assess the quality of the entered data and the interviewer's performance, especially in areas of adherence to standardized interviewing techniques (for example, correctly handling probes or inputting data into an open-ended comment as relayed by the respondent verbatim) and accuracy in entering data.

2. Why Playback?

One concern that comes from data gathered by surveys is whether the interviewer is conducting interviews correctly. It does no good to have data gathered that was erroneously inputted by an interviewer during the course of an interview. So, something is needed to be able to check data that has been gathered and verify its soundness.

A standard method that has been used to achieve this is "read back". Essentially, cases sent into the central data-gathering location (in our case, Ann Arbor, Michigan) are re-opened in a Blaise data-entry session by a reviewer. The reviewer then examines the case compared to any information on hand, typically a tape recording of the interview, to determine the accuracy of the case and evaluate how the interviewer handles situations that arise during the course of the interview.

The Health and Retirement Study (HRS), however, could not use this method of evaluation because the logic employed in the Blaise instrument itself removed question fields from the route upon completion of certain portions of the interview. This sort of protection keeps interviewers from going back to change data that might impact future questions in the course of the interview, but it also means that once the case returns to Ann Arbor for evaluation, those questions stay unavailable and cannot be evaluated. In HRS's case, this would be unacceptable because so many portions of the interview are blocked off in this manner. Another way had to be devised to see the data as the interviewer had inputted it.

The result is "Playback," a tool that was developed for the purpose of taking audit trail files generated by Blaise and using the data in those files to re-create the case as the interviewer saw it on screen.

3. History

The origins of Playback go back to the prior wave of HRS in 2002. As noted before, HRS has special portions of its interviewing process whereby questions get "locked away" after completion, preventing any chance of changing data once those questions are answered, thus negating any benefit of just opening up the case line in a Blaise data-entry session and reviewing the data, since so much of it would not be visible.

Enter Playback. The idea was to utilize the audit trail file generated by a case to "play back" a case based on the keystrokes entered during the course of the interview. Since the case being entered was "untouched" as it were, there were no sections that were locked away from view since no data had been entered yet. Playback would parse the audit trail file and send keystroke information to the data entry session to mimic the interviewer inputting information during the course of the interview.

The initial version that was developed for HRS 2002 was only partially successful, however. It encountered numerous problems ranging from being unable to share preload between respondents as the original interview did, thus precluding an entire group of people from ever getting reviewed, to simple Blaise mechanics that weren't fully programmed for. In addition, the 2002 version, while intended for HRS initially, was expected to be a general-purpose tool eventually, so there were complicated interfaces developed to try to meet this need that hampered the review process.

For 2004, case reviewers still needed a tool to handle HRS. Other methods, such as digitally recording interviews were considered, but ultimately the decision was made to improve Playback.

The most significant improvement didn't actually have to be done to Playback itself. As mentioned before, a significant set of interviews, those pertaining to the second respondent in a two-respondent household, could not be reviewed at all in 2002 because the preload that Playback tried to run with was always from a first-respondent perspective. Any data that the first respondent gave that might impact how the interview was to be handled for the second respondent was lost because there was no repository that contained these changes. To overcome this problem for the 2004 wave, it was agreed that programming was to be added to SurveyTrak (our in-house field survey management system) that preserved the preload string for each case. With preload preservation, the retained preload string could be fed into an ASCII-to-Blaise Manipula as it is done in the field and the preloaded database would be created. This allowed second respondents to be reviewed on-screen for the first time.

Another significant change came with the agreement that when cases were completed in the field and returned to Ann Arbor, the ADK files would be kept in separate folders identified by the date of the datamodel that recorded them. Previously, in all other Blaise-developed studies by ISR, the audit trail files were all kept in the same folder for review. By breaking them down by datamodel (and extending this breakdown to the new preload string files being retained) it allowed Playback the luxury of knowing which datamodel to use for review without having to resort to going through ADK files looking for the correct one.

In addition to retaining the preload string and breaking down the ADK and preload files by datamodel, there were several other changes made to streamline and improve the program. The resulting program is at present a simple program that caters to the needs of reviewers who are looking at cases completed in HRS.

It is this present version of the program that will be discussed in detail.

4. Development Environment

Playback had to be designed to not only provide the functionality of mimicking the interviewing process, but it also had to present it to users in a relatively basic, easy-to-use fashion. The target users for this utility were in-house reviewers. These reviewers, while generally proficient with the Windows operating system, are not as technically skilled as computer programmers, so the program has to be something that they can grasp quickly and easily, without having to guess what this symbol or that button means.

Visual Basic 6, with its ability to develop windowing programs quickly and easily and its ability to parse text files without much programming complexity, was the development environment of choice. There is also a ready-made function, "SendKeys" that is the primary function used to send keystroke information from the Playback program to the DEP session, eliminating any need to program a new such function for our purposes. When this paper describes the Playback tool as "sending keystrokes" or "inputting keystrokes" or other similar phrases, this function is the primary mechanism for doing it.

5. Implementation

This section will detail step-by-step how Playback is designed graphically and programmatically to play back a case based on audit trail output.

5.1 – Selecting a Case to Review

Playback was designed for use by reviewers to look at how cases were being handled in the field. A typical reviewer isn't as technically proficient with computers as a programmer is, so the interfaces needed to be clean, simple, and obvious to avoid confusion.

In the original 2002 version, this wasn't necessarily so. A reviewer was at first confronted with a large window that requested they fill out several text boxes that identified various command line parameters like menu files, configuration files, and so forth. Needless to say, this sort of thing wasn't very easy to use for those not familiar with all the things that go in to running a Blaise data-entry session.

With this version being written specifically to handle HRS cases, it also gave a chance to redress some of these interface issues. Since the project of choice is hardwired into the program, there was no need anymore to have a large, complicated screen to fill out in preparation of lauching the DEP session for case reviews. Instead, all that was necessary to know was what sample ID to review. The new starting interface is shown in Figure 1.

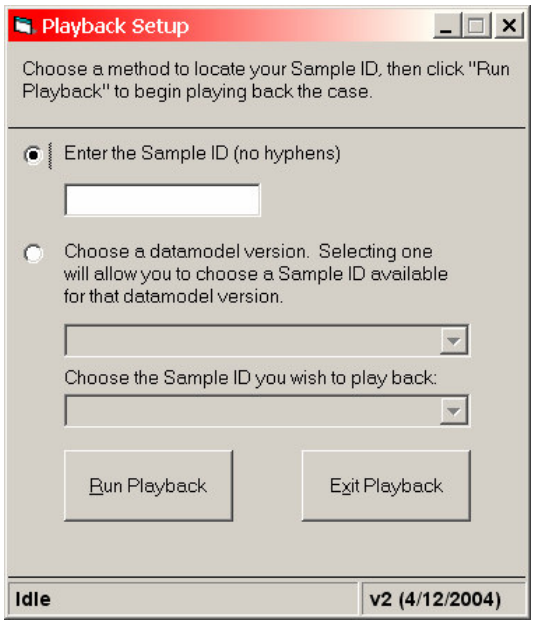


Figure 1 – Case selection interface.

In the new interface, the reviewer needs only to supply a sample ID (usually written on the tape he/she is about to listen to). Optionally, the reviewer can choose a datamodel and then browse the sample IDs available for that datamodel version to find the one he/she wants to review.

With the case now selected for review, the reviewer clicks "Run Playback" to begin the actual process of playing back a case.

At this point, Playback goes out to a network location where all the necessary files (ADK, preload, datamodel files) are stored and copies them to the reviewer's local machine. (NOTE: this network data is itself a copy of the actual data that our data operations staff uses to create reports to HRS staff, so there is no actual involvement of source files at any point in this reviewing process, just copies.)

With all files now brought down to the local machine, Playback's next step is to parse the audit trail file.

5.2 – Audit Trail Parsing

The most necessary component for Playback to work is the audit trail file, or ADK, as we call it. This file is generated by an in-house version of the auditkey.dll that Blaise supplies. A normal ADK file that is captured in the field looks something like figure 2.

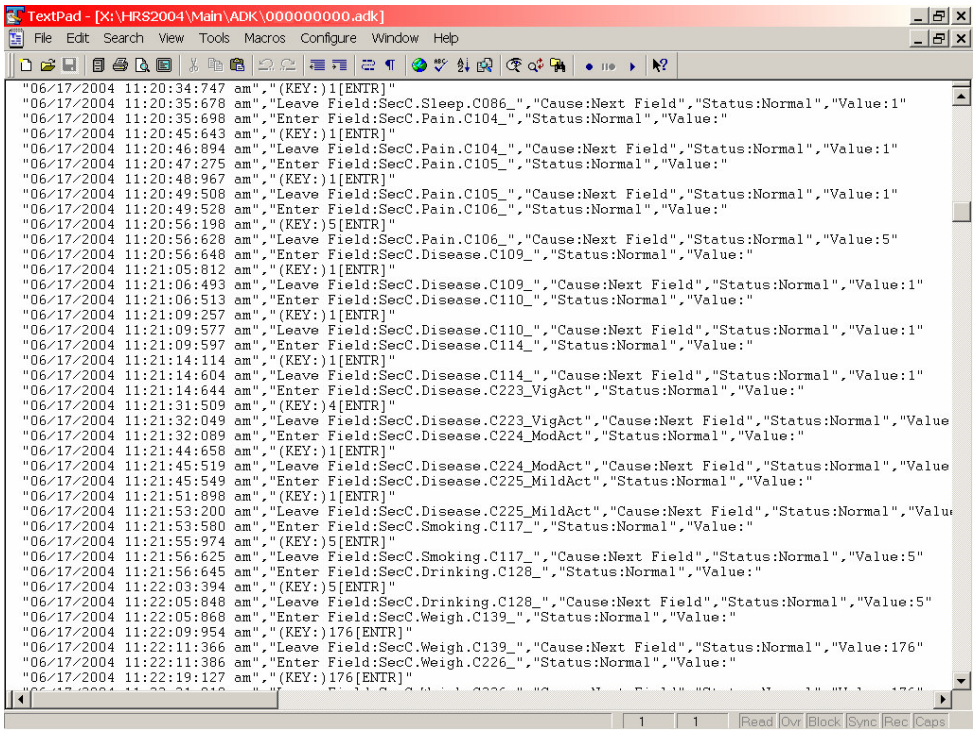


Figure 2 – An example of an ADK file.

A lot of information is gathered in these ADK files, and Playback must sift through the information to find what it needs to replicate the case. Chiefly, it parses through this file looking for references to "Enter Field" and "Leave Field". These references tell Playback which field data is being inputted into. More importantly, in between each set of "Enter" and "Leave" lines that say which field the interview is recording in is the line that states "(KEY:)". The data that appears after these "KEY" references are the actual keystrokes that are inputted into the interview.

Playback goes through the ADK looking for these markers and in turn stores these in an ADODB record set, which has two fields. One field contains the name of the field, the other the keystrokes as recorded. It should be noted that normal alphanumeric characters are identified separately from function keys like "Enter" or the space bar. Any string of alphanumeric characters makes up one record in the recordset. A function key such as "Enter" gets its own record in the record set. This enables Playback to accurately mimic the interviewer hitting things like "Backspace" and the reviewer is able to see that reflected on the screen, rather than having things occur in a blur if all the keystrokes for a field were played out at once. Conceptually, then, this is what Playback breaks down an ADK to:

FIELD	KEYSTROKES
Sec_C.Pain.C104_	1
Sec_C.Pain.C104_	[ENTR]
Sec_C.Pain.C105_	1
Sec_C.Pain.C105_	[ENTR]
Sec_C.Pain.C106_	5
Sec_C.Pain.C106_	[ENTR]

However, since Playback is sending keystrokes via Visual Basic to the DEP, the program must change function keys into something that can be understood by the SendKeys function in Visual

Basic. So, Playback also has been programmed to change "[ENTR]" as given by the ADK file to "{ENTER}" as SendKeys could understand and handle. This sort of manipulation is especially needed in case the interviewer has to put in a DK or RF response to a question. These combination keys shows up as "[ALT]d" or "[ALT]r" in the keystroke file. So, Playback has to infer that when [ALT] is pressed and it is followed by a "d" or "r" that it must mean the interviewer pressed both together at the same time, since combination keystrokes are not actually noted in the ADK file. As a result, Playback replaces these "ALT" keystrokes with the appropriate "^%" code that Visual Basic uses to represent a combination keystroke using the ALT key.

There are several other manipulations of the data that comes from the ADK file that allow for Playback to do its job better. We won't go into details on all of them, but suffice to say a lot of conversion between ADK output and Visual Basic input occurs.

5.3 – The Playback of the Case

Once the ADK has been parsed, Playback then takes the preload.asc file it has copied and runs it through a Manipula ASCII-to-Blaise script that creates a preloaded database for a starting point to the case. With that accomplished the program then calls the DEP session into existence. A screen similar to the one in figure 3 appears for the reviewer:

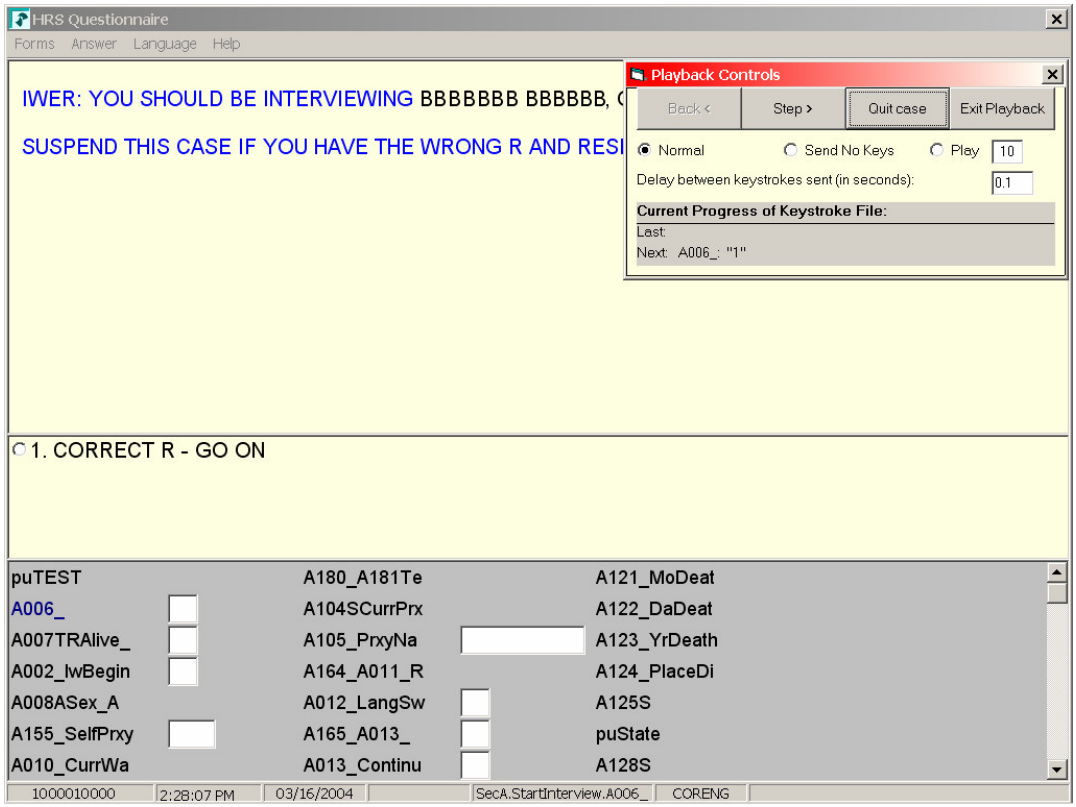


Figure 3 – Playback of a case at startup.

In the upper right is the actual control that manipulates the playback of the case. This interface has also gone under a significant change since the 2002 wave. In that version, this control consisted of VCR-like buttons for "play", "stop", and "pause" (pause was necessary because the original version allowed for a continuous play of keystrokes). In addition there were up and down arrows and a stop sign labeled "EXIT". The interface was somewhat confusing to use

because it wasn't immediately clear whether "Stop" or "EXIT" in a stop sign did the same thing or completely different things. In addition, with something like continuous play, a reviewer would have to constantly pause the playback and resume it to keep in time with what he/she was hearing on the tape. There was no mechanism to back up if something should go wrong with the playback of the case, which can and did happen frequently as Playback wasn't robust enough to handle all the little things that go on in a Blaise instrument, like out-of-range boxes. The only information the reviewer had about where he/she was in the audit trail of the case was a couple of lines describing the last field that had a keystroke inputted and which keystroke it was.

With the new version, the interface was re-worked to make it simpler and more informative for the case reviewer. Gone are the VCR buttons. The option for continous play has been removed. Instead Playback operates on a step-by-step process, going one field at a time, at the pace the reviewer wants based on the case being listened to. Figure 4 shows this new interface.

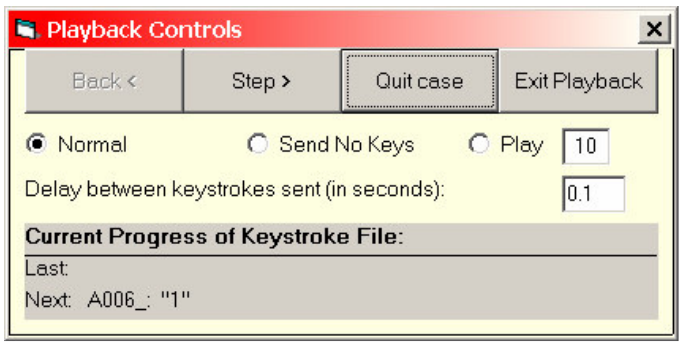


Figure 4 – The Playback control interface.

There are several parts to the Playback control interface.
*Back: This allows the reviewer to step backwards through the keystroke file. Do note that this does not alter the present state of the data entry session. The cursor in the DEP window will remain at the field the reviewer was at before pressing the back button. It does not attempt to "undo" the keystrokes that were played in the course of stepping backwards. (It would be difficult to reverse keystrokes, especially ones related to remark files, since the keystroke that opens one would occur before any text put into a remark.)

To resume Playback properly after using the Back button, move the cursor in the data entry session to the field listed in the "Next" portion of the "Current Progress of Keystroke File" part of the window.

*Step: This is the driving portion of the program. Each time "Step" is clicked to send keystrokes to the DEP session the program consults where it is in the record set, plays the keystrokes for the current record, and moves to the next one. If that record has the same field name as the prior one, it goes ahead and plays the keystrokes associated with the record it is now currently looking at. It continues in this process until it reaches a record where the prior record has a field name different from the current record. This allows Playback to put in all the keystrokes for a field without having the reviewer spend the time to click through each portion of the keystrokes for a given field. In addition, for remark files, Playback will delay closing the remark box for a number of seconds to allow the reviewer to read what was inputted. If it isn't enough time, the reviewer may return the cursor to that field and open the remark again. Once done the reviewer can move the cursor back to the field listed in the "Next" area of the playback control to resume normal playback. Playback also recognizes when checks or signals occur in the keystroke file and can close the box if the error was suppressed or exited or will attempt to display to the reviewer where

the instrument moved to if the interviewer elected to correct a mistake by moving to a different field.

Beneath the row of playback buttons, the reviewer can further control what the "Step" button does. The reviewer can do one of three things: (A) Normal, which means the program will send all of the keystrokes for one field at one time and then stop again, (B) Send No Keys, which essentially is like Back, only the reviewer is now stepping forward through the keystroke file without changing the data entry session's state, or (C) Play, which has a box next to it in which the user can put in a number of fields Playback should attempt to go through each time Step is clicked. In addition, there is an option below these choices to determine how quickly or slowly the user wants keystrokes fed to the data entry session, the larger the value, the slower the progression through the individual keystrokes. This allows the reviewer better ability to track what's being inputted for each field, but results in greater amounts of time spent reviewing the case.

*Quit Case: This option allows the reviewer to terminate the case being reviewed and return to the startup screen to select another case to review.

*Exit Playback: The reviewer can click this button to terminate the application entirely without returning to the startup screen.

At the bottom of the control window is displayed the current position of Playback within the set of keystrokes that make up the case. "Last" lists the last field that received a keystroke and the keystroke that was sent to the data entry session. "Next" normally lists the field the data entry session is found in and the upcoming keystroke ready to be sent to the session. This portion of the control acts as a check for the user to make certain that the keystroke file is in sync with the data entry session, since there is no programmed logic in the program that controls this aspect.

Once a case has been completed or the user terminates the control by clicking on the "Quit Case" or "Exit Playback" buttons, Playback will attempt to do garbage collection by deleting all the files it copied over from the network location, thus removing any personal data from the reviewer's machine after completion of case review.

6. Issues

Playback as it stands today is a solid utility that helps reviewers get through cases that they could not look at otherwise. This is not to say there are no flaws with the program, however. Playback still has many limitations. Chief among them is the fact that Playback is an unintelligent system. That is to say there is no way for the Playback control to understand what is going on in the DEP session. If, for example, an out-of-range error occurred in the DEP session, the Playback control has no awareness of it since audit trail files do not record interviewer actions in these external boxes. The reviewer must manually close this error box before continuing. This sort of bump is especially problematic for instances of continuous play, since the Playback program will continue to feed keystrokes to the DEP even if the DEP is no longer the focus because of the out-of-range error box. The only way to correct such a mistake is to use the "Back" button to move the keystroke file record back to the field that generated the error and then resume playback from the point of the correction.

Another point of weakness is the handling of alien routers. In HRS these routers produce word lists and state lists similar to lookups in Blaise. However, since these routers launch windows that exist outside the DEP, any keystrokes are not recorded and the ADK file actually has no "Leave Field" line denoting the interview moving on to the next field. This caused serious

problems for the parser. A workaround was made by having the parser recognize the specific fields that have alien router lookups (hardcoded) and adding the necessary information to the ADK file to put a DK response into that field, since normal keystrokes wouldn't work in those fields. However, the problem with this sort of solution is if the instrument's route depends on an answer given in the alien router call, it will throw the program off-track without any way to recover since the reviewer cannot input the data needed to send it down the correct path.

An issue that only comes up rarely, yet is significant is if the interviewer suspends a case. At present, Playback isn't programmed to resume a suspended case at all. Furthermore, there isn't any consideration for if the interviewer is issued a new datamodel and the case is therefore migrated to the new version. In that instance, more code would have to be programmed to handle a migration and resumption. None of this is there now.

Aside from those issues (and, of course, the occasional bug), Playback has been able to satisfactorily meet the needs of the reviewers involved in looking at HRS cases.

7. The Future

At the present time, Playback is geared specifically to handle HRS cases only. It was necessary due to the short development time available to get the product ready for cases coming in for the 2004 wave of HRS. However, if time and resources are made available, it is probable that this program will be rebuilt from the ground up into a general-purpose reviewing program that will handle a broad range of projects.

8. Conclusion

In conclusion, Playback has largely been able to meet the demands placed on it by those who use it most, the reviewers. The advances the program has made since the 2002 wave of HRS have been able to increase considerably the number of cases that reviewers can review and have been able to review. There are limitations and places for improvement, of course, but it has largely been a successful utility.

Replaying ADK Files for Testing Blaise Applications

Jason Ostergren and Rhonda Ash, Health and Retirement Study, Institute for Social Research, University of Michigan

1. INTRODUCTION

The Health and Retirement Study (HRS) is a major national longitudinal survey on the health and economics of aging and retirement. It consists of a complex multi-sectional hour long interview of 22,000+ participants and is collected every two years using CAI (computer aided interviewing) technology. It utilizes preload of more than 350 variables from previous wave data, and incorporates seven mode and language combinations. The scope of the HRS instrument exacerbates the common difficulties inherent in accurately programming and testing a CAI application.

This paper will cover the conceptualization, implementation and practical results of a program which replays audit trail files with the goal of increasing efficiency and accuracy of instrument programming and testing on a project of this scale.

2. PREMISE

Testing a CAI instrument presents a host of complexities. The desire to find a means of reducing the burden of testing and a means of automating parts of the process is apparently quite widespread.[5] For some time HRS had been interested in having an application at its disposal that would allow a tester to retrace the path through an interview as recorded in an audit trail. During the transition to Blaise from another survey language in 2002, HRS began to work on the concept for just such an application.

Much of the specification, programming and initial testing of a survey instrument falls outside the purview of this program. The application is not intended to supersede Q by Q and scenario testing, but rather to improve the process of fixing and retesting errors that have already been found and to reduce the likelihood of introducing new errors in the process. Additionally, HRS requires a way to routinize certain testing activities where a particular scenario or path through the instrument is repeated numerous times, possibly by different people. The different needs and types of testing that are addressed by this program are described below.

Multitask testing is difficult, at best. It is nearly impossible for a tester to focus on several detailed aspects of a large application, such as skip patterns, codeframe values, screen format and colors, and QxQ help, all at once. However, a single set of paths through an application, as recorded in the audit trail files, will allow testers who specialize in different areas of testing to replay those same paths, focusing, in the process, on their designated area of concern. As an example, HRS has testers that have experience with particular kinds of content that are often focused in a small set of sections. They know the expected population that should flow through their sections and how to recognize problems when testing scenarios. Audit trail files (or "keystroke files" for shorthand) can be captured from these specialists' testing activities. These keystroke files can then be reused or used by additional testers to review other areas of interest,

[5] Tarnai, John and Moore, Danna L., Methods for Testing and Evaluating Computer-Assisted Questionnaires, *Methods for Testing and Evaluating Survey Questionnaires*, Ed. By Stanley Presser, et al., New Jersey: John Wiley & Sons, 2004: pp. 319-335.

such as screen formatting. Further, minor changes can be made during keystroke replay to increase the utility of the process for these testers. For instance, a language specialist may wish to change the language keystroke at the beginning of the application and replay the rest of the audit trail, reviewing the text and codeframes in a different language than was originally recorded.

The complexity and length of a survey are major factors in the amount of time required for testing. In a large survey, programmers and testers typically expend enormous amounts of time repeatedly keying their way to a critical testing juncture. HRS sees a need to be able to replay a set of keystrokes to get to a particular point in an interview for situations in which a programmer or tester needs to repeatedly work in a given area of the instrument. Moreover, answers to a section in the middle of the application could impact the flow and/or fills in sections at the end, so it may be necessary to test repetitively through large parts of the instrument, stopping to alter a particular set of answers at a given point. A great deal of time can be saved by having the ability to reproduce part of an interview or testing path, change a few answers and continue to replay keystrokes up to the area of key interest for testing. Further, this functionality can be used more generally to replay a keystroke file and to diverge from it at a certain point to test other routes.

The most important function of this application is to aid in reproducing errors and retesting fixes. This is needed for both the development phase and during the field period. During development, audit trail files can be gathered from testers who report errors. Replaying those audit trails takes the programmer who is investigating the error directly to the problem, offering an advantage in reproducing problems that is fairly unique to the field of survey programming. Additionally, the programmer can readily replay the keystrokes to the point of the error to test and to confirm the validity of any fixes for the problem. Once in the field, this allows the replay of cases for which an interviewer reported a problem. By reproducing the interview through this process, the staff can diagnose, fix, re-test and document the problem rapidly. This can aid the correction of problems while still in the field and assist in the documentation of the data in cases that have already been completed.

The continual changes to programming specifications that HRS commonly experiences during development, even after the programming of a section has been completed, often affect other sections beyond the one being altered. Once a section has been fully tested, the ability to preserve a set of keystrokes reflecting the functioning section is important. It is then possible to replay them at a later date to identify any errors that were introduced by alterations to other sections. This can also allow for comparing data between runs to quickly pick out any differences.

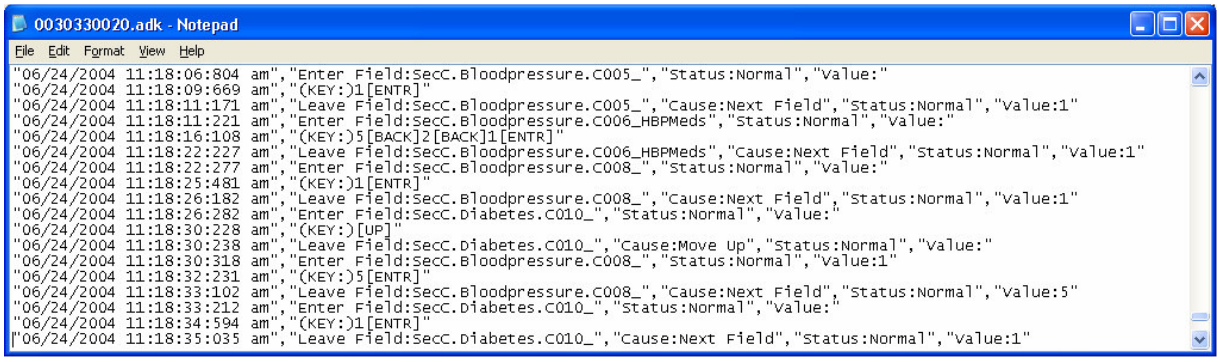
HRS endeavored to meet the need for a utility to reduce testing difficulties by conceiving of and building a utility called “Replayer.” This program makes use of the keystroke files produced as a byproduct of Blaise tests and interviews to allow the rapid playback of common scenarios, reports of problems by testers (the reporting system saves keystroke files for this purpose), and programmer fixes for verification. The “Replayer” needs to allow for “fast forwarding” to a chosen point in a given interview, as well as for playback and review of individual keystrokes, and for switching between the “Replayer” program and the interview in the Blaise Data Entry Program (DEP) to manually adjust values when needed.

3. IMPLEMENTATION

There were a few significant hurdles to overcome in order to build an effective program to meet the aforementioned needs (hereafter referred to as the "Replayer" program). Discussion will include the key problems involved in programming such a utility using BCP and Visual Basic 6. The most obvious of these is how to make effective use of the keystroke files, despite some deficiencies, in order to extract the necessary details of the interviewer's actions. Next, the need to send keystrokes reliably to the Data Entry Program (DEP) is best met using the Windows API; this method has distinct advantages over other methods. Finally, the principle technical obstacle involves finding a way to communicate the necessary information from the DEP to the "Replayer" program, particularly the currently focused field. This discussion will cover solutions to each of these problems and sample code will be provided as well.

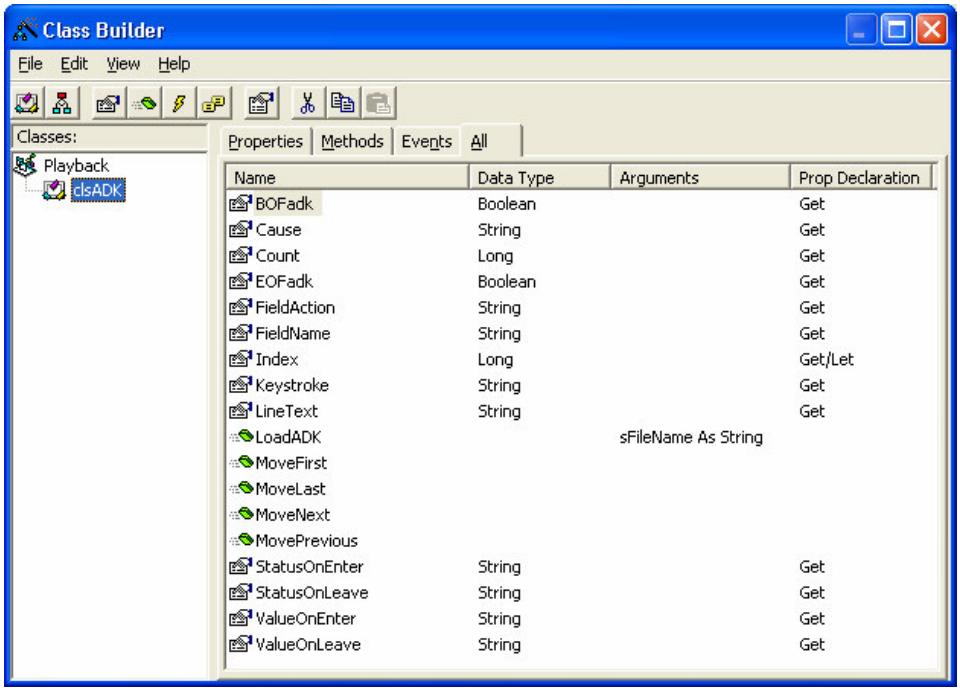
3.1 IMPLEMENTATION - BASICS

The prerequisite to building a "Replayer" program is a case management system or simulator that can serve up an accurate reproduction of the environment of the original test or interview during which the audit trail file (.adk) was generated. In some cases, it may be possible to use the same sample management system as the interviewers use in the field. In other cases, it may be necessary to develop an application to simulate this process. It is beyond the scope of this paper to describe this in detail. However, its basic functionality must include the ability to gather the correct datamodel, preload, and .adk file and to initiate the DEP with the preload correctly loaded.



A typical segment of an Audit Trail file (.adk)

The first task involved in programming a "Replayer" is to build a means to load the .adk data (see illustration above) in order for the program to know which values to place in which fields. This can be done, as needed, reading one line at a time in various parts of the program, but in the long run it will save time to program a simple .adk handler class. The class should contain cursor-like methods able to navigate an array containing the data which can be loaded from an adk by another method, and should provide easy access to the relevant data in the focused line (see illustration).



Properties and Methods of a class for handling .adks

As part of the design of a "Replayer" program, it is also necessary to consider the variety of functions needed, such as playback of single or multiple keystrokes, the ability to skip keystrokes when necessary, the ability to halt playback at a specified field, the ability to switch between playback of individual keystrokes and final values, and so on. HRS ultimately found it necessary to incorporate all of the above functionality.

3.2 IMPLEMENTATION - INHERENT OBSTACLES

There are a number of technical obstacles that should be considered which prefigure problems for the idea of retracing one's steps through a case. These can either be the result of questionnaire design decisions or they can be traced to deficiencies in the .adk generation process.

Questionnaire design can impede the process of retracing an .adk in a number of ways. If a datamodel is designed with conditions that affect branching that are based on time sensitive or randomized values generated on the fly, it may not be possible to expect that the instrument will play through the same as the .adk on any given occasion. For example, if the instrument calculates an age based on the current time and potentially branches differently based on that age, then an attempt to replay the case at a later date may result in a different path through the instrument. One solution may be to generate such values as preload. However, the instrument specifications may necessitate these session-sensitive values and special provision may have to be made, such as instrument-specific procedures for replaying certain problematic fields or even altering the computer clock during replay, for example. If such problems are infrequent then it may be sufficient to manually adjust the instrument when these issues arise, however. The HRS instrument does make use of session-specific data, but, due to the rarity of such problems during replay, no effort has yet been made to remedy this programmatically.

Similarly, if an instrument uses alien routers, these may also require workarounds, because .adk files contain little information about router activity. It is simple enough to manually answer routers when they appear, but there may be a need to replay cases without manual intervention. In

some cases, it may be possible to adjust the programming of a router to accommodate this need, or it may work to simply remove the files during replay if they do not alter values in the instrument. In the case of HRS, which calls two types of routers that output data, it has not yet been possible to adjust the programming of the routers, so special functions were built into the "Replayer" program to accommodate them. In this case, doing so involved making a separate version of the router for automatic playback which returned control to the DEP immediately. Then, by reading ahead in the .adk to determine the final value, the original router result could be output by the "Replayer" program in place of the router (in this case by means of a text file on the local drive).

Perhaps the most important instrument design issue to take into account is the complexity of preload. This is not specifically a concern for the "Replayer" program itself, but rather for the accompanying case setup application that may be required. For the program to retrace the original route of the .adk, it is of course necessary for the preload environment to match the original. In the case of HRS, the very complex preload has necessitated the development of more than one sample management simulation program to serve up the correct preload environment for different kinds of testing.

There are also deficiencies in the .adk generation process in Blaise (some probably by design, others perhaps are errors) that must be taken into account in the design of a "Replayer" application. The most significant issue is that mouse activity is not recorded in a direct way. This is a problem for two reasons. First, the mouse can be used to change the focused field of the questionnaire in a non-linear fashion. This is very disruptive to the job of retracing an .adk because it is not simple to determine how to move the current focus back (or forward) to the field put in focus by the mouse originally. Further, the act of realigning the focus can really only be done by adding keystrokes that did not originally occur (several up-arrow keystrokes, for example). HRS has approached this problem from two angles. First, HRS has implemented, in our testing and in the field, a small program which prevents mouse input to the FormPane and the InfoPane, thus forcing the tester or interviewer to move through the questionnaire using only keystrokes. Second, a function has been implemented in the "Replayer" program that optionally attempts to automatically realign with the .adk when the fields do not match. The latter has the added advantage of helping in situations where other kinds of problems have caused the actual and the recorded fields to become misaligned.

The second problem with mouse input is that answers can be chosen in the InfoPane without leaving information about the individual mouse clicks, but only the final value when exiting the field. This, too, has been addressed by HRS by means of disabling mouse input to the InfoPane. As an alternative, the "Replayer" also provides an option to use the values recorded when the field is exited rather than the actual keystrokes. This prevents the user from watching the actual original input, but works well for other purposes.

A sort of problem similar to changing focus with the mouse occurs when certain keys are pressed and held by the tester or interviewer. The .adk only records the original keypress, not the duration or the effect. Thus, pressing and holding the up-arrow key may move the focus by several fields, but will record only one keypress in the .adk. This also applies to keys such as Shift and Control; it is not possible to tell if the Shift key was pressed and released before the subsequent keys, if it was in effect for only one key, or if it affected several subsequent keystrokes, for example. Primarily, this issue affects focus and the solutions are similar to those used to combat the mouse-focus problem. This is one of the areas where the auto-realign feature of the "Replayer" might be used, or the realignment can always be done manually, if the process is supervised. In this case, HRS also instructs its testers and interviewers not to press and hold keys.

Other problems that occur are more difficult to pin down, such as lines missing or truncated in the .adk files, and a host of other rare and seemingly inexplicable mistakes. For example, one problem that comes up occasionally is that keystrokes appear between the Leave Field and Enter Field lines of an .adk, as if they were suspended in the gap between fields. What makes this issue particularly thorny is that sometimes it appears that these keys were processed by the DEP and at other times they appear to have been ignored. This inconsistency makes it very difficult to program a workaround into the "Replayer." Currently, HRS treats such keystrokes as if they should be ignored.

As much as these and other obstacles make the process of building a "Replayer" complicated, it is still possible to build an effective application to meet the testing needs outlined at the outset.

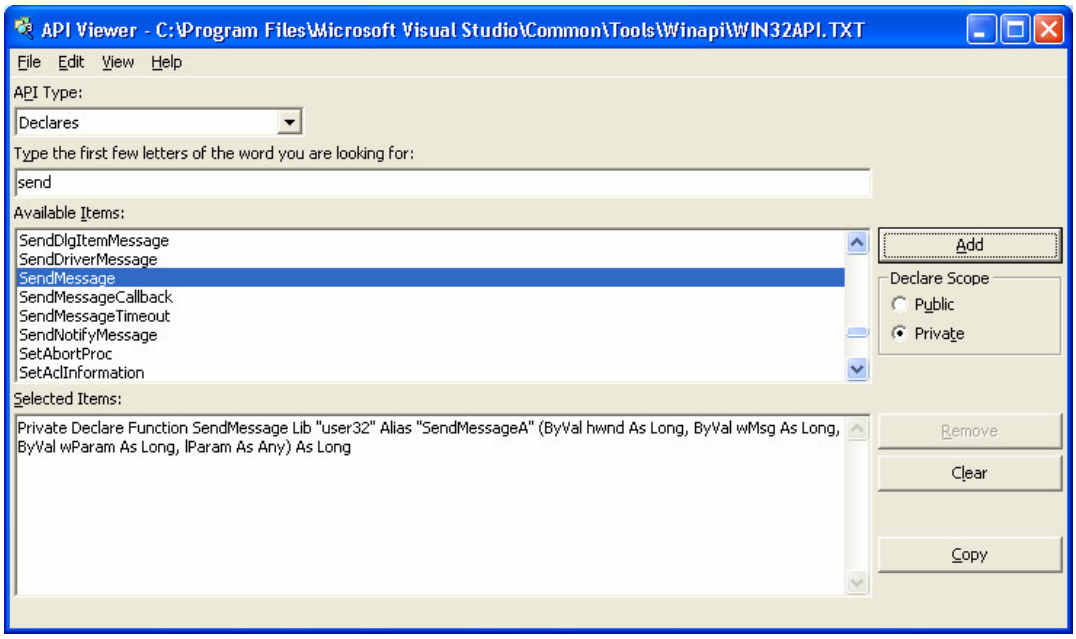
3.3 IMPLEMENTATION - TALKING TO THE DEP

The first requirement of such an application is that it be able to send information to the DEP in such a way as to simulate keyboard input. The basic concept is to transmit each new keystroke read from the .adk file to the DEP, mimicking the original test or interview. There are a number of ways to do this in Visual Basic 6; this section will touch on a few alternatives, but will focus mostly on a couple of useful Windows API commands.

The simplest way to simulate a keystroke in Visual Basic 6 is to use the `SendKeys` statement, which is implemented as follows: `SendKeys ("String to be sent")` This command will send any keystrokes given as an argument to the active window. However, the `SendKeys` statement has a number of shortcomings and eccentricities that make it problematic. The most important of these is that it requires the receiving window to be the active window, necessitating tight control over window switching. This limitation makes the application much less flexible, particularly insofar as there may be cases when it is necessary to alter the entries in the DEP manually during playback in order to test alternate scenarios or to compensate for the sort of inherent problems mentioned earlier that occasionally appear.

In order to bypass the problems of the `SendKeys` statement, it is necessary to use the Windows API, which is, in fact, the mechanism being used behind the scenes by `SendKeys` as well. Making direct use of it allows for greater flexibility and control in the production of simulated keystrokes. What is more, its use is required for other functions of the "Replayer."

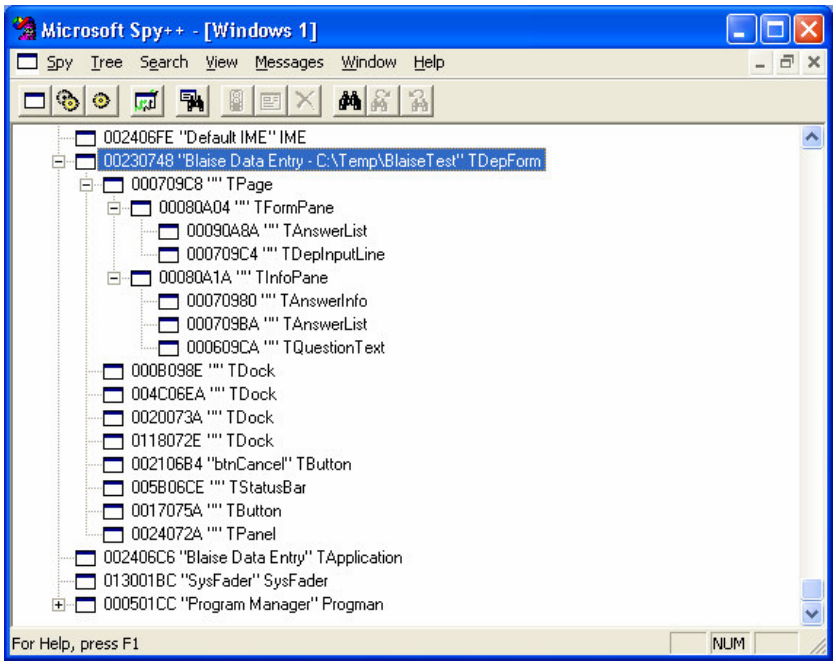
At this point, it may be worthwhile to review the basics of the Windows API and its use in Visual Basic 6 briefly. An API (or Application Programming Interface) is a collection of functions and properties of an operating system or application exposed for use by other developers. For example, BCP is an API for Blaise. The Windows API, which is of interest here, exposes much of the internal functionality of Windows to the developer. For purposes of the "Replayer," it offers direct access to the message systems upon which Windows functions. To make use of the API in Visual Basic 6, it is necessary to declare the API function in the general declarations section, and then it is possible to call the function just as a normal function. To find the correct declaration for an API function, launch the following utility (from the Start menu, assuming Visual Studio 6 is installed): Start > All Programs > Microsoft Visual Studio 6.0 > Microsoft Visual Studio 6.0 Tools > API Text Viewer (see illustration).



API Text Viewer

Using the Windows API, keystrokes can be simulated using the `keybd_event` function or the `SendInput` function which superseded it in NT-based Windows operating systems. However, these pose similar problems to `SendKeys` in terms of windows switching. Therefore, this paper will proceed directly to the `SendMessage` function and its variants, which are used by the HRS "Replayer" program.

Here, again, it may be useful to pause briefly to consider the relationship of these functions to the internal processes of Windows. A messaging mechanism in the operating system controls communication with the windows that appear on the desktop. For example, when a key is pressed, a message is generated by the operating system that is sent to a queue of messages associated with the active window. When the window receives the message, it will learn that a particular key was pressed and may take an appropriate action, such as displaying it on the screen. Each apparent window may also contain a number of other windows within. For example, the DEP contains a number of window objects, including the DEPInputLine, which will be the focus of much of this section. Each window object is referenced by a "handle," which will be an argument passed to many of the procedures described below. Note that the handle for an application will differ each time it is run; it only stays unchanged while the application remains open. The list of windows and their handles, and the associated message queues can be viewed with the following utility, which will be referred to again later: (from the Start menu): Start > All Programs > Microsoft Visual Studio 6.0 > Microsoft Visual Studio 6.0 Tools > Spy++ (see illustration).



Microsoft Spy++

The `SendMessage` function provides a way to send a message (containing a key to be processed) directly to the desired window, even if it is hidden. This is accomplished by passing as arguments of the function the handle of the window, the type of message, and a value indicating the desired key. First, the handle of the receiving window must be determined. The following code can be used to retrieve the handle of the DEP window and then of the DEPInputLine window. (From this point forward, where code is presented as follows, it can be considered to be part of a sample project referred to by the default name of "Project1" that will demonstrate the most important mechanisms of the "Replayer.")

[Put this code in a module]

```
Public Declare Function EnumWindows _
  Lib "user32" _
  (ByVal lpEnumFunc As Long, ByVal lParam As Long) As Boolean
Public Declare Function GetWindowText _
  Lib "user32" Alias "GetWindowTextA" _
  (ByVal hwnd As Long, ByVal lpString As String, _
  ByVal cch As Long) As Long
Public Declare Function GetWindowTextLength _
  Lib "user32" Alias "GetWindowTextLengthA" _
  (ByVal hwnd As Long) As Long
Public gsWndTxt As String
Public gsWndFind As String

Public Function EnumWindowsProc(ByVal lhwnd As Long, _
 ByVal lParam As Long) As Boolean
  Dim sGetWndTxt As String
  Dim lRet As Long
  lRet = GetWindowTextLength(lhwnd)
  sGetWndTxt = Space(lRet)
  GetWindowText lhwnd, sGetWndTxt, lRet + 1
  If sGetWndTxt <> "" Then
    If InStr(1, sGetWndTxt, gsWndFind) Then
```

```
      gsWndTxt = Trim$(sGetWndTxt) 'get exact window text
    End If
  End If
  EnumWindowsProc = True 'continue enumeration
End Function
```

[Put this code in a form]

```
Private Declare Function FindWindow _
  Lib "user32" Alias "FindWindowA" _
  (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Private Declare Function GetWindow _
  Lib "user32" _
  (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function GetClassName _
  Lib "user32" Alias "GetClassNameA" _
  (ByVal hwnd As Long, ByVal lpClassName As String, _
  ByVal nMaxCount As Long) As Long
Private Const GW_CHILD = 5
Private Const GW_HWNDNEXT = 2
Private mlhWnd As Long
Private mlhInputLine As Long

Private Sub FindWindowHandle(sFind As String, Optional sChildFind As
String)
  gsWndFind = sFind
  gsWndTxt = ""
  EnumWindows AddressOf EnumWindowsProc, ByVal 0&
  If gsWndTxt <> "" Then
    mlhWnd = FindWindow(vbNullString, gsWndTxt)
    GetChildHandle mlhWnd, sChildFind, mlhInputLine
  Else
    mlhWnd = 0
    mlhInputLine = 0
  End If
End Sub

Private Sub GetChildHandle(lhwnd As Long, sFind As String, _
 ByRef lhwndChld As Long)
  Dim lChldWnd As Long
  Dim sChldTxt As String * 250
  Dim sGetWndTxt As String
  Dim lRet As Long
  lChldWnd = GetWindow(lhwnd, GW_CHILD)
  If lChldWnd <> 0 Then
    Do Until lChldWnd = 0
      GetClassName lChldWnd, sChldTxt, 250 'get child window's
classname
      lRet = GetWindowTextLength(lChldWnd)
      sGetWndTxt = Space(lRet)
      GetWindowText lChldWnd, sGetWndTxt, lRet + 1
      If InStr(1, sChldTxt, sFind) > 0 Then
        lhwndChld = lChldWnd 'find class name like TDEPInputLine
      ElseIf InStr(1, sGetWndTxt, sFind) > 0 Then
        lhwndChld = lChldWnd
      End If
      GetChildHandle lChldWnd, sFind, lhwndChld
```

```
      lChldWnd = GetWindow(lChldWnd, GW_HWNDNEXT) 'Search next child
    Loop
  End If
End Sub

Private Sub Command1_Click()
  'Make a button and three text boxes for this Sub
  FindWindowHandle "BlaiseTest", "TDepInputLine"
  'the argument "Blaise Test" in the line above should be the optional
  'text description of the datamodel or part of the datamodel filename
_
  'it simply has to be part of the text that appears in the title bar
of
  'the DEP window which distinguishes it from other windows
  Text1 = gsWndTxt 'full title bar of DEP Window
  Text2 = mlhWnd 'handle of DEP window
  Text3 = mlhInputLine 'handle of DEP Input Line
End Sub
```

Once the handle of the DEPInputLine has been established for a given session, it is possible to make values appear in it by using the `SendMessage` function. The declaration for the Windows API `SendMessage` function is as follows. Additionally, a few helpful constants are specified as well.

[Add this code to the declarations section (the top) of the form]

```
Private Declare Function SendMessage _
  Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, lParam As Long) As Long
Private Const WM_CHAR = &H102
Private Const WM_KEYDOWN = &H100
Private Const WM_KEYUP = &H101
```

Then it is quite simple to send a character to the DEP using the `SendMessage` function. An example of this follows. Note that it makes use of the `WM_CHAR` constant to send a character to the DEPInputLine window and that it requires an Ascii value representing the character to be sent.

[Add this code to the form]

```
Private Sub Command2_Click()
  'Make another button for this Sub
  Dim s As String
  s = "1" 'This is the character to be sent to the DEP
  Call SendMessage(mlhInputLine, WM_CHAR, Asc(s), 1)
  'Arguments:
  'mlhInputLine - the handle of the DEPInputLine
  'WM_CHAR - a constant that indicates a character message
  'Asc(s) - the Asc() function turns the string, s, into an Ascii value
  '1 - the last argument indicates one repetition - see MSDN for a
list:
  'go to http://msdn.microsoft.com and search for WM_CHAR
End Sub
```

It is also quite easy to send non-character keystrokes, like a carriage return, a tab or an arrow key. In this case, the `WM_KEYDOWN` and `WM_KEYUP` constants can be used in succession to simulate a

press and a release of a key. Here, the arguments for the `SendMessage` function include a virtual key constant representing the non-character key to be pressed. The code that follows is an example of this.

[Add this code to the declarations section (the top) of the form]

```
Private Const VK_RETURN = &HD 'virtual key constants
Private Const VK_UP = &H26
Private Const VK_DOWN = &H28
Private Const VK_LEFT = &H25
Private Const VK_RIGHT = &H27
Private Const VK_TAB = &H9
Private Const VK_HOME = &H24
Private Const VK_END = &H23
Private Const VK_PRIOR = &H21 'Page Up
Private Const VK_NEXT = &H22 'Page Down
Private Const VK_DELETE = &H2E
Private Const VK_BACK = &H8
Private Const VK_CLEAR = &HC 'NumPad 5 with NumLock off
Private Const VK_SHIFT = &H10
Private Const VK_ESCAPE = &H1B
Private Const VK_CAPITAL = &H14 'Caps Lock
```

[Add this code to the form]

```
Private Sub Command3_Click()
  'Make another button for this Sub
  Dim s As String 'string representing non-character key to send
  Dim iKey As Integer 'counter of keys pressed in the current field
  Dim lVK As Long 'virtual key constant to send
  Dim lhwnd As Long 'window handle to receive message
  lhwnd = mlhWnd 'mlhwnd is the handle of the DEP window
  s = "[TAB]" 'would normally be passed in as an argument
  iKey = 1 'would normally be passed in as an argument
  Select Case s
    Case "[ENTR]":
      lVK = VK_RETURN
    Case "[BACK]":
      lVK = VK_BACK
      lhwnd = mlhInputLine 'this key should go to the input line handle
    Case "[DEL]":
      lVK = VK_DELETE
      lhwnd = mlhInputLine 'this key should go to the input line handle
    Case "[TAB]":
      lVK = VK_TAB
    Case "[DOWN]":
      lVK = VK_DOWN
    Case "[UP]":
      lVK = VK_UP
    Case "[LEFT]":
      lVK = VK_LEFT
      If iKey <> 1 Then 'If it is not the first keystroke on a line,
        lhwnd = mlhInputLine 'this key should go to the input line
handle
      End If
    Case "[RIGHT]":
      lVK = VK_RIGHT
      If iKey <> 1 Then 'If it is not the first keystroke on a line,
```

```
        lhwnd = mlhInputLine 'this key should go to the input line
handle
      End If
    Case "[HOME]":
      lVK = VK_HOME
    Case "[END]":
      lVK = VK_END
    Case "[PGUP]":
      lVK = VK_PRIOR
    Case "[PGDN]":
      lVK = VK_NEXT
    Case "[ESC]":
      lVK = VK_ESCAPE
    Case "[CAPS]":
      lVK = VK_CAPITAL
  End Select
  If lVK <> 0 Then 'Send Virtual Key to message queue
    Call SendMessage(lhwnd, WM_KEYDOWN, lVK, 0)
    Call SendMessage(lhwnd, WM_KEYUP, lVK, 0)
  End If
  If s = "[BACK]" Or s = "[DEL]" Then
    'This fixes a problem with nonresponse after sending Delete
    Call SendMessage(lhwnd, WM_KEYDOWN, VK_CLEAR, 0)
    Call SendMessage(lhwnd, WM_KEYUP, VK_CLEAR, 0)
  End If
End Sub
```

With the information outlined above, it is possible to construct a program which reads the .adk file, locates the DEP window, and sends the keystrokes recorded within the .adk file to the DEP one by one. This assumes, of course, that the DEP is prepared to receive the keys in the manner and in the order specified by the .adk file. The next section will discuss ways to get information from the DEP about its status, such as the currently focused field.

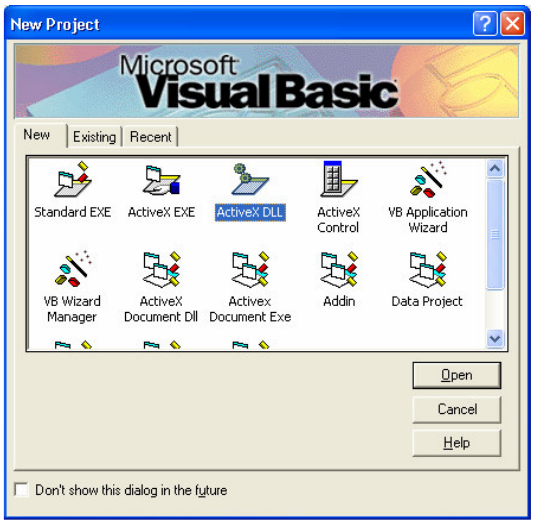
Finally, there are other potential options. It would be interesting to consider a program that recorded keystrokes and mouse movements separately from the DEP, forming its own trail for playback that could be better tailored to the specific needs of the utility. The fundamental problem for that hypothetical utility, and, indeed, for the one under consideration here, is that the keystroke information has to be matched to field information from the DEP in order for replay to occur in a well-orchestrated fashion. A solution to this problem will be covered in detail in the next section.

3.4 IMPLEMENTATION - BUILDING A TWO-WAY BRIDGE

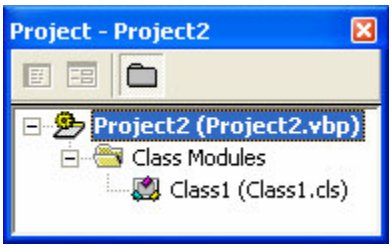
At this point, a system for receiving information from the DEP is needed. There are a variety of ways to build a communication bridge between the “Replayer” program and the DEP. This paper will present three variants in order of performance. All three are based on the same basic scheme, but differ in the way that they return data from the DEP to the “Replayer” program. The first method to be discussed was not implemented by HRS because it seemed too slow, but it has the virtue of being quite simple to implement and it lays a nice foundation for understanding the other two. The latter two methods have been used by the HRS “Replayer,” with the last one being in current use.

The basic scheme is to add a menu entry to the DEP which can call an external procedure to provide information about the status of the DEP. This menu item can be activated by the "Replayer" program as needed. Currently, the preferred type of external procedure call utilizes COM objects in ActiveX libraries. These are longstanding Microsoft technologies that are currently obsolescent, but which are not difficult to use with Visual Basic 6. This section will discuss building a simple ActiveX dynamic link library (DLL) which will provide information about the DEP. Then the discussion will turn to making use of the DEP Menu Manager (found under tools in the Blaise Control Centre) to add a menu entry which will invoke the COM object method in the ActiveX DLL. Finally, the discussion will cover how to activate the DEP menu entry from the Sample project covered in the previous section and how to receive the information which this process makes available. (For more information about using COM with Blaise, search for "calling COM DLL" in Blaise Help.)

For this discussion, a second Visual Basic project will be referenced. This project will be an ActiveX DLL project and will be named "Project2." The name is important because it will be used by the Blaise Menu Entry that will be added. This project will also need to reference BCP.



Start an ActiveX DLL Project…



Rename the project to "Project2"…

The Active X DLL should have at least two procedures. One of these should be of public scope so as to be able to be called directly by the Blaise Menu Entry. The other should be a supporting function, called from the first, which will transmit the information. This second procedure can initially be set up to transmit the information by the simplest means possible, which is to write a text file in a predetermined location.

[Put this code in the class]

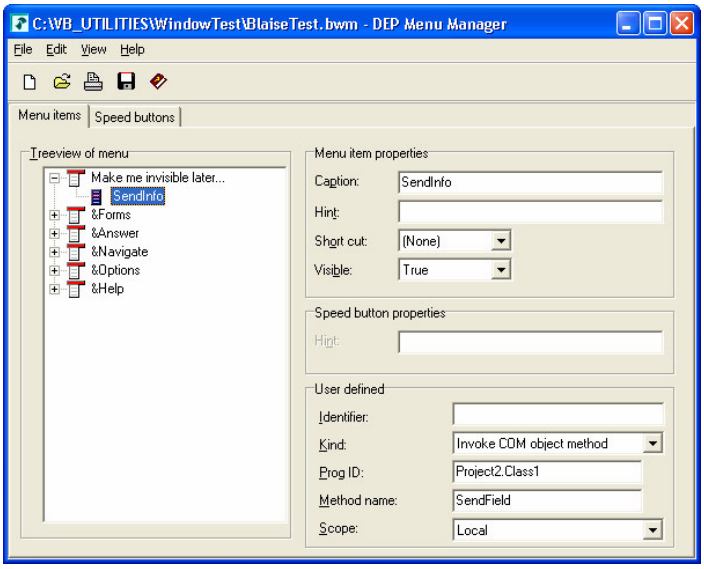
```
Public Sub SendField(db As BlAPI4A2.Database, ds As BlAPI4A2.DepState)
  'Don't forget to select "Blaise API Objects Library 2.0"
  'under Project > References...
  'See Blaise Help for more information about this code
  Dim Quest As Question
  Dim Layout As LayoutSet
  Dim Parallel As Parallel
  Dim Page As StoredPage
  'Get hold of the currently active question...
  'The properties of the DepState parameter need to be used for this...
  Set Layout = db.Screens.LayoutSetCollection(ds.LayoutSetIndex)
```

```
  Set Parallel = Layout.ParallelCollection(ds.ParallelIndex)
  Set Page = Parallel.StoredPageCollection(ds.StoredPageIndex)
  Set Quest = Page.QuestionCollection(ds.QuestionIndex)
  'send name of current field
  SendData Quest.Field.Name
End Sub

Private Sub SendData(sData As String)
  Open "C:\DEPInfo.txt" For Output As #1
  Print #1, sData
  Close #1
End Sub
```

When the code for this DLL is compiled (File > Make Project2.dll…), it will automatically be registered for use in its initial location. If it is later moved or installed on another machine, it will need to be registered in its new location. This can be done by accessing a command prompt in that location and typing "regsvr32 Project2.dll" at the prompt.

The DEP Menu Manager makes possible the generation of a .bwm file which can add new menu items to the DEP. To proceed with this example, select "Add Menu Entry" and enter a name for the new menu entry in "Menu item properties." Then set the values in the "User defined" segment as follows: Kind = Invoke COM object method; Prog ID = Project2.Class1; Method Name = SendField (see illustration).

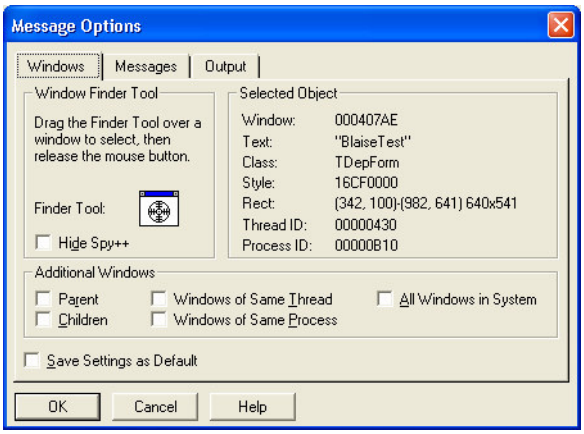


DEP Menu Manager

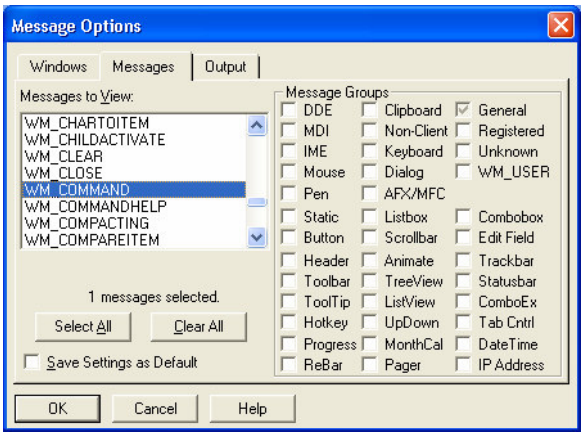
After saving and exiting the DEP Menu Manager, select the menu file under Run > Parameters in the Blaise Control Centre or specify it in the command line parameter of the DEP. This will allow the new menu entry to appear in the DEP when it is run.

At this point, it is necessary to run the DEP with the new menu file. When the DEP is running with the new menu option, start the program called Spy++ which was referred to in the previous section. Open the Message Options dialog (Spy > Messages) from the menu. This will cause a dialog box to appear with a target-shaped cursor called the Finder Tool. Click and drag on the finder tool to pull it over the DEP title bar and then release. The information in the Selected Object segment should appear as in the illustration below (with different numbers). Now that the window has been selected, switch to the messages tab and press "Clear All." Then select WM_COMMAND in the

Messages to View list and press the OK button to close the dialog. This will make it easy to see only the WM_COMMAND message that is of interest.



The Finder Tool



Message Options in Spy++

After closing the Message Options dialog, switch to the DEP window and select the new menu entry; incidentally, this action should produce the C:\DEPInfo.txt file containing the currently focused fieldname. Switch back to Spy++ and record the ID of the menu item (see the illustration below). After all of this has been done, it is possible to make the menu entry invisible in the .bwm so that the user cannot activate it unintentionally.

The ID of the menu item as seen in Spy++ (here the ID is 136)

At this point, the menu entry can be activated from the "Replayer" program whenever the program needs to know what field is currently active in the DEP. For this operation, the windows API function that should be used is called `PostMessage` and the message type is `WM_COMMAND`. Internally, `PostMessage` operates slightly differently than `SendMessage`, but it still takes the window handle as an argument. The following code examples are intended to be added to the sample project called "Project1" from the preceding section.

[Add this code to the declarations section (the top) of the form]

```
Private Declare Function PostMessage _
  Lib "user32" Alias "PostMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, _
  ByVal wParam As Long, ByVal lParam As Long) As Long
Private Const WM_COMMAND = &H111
'ID of menu item for calling ActiveX dll (Get from Spy++)
Private Const wID_SendInfo = 136
```

Two procedures are given below. The first activates the new menu entry when the DEP is running. The second simply loads and displays the fieldname that is saved when the first procedure is executed.

[Add this code to the form]

```
Private Sub Command4_Click()
  'Make another button for this Sub
  Call PostMessage(mlhWnd, WM_COMMAND, CLng(wID_SendInfo), 0)
End Sub

Private Sub Command5_Click()
  'Make another button and another text box for this Sub
  Dim s As String
  Open "C:\DEPInfo.txt" For Input As #1
  Line Input #1, s
  Close #1
  Text4 = s
End Sub
```

This setup is clearly quite inefficient and lacks automation, but there are many ways to automate the process. For example, a loop could be added to the first procedure which would wait for the date on the file to change (indicating that new data had arrived) before loading the data from the

file. However, there are even better ways to automate this process and they will be described below.

The beginning of this section announced that three different methods would be presented. The first has already been covered and uses a text file to return the data to the calling program. The other two methods operate on the same principle, but offer a more complicated yet more efficient means of returning the data.

3.4.1 IMPLEMENTATION - BUILDING A TWO-WAY BRIDGE – RETURN METHOD 2

The second method of returning data is to send it to a hidden textbox in the "Replayer" program; this was used in the HRS "Replayer" for about a year and a half and worked quite well. In this method, the data is sent to a textbox in a hidden form in the "Replayer" program by means of a `WM_SETTEXT` message.[6] The Change event of the textbox will fire when the data arrives, at which point the textbox contents can be used to check the alignment of the DEP and the .adk. When the whole process first begins with the message to activate the menu item, the main program is set to wait until this data shows up before it proceeds.

As before, the window handle of the receiving object must be determined, and the same code can be used as shown above, except that it will search for the name of the hidden form and textbox to receive data. The message can be sent using the `SendMessage` function as follows:

```
SendMessage(longHandleOfTextBox,       WM_SETTEXT,       0,       ByVal
stringDataToBeSent)
```

On the hidden form, the Change event of the textbox can be set to do something with the data, such as change the value of a global variable. If the textbox is called Text1 then the `Text1_Change` event can also include this line: `Call ReplyMessage(1)` This requires reference to the `ReplyMessage` API function in the declarations section:

```
Private Declare Function ReplyMessage _
  Lib "user32" _
  (ByVal lReply As Long) As Long
```

3.4.2 IMPLEMENTATION - BUILDING A TWO-WAY BRIDGE – RETURN METHOD 3

For the past several months, HRS has been using a new system to send data back to the "Replayer" using `WM_COPYDATA`. This process makes use of undocumented pointer functions in Visual Basic to copy a data structure and is based on *Microsoft Knowledge Base Article #176058 - How To Pass String Data Between Applications Using SendMessage*.[7] Please note the warning about using undocumented functions. The article covers the needed declarations and retrieval code that should be put in a module in the "Replayer," but, as an example, the code for the send procedure in the DLL might look like this:

```
Private Sub SendData(sData As String)
  Dim cds As COPYDATASTRUCT
  Dim buf(1 To 255) As Byte
  Dim l As Long
  Call CopyMemory(buf(1), ByVal sData, Len(sData))
  cds.dwData = 3
  cds.cbData = Len(sData) + 1
  cds.lpData = VarPtr(buf(1))
  lhWnd = FindWindow(vbNullString, "Title of Target Replayer Window")
  l = SendMessage(lhWnd, WM_COPYDATA, 0, cds)
End Sub
```

[6] For further discussion of this method, see http://www.thescarms.com/vbasic/PassString.asp
[7] See http://support.microsoft.com/default.aspx?scid=kb;en-us;176058

This method seems to operate considerably faster than the ones previously described and has worked well in testing up to this point.

3.5 IMPLEMENTATION – CLOSING ERROR DIALOGS

Most Blaise instruments will incorporate some consistency checks. These, and the messages that appear, for example, when an invalid value is input, have to be handled if replay is to proceed without the need for constant supervision. This can easily be accomplished by sending a `BM_CLICK` message to the handle of the button which will close the dialog. In order to determine when to send this message, a timer can be set up to call a procedure that will search for known error dialog types using code similar to that used to find the DEP and its handle. This needs to happen asynchronously, so HRS initially set up a small external executable, the sole function of which was to find and close error dialog boxes. This executable was loaded to coincide with the start of the process and was terminated at the end of replay. Later, as the need to return information about the error dialogs surfaced, HRS replaced the separate program with an object in an ActiveX .exe which could be referenced directly. This object performs the same function as the separate executable described above but also exposes information about its activity so that the "Replayer" program is aware of any attempts to close a dialog. This solves certain occasional problems that arise during replay in which the "Replayer" needs to know whether certain keystroke combinations were caused by an error dialog.

Despite the apparent complexity of the "Replayer" due to the number of operations that must be taken into account, the amount of program code needed is rather small. Neither is it terribly time consuming or difficult to program.

4. RESULTS

The purpose of developing this application was to improve data quality while reducing cost and increasing testing efficiency. During the year and a half that the "Replayer" program has been in use, HRS has seen a marked decrease in the amount of time and effort expended on investigating and solving problems as part of the instrument development process. In 2003 and 2004, the principal users of this application were programmers who used it to replay both tester and field .adks. It proved capable of significantly reducing the amount of time programmers spent on diagnostic activities, such as reproducing problems, and drastically reduced the amount of time needed to retest fixes. Programmers found that they were able to quickly replay keystroke files that they received from the testers who reported problems in order to verify the existence of an error being reported. After making changes to the Blaise code intended to fix an error, programmers were able to use the "Replayer" again to check the effectiveness of the fixes.

The successful incorporation of this program into the testing process is reflected in the fact that the number of reported errors from the field has declined by more than half against a comparable period in the previous wave. As an added bonus, the "Replayer" has proven to be an effective tool in reviewing interviews exactly as they were taken and has substantially reduced the time it takes to investigate problems reported from the field and from the data review process. This ability not only reduces the time involved in program corrections, but also in reviewing issues of Interviewer performance. For example, at the time of this writing, the program had just been used to replay a couple of cases in which the interview was conducted following the wrong path (e.g. Self-R instead of proxy; living R instead of deceased) for the situation. In these cases, it was necessary to examine the incorrect flow to diagnose the situation and to determine whether the interviews in question needed to be retaken. More generally, this program has helped HRS to spot less than optimal interviewer behavior, such as backing up and changing fields from earlier sections. Sometimes this leads to improvements in the field process that may result in better data being collected.

5. CONCLUSION

Plans to expand the use of this utility have already been made for the 2006 wave of HRS. HRS is planning on incorporating this utility fully into the overall testing process and is developing a database for specific scenarios that will capture scenario attributes and matching ADK files. Testers will then be allowed to select and replay a scenario at any phase of testing and check results against a number of saved formats.

The success HRS has had in using this utility would seem to be applicable to other studies as well. It is hoped that a discussion of this program will generate interest in this aspect of instrument development.

Blaise AT Report System

Youhong Liu, Eduardo Galvan, Gina Cheung
Institute for Social Research
University of Michigan

1. Introduction

During an interview or data entry session, Blaise records interviewer actions in an Audit Trail/Keystroke (ADK) file. This text file not only records the values of each field before and after the cursor enters a field, it also captures the actions that brought the cursor to the field and when exiting the field. Timestamps, keystrokes, remarks, the use of Help, switching languages, and other functions are recorded for each field visited.

In the last few years, the Institute for Survey Research at the University of Michigan (UM) implemented the Blaise Audit Trail for all projects and developed tools to analyze the data produced by them. In this paper, we will present the most recent tool developed, the Blaise Audit Trail Report System.

2. History

UM had an Audit Trail analysis program named ADK_Report.exe. It had been used extensively by the UM data processing and quality control team and methodology researchers. However, there were some problems and limitations with this program. The program also was written in Delphi, an older programming language no one in our group was familiar with. After using BCP for several years, we decided to rewrite the Audit Trial analysis program using BCP.

3. Programming Language and Backend Database

The programming language we used for the system is Visual Basic (VB). It is a high level language that we have much experience using with BCP. The system produces online reports that require a backend database such as Microsoft Access, Microsoft SQL or Oracle. MS SQL and Oracle seemed overkill for this system, while Access is not only more portable but does not have to be connected to a network to perform an analysis.

4. Reporting Tool

The built-in reporting tool found in VB did not produce the desired results. By switching to Crystal Reports and its many functions, the data could be presented in various ways. This will be discussed later in the paper.

5. Two Subsystems

The system contains two subsystems:

a. Standard and Customer Report Creation

This loads Audit Trail timing data to an Access database and produces various reports through the Crystal Reporting tool.

b. Audit Trail Data Generation

This loads data into a text file that can be used to feed to other software programs such as SAS, MS Access and Excel for further analysis. This option produces more robust information from the Audit Trail. The General Audit Trail option provides information on functions used, language switching and back up patterns that are not available with the Standard and Custom Report Creation option.

6. Standard and Customer Report Creation

a. Database Selection – Figure 1

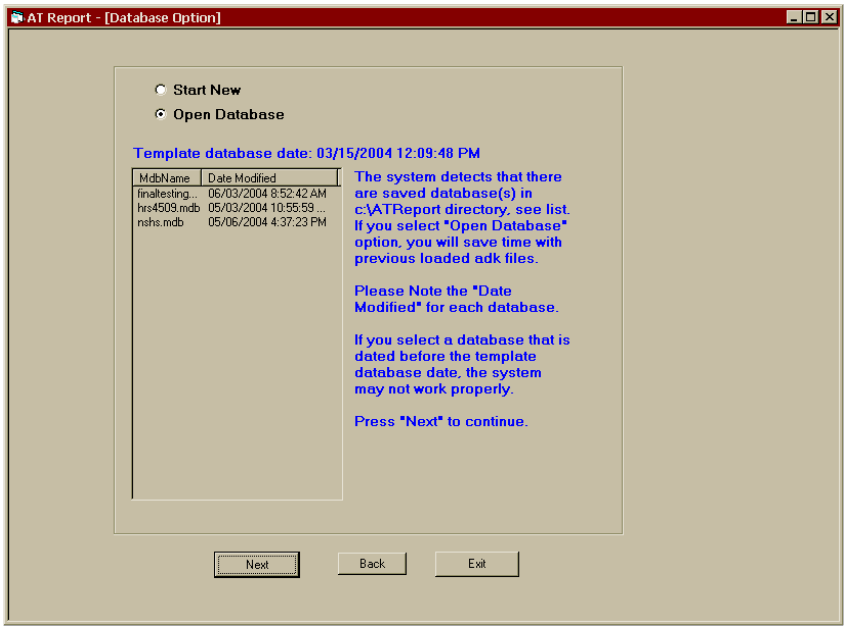


Figure 1 – Database Selection

The backbone of this portion of the system is an MS Access database – ATReport.mdb. The empty ATReort.mdb is located in the \support subdirectory where the AT report executable is located. E.g., if ATReport.exe is under C:\temp, then the Access file will be in C:\temp\support directory.

Database Structure:

Tables

- Case_Standard – For Blaise case level data
  Fields
  - CaseID
  - CaseTime
  - NumVisit
  - Iwer
  - IwerDate
  - FileIndex
- Field_Standard – For Blaise field level data
  Fields
  - CaseID
  - FieldName
  - FieldTime
  - Visits
  - BlockName
  - FileIndex
  - IwerName
- FileName – For file name (keeping track of the file name to save loading time)
  Fields
  - FileNamePath
  - FilePath
  - FileIndex
  - TSTAMP
  - FieldUpdate
  - CaseUpdate
- Field_Cust – For customer section reports
  Fields
  - FieldBlockName
- Section_Cust – For customer section reports
  Fields
  - SectionName
  - FieldBlockName

As shown in Figure 1, the user can perform one of two things:

- Start New:
  This will copy an empty ATReport database to the C:\ATReport directory. Upon exiting program, you will have a chance to rename the database to a meaningful name.

- Open Databases
  This will allow you to open a previously processed database. In this case, the data is already in the database, and it will eliminate time for loading data to the tables.

b. ADK File Selection (Figure 2)

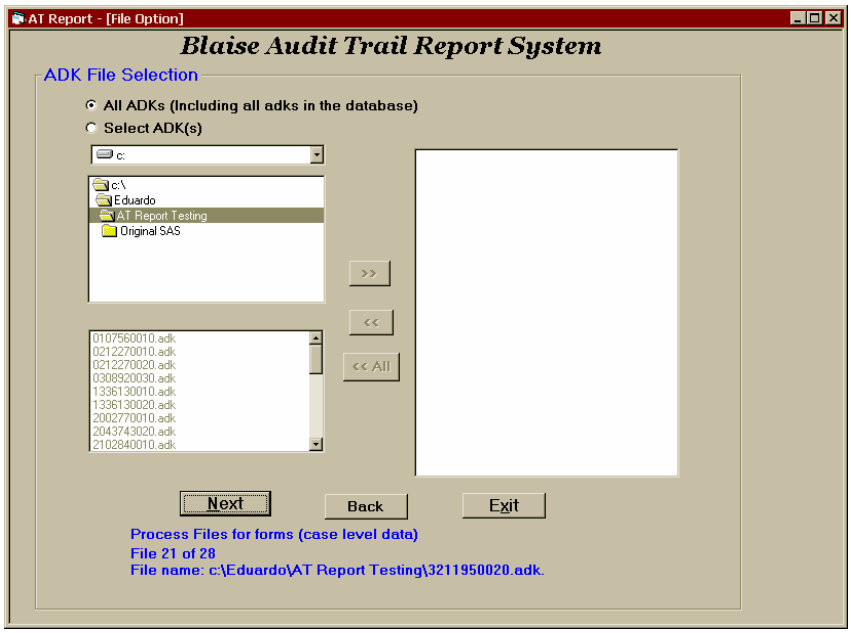


Figure 2 ADK File Selection

At this point, users can navigate to the directory containing Audit Trail files that are to be processed. At the University of Michigan, the extension of the Audit Trail file is ADK. The system offers two options. You can either select the whole directory of files or a subset of files in the directory.

c. Loading Data to Tables

After clicking "next" button shown in Figure 2, the system will start to load data to the Access tables; this part is the heart of the program. It processes Audit Trail files in two passes: case level and field level.

Case level:

Case ID – Primary Key
CaseTime – Total interview times of all forms for a case
NumVisit – Number of visits to a case
Iwer – Window user name in an Audit Trail
IwerDate – Interview date

Field level

Case ID
FieldName – Blaise field name
FieldTime – Total times for a field
Visits – Number of time the fields is accessed
BlockName – Top Block name of the field.
IwerName – Windows user name in an Audit Trail

d. Standard Reports (Figure 3)

Figure 3 – Standard Report Selection

Once the data are loaded into the database, the Crystal Reporting tool is used to generate reports. Unlike the Custom Reports that will be discussed later in this paper, the Standard Reports portion of the application does not allow a user to select which field or blocks to include in the timing reports that are generated. Although this may seem inflexible, the tradeoff is that these reports are simpler for the user to generate and also take less time to generate.

- Total Interview Time By Case ID—This report lists (for each ADK processed) the caseid, interviewer id, interview date, the number of times the case was accessed by the interviewer, and the total amount of time it took to complete the case.

  Basic SQL statement: *Select * from case_standard*

- Average Interview Time by Interviewer ID—This report lists average interview length and number of cases completed by each interviewer.
  Basic SQL statement: *Select * from (Select Iwer, Avg (CaseTime), count(caseID) from Case_standard Group by Iwer)*

- Average Interview Time for All Cases—This report provides the average interview length for all the cases that were selected for processing by a user. The total number of cases is also provided.

  Basic SQL statement:*Select Avg(CaseTime) From Case_Standard;*

- Average Field Time by Field Name—This report lists the Blaise instrument field name, the number of interviews in which the field was visited, and average time spent in the field.

  Basic SQL statement: *Select FieldName, avg(FieldTime), count(visits) from field_standard group by fieldname;*

- Average Block Time by Block Name—This report lists Blaise block names, the number of cases having timing data for the block, and average time spent in the

block. The average is based on the number of cases that visited the particular block and is not based on the total number of ADKs processed.

Basic SQL statement:*Select Blockname, avg (sumTime) from (Select CaseID, Blockname, sum(fieldtime) as sumTime from field_standard group by caseID, Blockname Having BlockName Is Not Null)*
Group by Blockname

e. Custom Reports – Selected Fields/Blocks

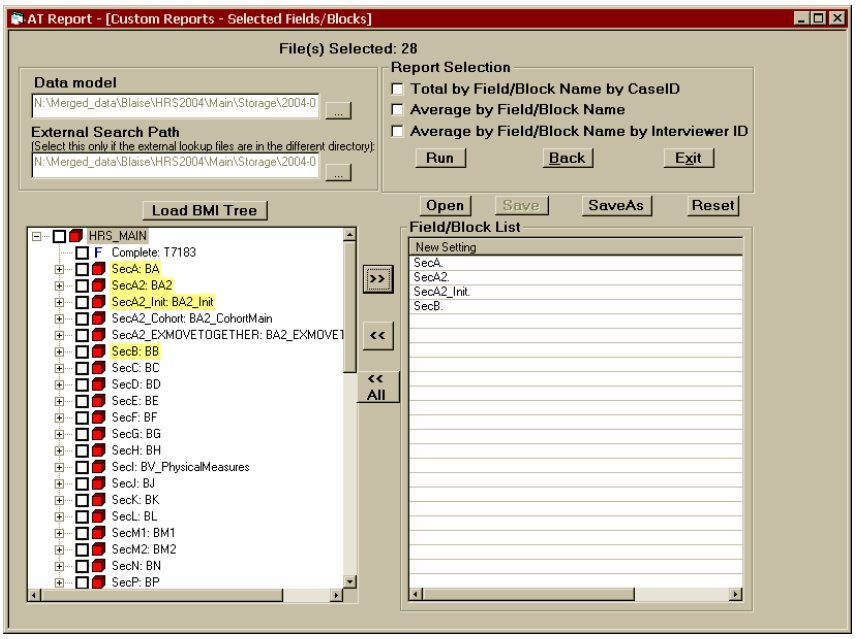


Figure 4 – Custom Report- Fields/Blocks Selection

The Custom Reports – Selected Fields/Blocks portion of the application gives a user more flexibility in determining which fields/blocks to use in running reports.

In order to pick Fields/Blocks, the corresponding Data Model has to be displayed for the user.

See Figure 4, after picking up the Blocks/Fields and "Run" button is clicked, the selected Blocks/Fields is inserted into Cust_field table.

The insert SQL Statement: *Insert into cust_field values ('" & ListView1.ListItems(i).Text & "')"*

- Total by Field/Block Name by CaseID—This report provides the total amount of time (in seconds) spent and the number of visits made by an interviewer into each block/field specified by a user. The data are sorted in CaseID order for each of the specified blocks/fields.

  Basic SQL Statement: *Select cust_field.fieldblockname, caseID, sum(fieldtime/1000), sum(Visits) from (select * from field_standard) as T1, cust_field where mid(T1.Fieldname, 1, len (cust_field.fieldblockname)) =cust_field.fieldblockname group by fieldblockname, caseID*

- Average by Field/Block Name—For each block/field specified the report lists the average time (in seconds) needed to complete the particular

block/field. The report also lists the number of cases that had the timing data for the particular block/field.

Basic SQL Statement: *Select fieldblockname, avg(sumfieldtime), count(CaseID) from (Select cust_field.fieldblockname, caseID, sum(fieldtime/1000) as sumFieldtime from (select * from field_standard) as T1, cust_field where mid(T1.Fieldname, 1, len (cust_field.fieldblockname)) = cust_field.fieldblockname group by fieldblockname, caseID) group by fieldblockname*

- Average by Field/Block Name by Interviewer ID--For each block/field specified, the report lists the average time (in seconds) needed by each interviewer to complete the particular block/field. A count of the number of cases completed by each interviewer is provided. The data are sorted in interviewer id order for each of the specified blocks/fields

  Basic SQL Statement: *Select IwerName, fieldblockname, count(CaseID), avg(sumfieldtime) from (Select cust_field.fieldblockname,Iwername, caseID, sum(fieldtime/1000) as sumFieldtime from (select * from field_standard) as T1, cust_field where mid(T1.Fieldname, 1, len (cust_field.fieldblockname)) = cust_field.fieldblockname group by fieldblockname, caseID, IwerName) group by fieldblockname, IwerName*

f. Custom Reports – User Defined Sections

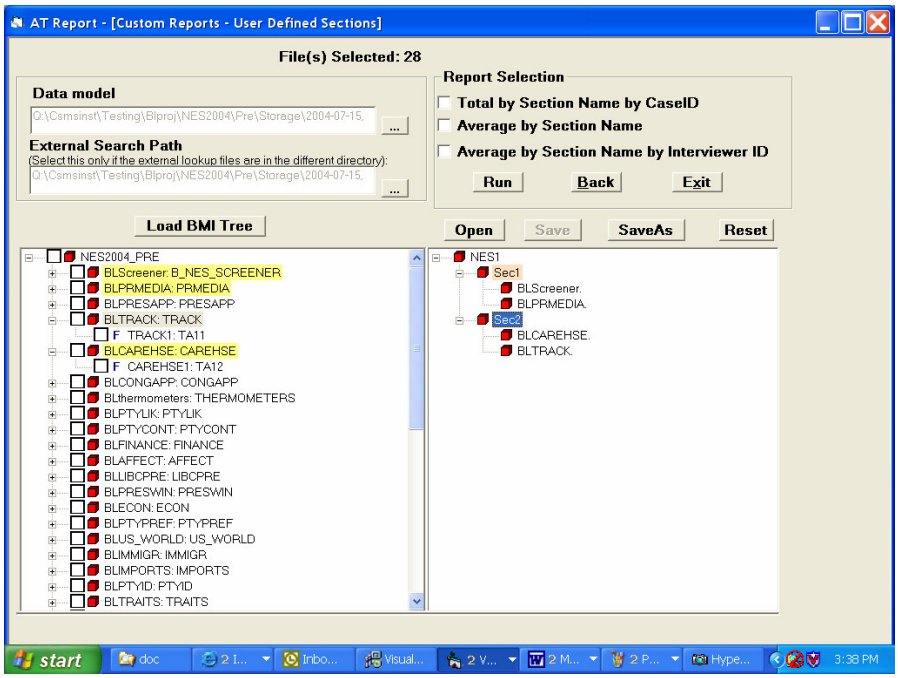


Figure 5 – Custom Report- Section Selection

The Custom Reports – User Defined Sections portion of the application looks and functions in a somewhat similar manner to the Custom Reports – Selected Fields/Blocks component described in the previous pages. In addition to allowing a user to specify which fields and/or blocks to include in a timing report, this portion of the application allows a user to group fields and/or blocks into sections defined by the user. For example, if a user is interested in generating timing data for a portion of a block rather than the entire block, this portion of the application will allow that. Likewise, a user can define a section to include items from more than one block.

See Figure 5. After setting up the sections, it is ready to run reports. First it will insert Section name and Blocks/Fields name into Cust_Section table.

The insert SQL Statement: *Iinsert into cust_Section values ('" & tempnode1.Text & "', '" & tempnode2.Text & "')*

- Total by SectionName by CaseID—This report provides the total amount of time (in seconds) spent and the number of visits made by an interviewer in each user-defined section. The data are sorted in CaseID order for each of the specified blocks/fields
  Basic SQL Statement: *Select Cust_section.SectionName as sectionname, temp.caseID as caseID, sum(temp.Fieldtime/1000) as sumoffieldtime from (Select * from field_standard) as temp, cust_section where mid (temp.fieldname, 1, len(cust_section.fieldblockname)) = cust_section.fieldblockname group by sectionname, caseID*

- Average by Section Name—The report lists the average time (in seconds) needed to complete each of the user-defined sections. The report also lists the number of cases that had timing data for the section.

  Basic SQL Statement: *Select SectionName, avg (sumofFieldtime), count (caseID) from (select Cust_section.SectionName as sectionname, temp.caseID as caseID, sum(temp.Fieldtime/1000) as sumoffieldtime from (Select * from field_standard) as temp, cust_section where mid (temp.fieldname, 1, len(cust_section.fieldblockname)) = cust_section.fieldblockname group by sectionname, caseID) group by sectionname*

- Average by Section Name by Interviewer ID-- For each section specified the report lists the average time (in seconds) needed by each interviewer to complete the particular section. A count of the number of cases completed by each interviewer is provided. The data are sorted in interviewer id order for each of the specified sections.

  Basic SQL Statement: *Select IwerName, Sectionname, avg(sumoffieldtime), count(CaseID) from (select Cust_section.SectionName as sectionname, temp.caseID as caseID, IwerName, sum(temp.Fieldtime/1000) as sumoffieldtime from (Select * from field_standard) as temp, cust_section where mid (temp.fieldname, 1, len(cust_section.fieldblockname)) = cust_section.fieldblockname group by sectionname, caseID, IwerName) group by sectionname,Iwername*

g. Other Report Features

There are several other useful report features:

- A user can then choose to print the output or export the output to a Text (*.txt), Excel (*.xls), or PDF (*.pdf) file.

- A user can save his customer report settings to xml files that can be opened later by the same user other users

- A user can choose different time format that are presented on the reports.

- A log file is produced if there are problems with the Audit Trail files.

7. Audit Trail data Generation

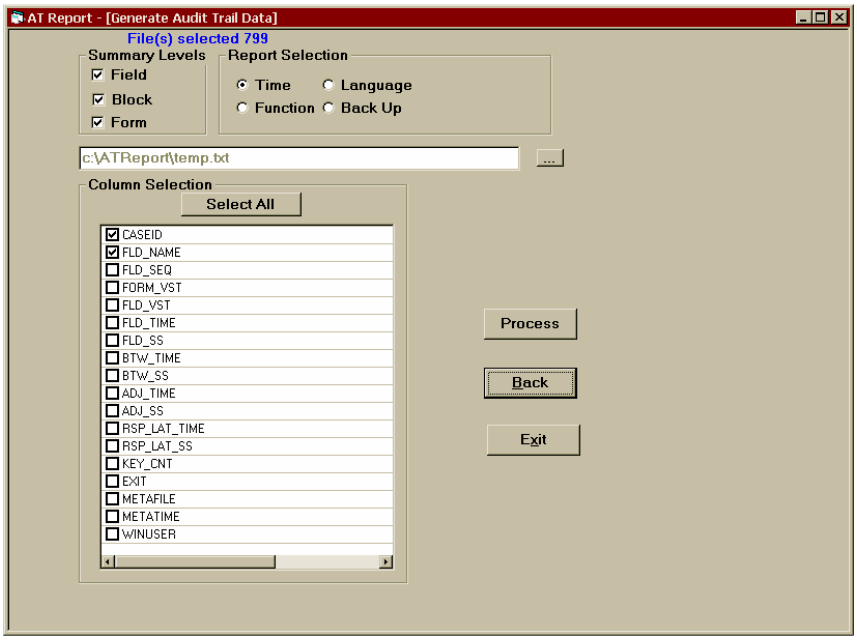


Figure 6 – Audit Trail Data Selecton

This part of program can generate four Audit Trail Data files:

a. Time File

This file summarizes times and key counts.

Columns:

CASEID! FLD_NAME! FLD_SEQ! FORM_VST! FLD_VST! FLD_TIME! FLD_SS! BTW_TIME! BTW_SS! ADJ_TIME! ADJ_SS! RSP_LAT_TIME! RSP_LAT_SS! KEY_CNT! EXIT! METAFILE! METATIME! WINUSER

b. Function File

This file summarizes important functions captured in the audit trail and key stroke files.

This file has a unique record for each CASEID and FLD_NAME combination. If there is more than one record for a field for a CASEID, then the data is collapsed/summarized by FLD_NAME.

Columns:

CASEID! FLD_NAME! VISITS! SUSP! ABSUSP! TOTSUSP! C_X_SUSP! EXIT_ACTION! REMCLK! REMCHNG! QHELP! BLAISEHELP! ERROR_ESC! ERROR_SUPP! ERROR_JUMP! MEDIA_START! MOUSE_CLICK! F1! F2! F3! F4! F5! F6! F7! F8! F9! F10! F11! F12

c. Language File

This file summarizes important language related actions in the audit trail and keystroke files. The development of this file is not complete. The fields we have to date as well as the incomplete fields are described below.

The file has a record for each visit to a field.

Columns:

CASEID! FLD_NAME! FLD_VST! NEXTLANG! PREVLANG! SETLANG! C_L_SETLANG! ENTER_LANG! EXIT_LANG

d. Back Up File

This file summarizes important information about movement up and down in the instrument.

The file has a record for each visit to a field.

Columns:

CASEID! FLD_NAME! FLD_VST! ENTER_VALUE! LEAVE_VALUE! LEAVE_CAUSE

e. Reading the output files

The files produced in this part of the program can be read by most spreadsheet programs (e.g. Excel), database programs (e.g. Access), SAS and SPSS. We are not going discuss this further in this paper.

8. Conclusion

This is the first version of the program. Many types of users at the University of Michigan are using this tool frequently for various purposes. After collecting feedback from these users, the UM programming group will make improvements to the program. Minor changes may be needed for the program to work with the output generated by the Blaise 4.7 Audit Trail DLL.

9. References

Statistics of Northlands Blaise Guide
Hansen, S. and Marvin T., (2001) "Reporting on item times and key strokes from Blaise Audit Trails", 7th International Blaise Users Conference

Converting Translated Blaise System Texts To A New Version

Rob Groeneveld, Statistics Netherlands

1. Introduction

We assume a basic familiarity with Blaise System Text translations. A brief example:

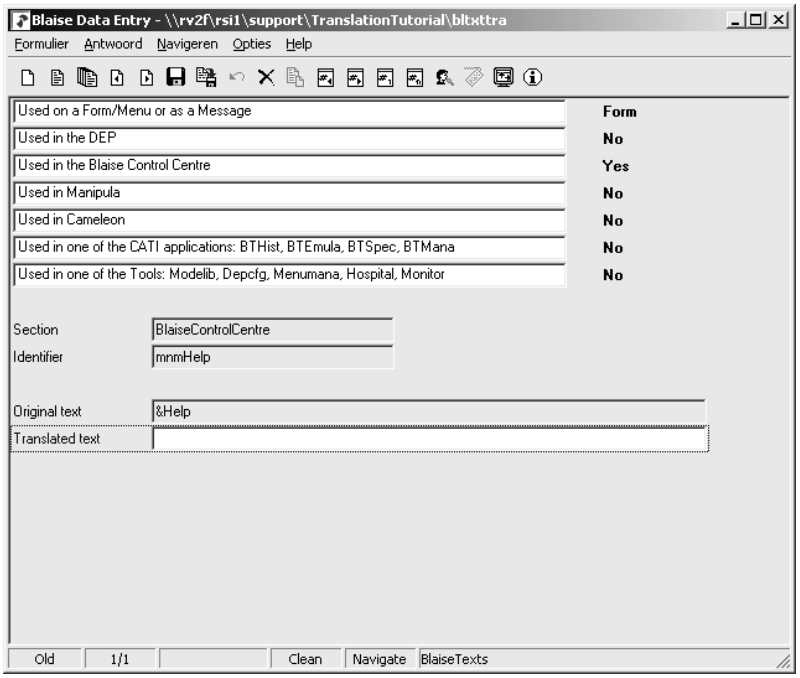


Figure 1. Entering translations

The above image is central to the translating process. We have a section (here, the Blaise Control Centre) and an identifier (here, mnmHelp) which together define a screen element appearing in a particular situation in Blaise. The original text is &Help in English and the Translated text can be typed in another language (in French, you would type in '&Aide'). More information about translations can be found in the Blaise Help file.

2. A New Translation

A Maniplus application was developed which guides you through the translation process. The opening dialog of the main Maniplus setup BlTranslat is:

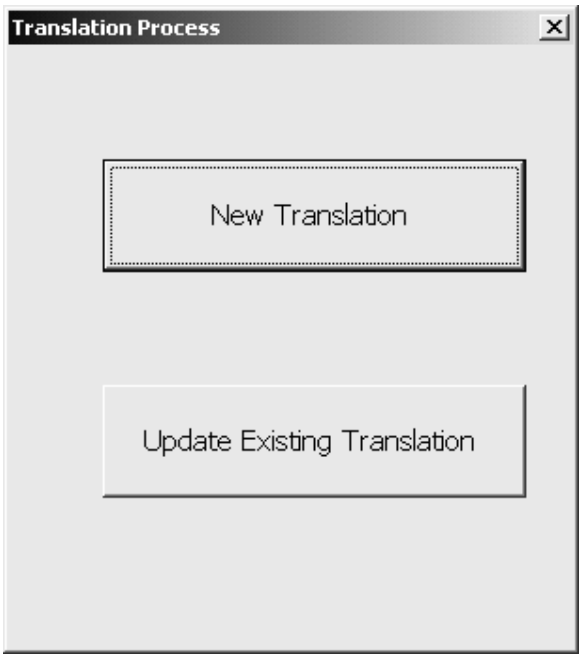


Figure 2. The starting dialog for the translation system

By clicking on the button 'New translation' the translation process is started. You will have to browse to the translation database (without translated texts at first) and start typing in translations, like in Figure 1.

When you exit from the data entry session, the question appears:

'Did you finish the translations and do you want to install them into Blaise?

If you answer Yes, this will initiate the transformation of the translation database to a language file in the INI format called BLAISE.TRA. The next step in the Maniplus setup is to copy the language file under the name of BLAISE.LNG to the folder where BLAISE.EXE is located. It may be impossible to access this folder because you have read-only rights in the folder. In that case, the registry setting must be changed as indicated in the Blaise Help or you must approach an administrator. The Maniplus setup does not provide assistance with changing the registry.
After installing the language file, when you start Blaise again, the translated texts appear.

3. Updating an Existing Translation

If you have previously translated Blaise system texts, you have a database BLTXTTRA.BDB with your previous translations. Now suppose you get a new version of Blaise with a new translation database, which is empty at first because it is shipped that way with the new version of Blaise. You want to copy your existing translations to the new database and work on translations of new elements in the Blaise system texts. On closer analysis, there are four categories of translations. The elements in the new database as compared to the old database can be:

- Added: elements which were not present in the old translation database because they represent new and extended possibilities in Blaise.
- Changed: elements which were there before but which have a different text.
- Removed: elements which were present in the old translation database but have been removed from the system.
- Copied: elements which were there before, do appear in the new database and are unchanged.

In order to facilitate the transition to the new database, the Maniplus setup offers the choice to update an existing translation (the second button in Figure 1). You have to browse to your old translation database, which must be present in a working folder of your own. It is advantageous to copy the new translation database to another working folder outside the Blaise installation folder (for instance, a subfolder NEW of the folder with the old translation database). Use the menu choice Database | Data File Management | Copy… from the Blaise control centre to do this. Don't forget to copy the meta-information files BLTXTTRA.BMI and BLTXTTRA.BXI as well (using Windows Explorer) to both folders. The folder with the old translation database will be used as a working folder for the intermediate files. The Maniplus setup compares the old and new databases and produces four Blaise databases with a structure identical to the translation databases: Added.bdb, Changed.bdb, Copied.bdb and Removed.bdb. The translator need only work on the Added and Changed texts. After clicking on the button 'Update existing translation' (see Figure 2), a menu is presented:

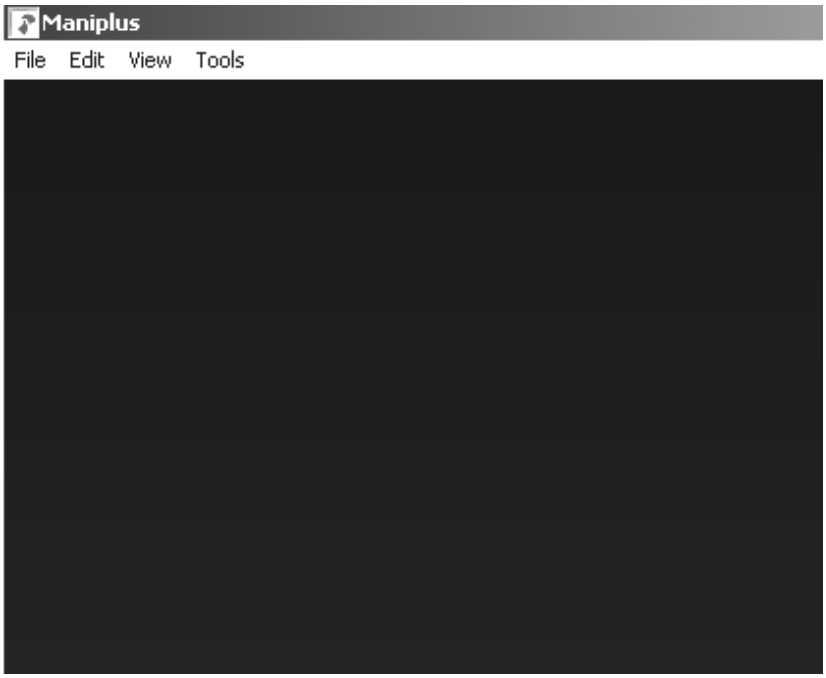


Figure 3. The menu in case of an update

The menu choices are as follows:

File	Edit	View	Tools
Open OLD Database	Changed Texts (4)	Changed Texts File (2)	Compare databases (1)
Open NEW Database	Added Texts (5)	Added Texts File (3)	Apply translations (6)
Exit Alt+F4		Removed Texts File	
		Copied Texts File	

Table 1. Menu Choices when updating an existing translation

Between parentheses the suggested order is indicated. But it is possible to interrupt the translation at any time and come back to it later. We will now discuss the steps in the recommended order.

1. **Compare databases:** the old and the new translation databases are compared and the added, changed, removed and copied texts are put into separate Blaise files. If the comparison was done before, the files are present and a warning is shown. You can choose to repeat the comparison, replacing the Blaise files. You can also skip this step and choose another item from the menu.
2. **View the changed texts:** when choosing this option, the changed texts are transferred to a text file Changed.txt which is shown in a viewer. This can be used to inspect the changes easily, but it is also useful to print the file Changed.txt using the program Notepad or a similar program. One then has all the sections and identifiers on paper, with the English texts and room to write the translation in one's own language. A sample of the output appears below:

```
AskLic                    Caption                   Blaise Licence

AskLic                    lblGiveLicenceInfo        Please enter your licence information

AskLic                    lblLicenceNumber          Licence &key:

AskLic                    lblUserNumber             Licence &number:

Blaise                    mgMASaveMSF               Save MSU file

BlaiseControlCentre       mnmManipulaWizard         Manipula Set-up &Wizard
```

Figure 4. Viewing the Changed texts

Figure 5 shows an example of texts written on paper as preparation for a translation into another language (in this example, typing them in Changed.bdb).

AskLic	Caption	Blaise Licence
Blaise-licentie		
AskLic	lblGiveLicenceInfo	Please enter your licence information
Voer a.u.b. uw licentie-informatie in		
AskLic	lblLicenceNumber	Licence &key:
AskLic	lblUserNumber	Licence &number:
Blaise	mgMASaveMSF	Save MSU file
BlaiseControlCentre	mnmManipulaWizard	Manipula Set-up &Wizard
BlaiseControlCentre	mnmOnlineReference	&Reference Manual
BlaiseControlCentre	mnmPrint	&Print...
BlaiseControlCentre	mnmWindowTile	&Tile Horizontal
BlaiseGlobalOptions	ccxShowDetails	Sho&w
BlaiseGlobalOptions	ccxShowKeyfields	Include &key fields

Figure 5. Handwritten notes on the Changed texts

3. **View the added texts:** when choosing this option, the added texts are transferred to a text file Added.txt which is shown in a viewer. As before, this can be used to inspect the changes, but it is also useful to print the file Added.txt using the program Notepad or a similar program. One can then write the translations in one's own language on paper as preparation for entering them into the Blaise file Added.bdb.
4. **Edit the changed texts:** The translations can be entered into the Blaise database Changed.bdb containing the elements with the changed texts.
5. **Edit the added texts:** In the same way, the translations can be entered for the added elements.
6. **Apply the translations:** The translations are applied to the new translation database, the location of which you entered in a previous step. If the program does not know the location of the new database, it will be asked. Anywhere in the program, if you made a mistake in entering the location of one of the databases, you can correct the location from the File menu using File | Open OLD database or File | Open NEW database. The process to activate the translations is the subject of the next paragraph.

4. Applying the translations and updating the language file

After step 6, the database with the translations has been updated and the question appears:

'Did you finish the translations and do you want to install them into Blaise?

If you answer Yes, this will initiate the transformation of the translation database to a language file in the INI format called BLAISE.TRA. The next step in the Maniplus setup is to copy this language file under the name of BLAISE.LNG to the folder where BLAISE.EXE is located. It may be impossible to access this folder because you have read-only rights in the folder. You can approach an administrator for this.
You can also change the registry setting for the language file as indicated in the Blaise Help. The registry entry can point to any file location you have access to. The Maniplus setup does not provide assistance with changing the registry.

5. The remaining menu entries

The menu entries not covered so far are: 'View removed texts file' and 'View copied texts file'. They are there for information only. With them, the files Removed.txt and Copied.txt can be viewed and printed like the Changed and Added text files. Only the Copied texts are processed in a later step (they are the translations of the unchanged elements and are simply copied to the new translation database).

6. Conclusion

We presented a Maniplus project helping in the translation and especially in the updating process, when a translator is developing and maintaining a different-language version of the Blaise system texts. This project organises the different tasks and enables the translator to keep an overview of the different steps necessary to accomplish an update from an earlier version of his translations, even if he skipped translating some versions of Blaise after the previous translation. Moreover, the translation work can be interrupted and resumed later.

Appendix

Technical notes on the Maniplus project

BlTranslat.man. The main file is BlTranslat.man. In order to run the project, this file must be run from the Blaise Control Centre or with the command

[path]Manipula.exe bltranslat

The Maniplus file BlTranslat.man presents the starting dialog (Figure 2) and declares the procedures ExportTranslation and TranslateNew, both of which are used when a new translation is made. When an existing translation is to be updated, i.e., the second button is pressed, the Maniplus setup ExistTrlMenu.man is called.

ExistTrlMenu.man presents the menu shown in Figure 3 and Table 1 and calls the procedures and setups for the various menu choices.

CreateMU.inc declares a procedure which creates the menu in the standard Maniplus way using a TEMPORARYFILE. The procedure MakeMenuLine is defined in ExistTrlMenu.man.

MenuDescription.bla defines a datamodel for Maniplus menu files. The included file Plusmenu.inc defines the function keys available to menus in Maniplus.

ExportTranslation.inc contains the procedure ExportTranslation which is called from BlTranslat.man and from ExistTrlMenu.man.

CARC.man divides the records in the new translation database into **C**hanged, **A**dded, **R**emoved and **C**opied records in corresponding Blaise database files (the auxiliary databases). This categorisation is done by comparing the old and new translation databases. The datamodel for all Blaise files involved in the translation process is defined in BLTXTTRA.BLA.

PrintAuxDb.man prints the contents of the chosen auxiliary database. It produces text files like the one in Figures 4 and 5. It uses the datamodel in TextFiles.bla.

Bltxt2in.man writes the contents of the translation database as a file in INI format called Blaise.tra. When copying this file to the Blaise system folder with the procedure ExportTranslation, the file name is changed to Blaise.lng.

Automatic Customization of Datasets Using Cameleon

Barbara S. Bibb & R.. Suresh, RTI International

1. Overview

Cameleon is the utility that extracts information from Blaise databases and data models. The default settings and options of Cameleon are usually sufficient for simple questionnaires in Blaise. For complex questionnaires which use nested blocks or other powerful features within Blaise, the default output of Cameleon would require additional effort on the part of the programmer to provide mapping of the variables to the questions or recoding of variables for ease of analyses. But there are additional features and constructs in Cameleon that allow information within the Blaise data model to be queried and manipulated so as to generate code in languages like SAS. These generated programs can provide the mappings or recode variables as desired. This paper provides the details for two such programs.

2. Converting SET type responses to toggle variables

Blaise SET type responses are stored in arrays. Each answer category selected is stored in an array element in the order it is collected. This method of data storage preserves both the response given and the order in which it was entered or chosen. However, during analysis, to know whether a response was chosen at all, each array element must be checked for the particular response of interest. It is easier for the analysts if there is a toggle variable for each of the responses to the SET question. This Cameleon procedure generates a SAS program to define and populate sets of toggle variables for each SET question.

There are Cameleon constructs that provide information about the SET variable properties. For example, NUMBEROFCHOICES gives the number of answer or response choices the respondent can choose for each SET variable. ANSWERCODE identifies the current response and NUMBEROFANSWERS provides the number of answer categories that exist for that particular item. By using other Cameleon constructs such as the TYPE and LOOP commands, the programmer can leverage the Cameleon tool to create and manipulate code.

Using this SET variable example, consider a new array of toggle variables. These variables will be generated and populated so that there is one variable for each possible answer category. The value of this variable is TRUE if the response was chosen and stored in any of the array elements for that item in the original Blaise database. This new variable is FALSE if the response was not chosen.

By default, the size of SET variable arrays in Blaise is determined by the number of answers the respondent can enter (NUMBEROFCHIOCES). Whenever the number of answers allowed is less than the NUMBEROFANSWERS or actual categories, adjustments must be made so that the correct number of toggle variables is created. That is, the newly produced code must generate more variables than exist in the original data model.

The Blaise software package includes a number of *.CIF (Cameleon Translator) files. These files are Cameleon code files that produce basic code needed to convert the Blaise data models to various software applications like SAS, SPSS, Oracle, etc. These very useful files can be enhanced to produce alternate or modified code that may be more useful under certain circumstances.

The example shown starts with the standard SAS.CIF. Code has been added to the SAS.CIF to generate a length statement for each of the newly created toggle variables. These variables are filled with a 1 for “category chosen”. Each toggle variable represents an answer choice. Additionally, both format and label statements are generated for these toggle variables. This

code will convert the Blaise ASCII data to a SAS dataset with the desired nonstandard manipulations and will provide SAS formats and labels for these newly created variables.

2.1 Step 1

To [PROCEDURE WriteAllTypes], code has been added to generate a value statement for the SAS PROC FORMAT. The loop will execute for each variable of type = SET.

```
[if type = SET then]
  [/]VALUE rcTE_[ENUMTYPENUMBER]F
    [:2]-1='Dontknow'
    [:2]-2='Refusal'
    [:2] 1='Category selected'
  [];
```

2.1.1 Result 1

The initial value statement shown below is output as a result of the original SAS.CIF. A newly generated value statement for the data that will be recoded is created by the addition of the new loop in STEP 1.

```
VALUE TE_1F
  1='blue'
  2='green'
  3='yellow'
  4='purple'
  5='white'
  6='black'
;

VALUE rcTE_1F
  -1='Dontknow'
  -2='Refusal'
   1='Category selected'
;
```

2.2 Step 2

The [PROCEDURE DEFINEALLTOGGLES] is created to generate a length statement for the new toggle variables.

```
[PROCEDURE DEFINEALLTOGGLES]
[BLOCKPROC]
  [FIELDSLOOP]
    [ARRAYLOOP]
      [IF TYPE = BLOCK THEN] [BLOCKCALL]
        [ELSE]        [IF TYPE = SET THEN]
            [FOR loopct := 1 TO NUMBEROFANSWERS DO]
             [:4]length[:2]rc[loopct][UNIQUENAME 8:]3.;
            [ENDDO]
          [ENDIF]

      [endif]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
```

```
[ENDBLOCKPROC]
[ENDPROCEDURE][*DefineAllToggles]
```

2.2.1 Result 2

The new toggle variables are named using the first array element name. Their number depends on the total number of answer categories (NUMBEROFANSWERS) available for the item.

```
length  rc1set11   3;
length  rc2set11   3;
length  rc3set11   3;
length  rc4set11   3;
length  rc5set11   3;
length  rc6set11   3;
```

2.3 Step 3

The [PROCEDURE WriteAllrecodes] sets the newly created toggle variables to 1 if the category corresponding to their value is in any position of the original Blaise variable array. It also sets the first toggle to "Don't Know" or "Refused" if that was the answer given by the respondent. The toggle variables do not retain any information about the order in which the categories were chosen. That information is available from the original SET variable array.

```
[PROCEDURE WriteAllrecodes]
[BLOCKPROC]
  [FIELDSLOOP]
    [ARRAYLOOP]
      [IF TYPE = BLOCK THEN] [BLOCKCALL]
      [ELSE]
      [IF TYPE = SET THEN][loopct:=1]
       [currsetfield:=UNIQUENAME]
       [SETLOOP][loopct2:=1]
        [answerloop]
        [thiscode:=ANSWERcode]
        [settxt:=''] [loopct3 :=1]
        [SETLOOP]
         [IF SETTXT = '' THEN]
         [settxt:= 'if ' + UNIQUENAME + ' = ' + STR(thiscode) + ' ']
         [ELSE][settxt:= SETTXT + ' or ' + UNIQUENAME + ' = ' + STR(thiscode) + '
'][ENDIF][loopct3 :=loopct3+1]
        [ENDSETLOOP]
         [if loopct = loopct2 then][settxt:= settxt + ' then']
         [:4][settxt 80:]
         [endif][loopct2:=loopct2+1]
        [endanswerloop]
         [:8]rc[loopct][currsetfield] = 1; [refcode:=thiscode+1][dkcode:=thiscode+2][if dkcode < 9
then][refcode:=8][dkcode:=9][elseif dkcode < 99
then][refcode:=98][dkcode:=99][else][refcode:=998][dkcode:=999][endif]
         [if loopct=1 then]
         [:4]else if [UNIQUENAME] = [DKCODE] then
         [:8]rc[loopct][currsetfield] = -1;
         [:4]else if [UNIQUENAME] = [refcode] then
         [:8]rc[loopct][currsetfield] = -2;
         [endif]
         [loopct:=loopct+1]
```

```
      [ENDSETLOOP]
      [topnum := NUMBEROFANSWERS]
      [while loopct < topnum do]
       [SETLOOP][loopct2:= 1]
       [answerloop]
       [thiscode:=loopct]
       [settxt:=''] [loopct3 :=1]
       [SETLOOP]
        [IF SETTXT = '' THEN]
        [settxt:= 'if ' + UNIQUENAME + ' = ' + STR(thiscode) + ' ']
        [ELSE][settxt:= SETTXT + ' or ' + UNIQUENAME + ' = ' + STR(thiscode) + '
'][ENDIF][loopct3 :=loopct3+1]
       [ENDSETLOOP]
        [if loopct2 = NUMBEROFANSWERS then][settxt:= settxt + ' then']
        [:4][settxt 80:]
        [endif][loopct2:=loopct2+1]
       [endanswerloop]
        [:8]rc[loopct][currsetfield] = 1;
        [loopct := loopct+1]

       [ENDSETLOOP]
      [enddo]
     [ENDIF]
   [ENDIF]
   [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*WriteAllrecodes]
```

2.3.1 Result 3

The following SAS code is produced.

```
if set11 = 1  or set12 = 1  or set13 = 1  or set14 = 1  or set15 = 1  or set16 = 1  then
      rc1set11 = 1;
   else if set11 = 9 then          /** set of ( 6 categories) variable**/
      rc1set11 = -1;
   else if set11 = 8 then
      rc1set11 = -2;
if set11 = 2  or set12 = 2  or set13 = 2  or set14 = 2  or set15 = 2  or set16 = 2  then
      rc2set11 = 1;
if set11 = 3  or set12 = 3  or set13 = 3  or set14 = 3  or set15 = 3  or set16 = 3  then
      rc3set11 = 1;
if set11 = 4  or set12 = 4  or set13 = 4  or set14 = 4  or set15 = 4  or set16 = 4  then
      rc4set11 = 1;
if set11 = 5  or set12 = 5  or set13 = 5  or set14 = 5  or set15 = 5  or set16 = 5  then
      rc5set11 = 1;
if set11 = 6  or set12 = 6  or set13 = 6  or set14 = 6  or set15 = 6  or set16 = 6  then
      rc6set11 = 1;

   if set21 = 1  or set22 = 1  or set23 = 1  then
      rc1set21 = 1;
   else if set21 = 9 then             /** set [3] of ( 6 categories) variable**/
      rc1set21 = -1;
   else if set21 = 8 then
      rc1set21 = -2;
   if set21 = 2  or set22 = 2  or set23 = 2  then
```

```
      rc2set21 = 1;
   if set21 = 3  or set22 = 3  or set23 = 3  then
      rc3set21 = 1;
   if set21 = 4  or set22 = 4  or set23 = 4  then
      rc4set21 = 1;
   if set21 = 5  or set22 = 5  or set23 = 5  then
      rc5set21 = 1;
   if set21 = 6  or set22 = 6  or set23 = 6  then
      rc6set21 = 1;
```

2.4 Step 4

The procedures ToggleWriteLabels and ToggleWriteFormats were added to the stream. They mimic the existing procedures WriteLabels and WriteFormats but address only “set” variables.

```
[PROCEDURE ToggleWriteLabels]
  [BLOCKPROC]
    [FIELDSLOOP]
      [ARRAYLOOP]
         [IF TYPE = BLOCK THEN] [BLOCKCALL]
         [ELSE]
            [if type = set then][catct:=1]
              [for loopct := 1 to numberofanswers do]
[if catct = 1 then][rangetxt:= '(1, blank,-1=REF,-2=DK)' ][else][rangetxt:='(1 or
blank)'][endif][cattxt:= ' cat ' + str(catct) + ' selected ' +rangetxt]
[:2]rc[LOOPCT][UNIQUENAME 8:] = '[COPY(FIELDLABEL, 1 ,
60)][cattxt][catct:=catct+1]'
              [enddo]
            [endif]
         [ENDIF]
      [ENDARRAYLOOP]
    [ENDFIELDSLOOP]
  [ENDBLOCKPROC]
[ENDPROCEDURE][*ToggleWriteLabels]

[PROCEDURE TOGGLEWriteFormats]
[BLOCKPROC]
  [FIELDSLOOP]
    [ARRAYLOOP]
       [IF TYPE = BLOCK THEN] [BLOCKCALL]
       [ELSEIF TYPE = ENUMERATED OR TYPE = SET THEN]
         [IF TYPE = SET THEN]
          [FOR LOOPCT := 1 TO NUMBEROFANSWERS DO]
          [:2]rc[LOOPCT][UNIQUENAME 8:]rcTE_[ENUMTYPENUMBER]F.
          [ENDDO]
         [ENDIF]
       [ENDIF]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*TOGGLEWriteFormats]
```

2.4.1 Result 4

Example code produced using these modified procedures is:

```
LABEL
  rc1set11    = 'code as many as there are cat 1 selected (1, blank,-1=REF,-2=DK)'
  rc2set11    = 'code as many as there are cat 2 selected (1 or blank)'
  rc3set11    = 'code as many as there are cat 3 selected (1 or blank)'
  rc4set11    = 'code as many as there are cat 4 selected (1 or blank)'
  rc5set11    = 'code as many as there are cat 5 selected (1 or blank)'
  rc6set11    = 'code as many as there are cat 6 selected (1 or blank)'
  rc1set21    = 'code only up to 3 cat 1 selected (1, blank,-1=REF,-2=DK)'
  rc2set21    = 'code only up to 3 cat 2 selected (1 or blank)'
  rc3set21    = 'code only up to 3 cat 3 selected (1 or blank)'
  rc4set21    = 'code only up to 3 cat 4 selected (1 or blank)'
  rc5set21    = 'code only up to 3 cat 5 selected (1 or blank)'
  rc6set21    = 'code only up to 3 cat 6 selected (1 or blank)'
;

FORMAT
  rc1set11   rcTE_1F.
  rc2set11   rcTE_1F.
  rc3set11   rcTE_1F.
  rc4set11   rcTE_1F.
  rc5set11   rcTE_1F.
  rc6set11   rcTE_1F.
  rc1set21   rcTE_2F.
  rc2set21   rcTE_2F.
  rc3set21   rcTE_2F.
  rc4set21   rcTE_2F.
  rc5set21   rcTE_2F.
  rc6set21   rcTE_2F.
;
```

2.5 Summary, example 1

Code from the standard SAS.CIF was modified and enhanced to produce modified SAS code. This new code generates a new set of variables that are used to determine if a category was chosen by the respondent as an answer to a SET question. These variables can be checked when answer order is irrelevant and the analyst is only concerned with whether or not the category was chosen. Automated code to do this task causes all the SET variables to be converted to toggles and the programmer need not be concerned about locating individual SET variables in a large complex questionnaire

3. Generating RENAME statements for potentially confusing variables

Questionnaires in Blaise can use nested blocks, SET questions, and other features that allow looping over blocks of questions. The field names within Blaise use “dot notation” for fully qualified path names. But when the data are extracted into SAS or other similar databases, Blaise generates variable names that are simply consecutive numbers. This essentially renumbers the variables from the original questionnaire and hence, requires the analysts to do additional work in mapping from Blaise generated variable names to analytical variable names.

Consider the example above that uses “set” variables. At the beginning of the sequence, the set variables are defined by Blaise for the item SET1 and named SET11, SET12, SET13, …SET16. If the questionnaire contained another item or question, called SET11, further down in the instrument, Blaise would assign it the next available variable number because SET11 has all ready been assigned by the data model.

To avoid confusion, final datasets should have variable names that correspond to the original item or question numbers. To achieve this end, a Cameleon cif can be used to extract the variable attributes from the Blaise datamodel. These variable attributes can be read and manipulated by alternate software to generate new code which will do the desired manipulations. For this example, variable information is read and RENAME statements are generated in SAS. The variables are renamed so that the variable names of the final data set correspond more closely with variable names of the questionnaire.

3.1 Step 1

By modifying the WriteAllFields procedure in the standard SAS cif, a *.txt file is generated that describes the variables in the datamodel. This procedure is outputting the UNIQUENAME, FIELDNAME, FIELDPATH, variable type (string/numeric), postion information, and various counts such as NUMBEROFCHOICES and NUMBEROFANSWERS.

```
[PROCEDURE WriteAllFields]
[VAR htxt ctxt: STRING  arryct SETCT: REAL]
[BLOCKPROC]
  [FIELDSLOOP]
    [arryct:=0]
    [ARRAYLOOP]
      [arryct := arryct+1]
      [setct:=0]
      [SETLOOP]
        [setct:= setct+1]
        [IF TYPE = BLOCK THEN] [BLOCKCALL]
        [ELSE]
          [IF TYPE <> OPEN THEN]
            [htxt:= UNIQUENAME]
            [ctxt:= '  ']
            [IF TYPE = STRING OR
               TYPE = DATE OR
               TYPE = TIME OR
               TYPE = CLASSIFICATION THEN]
               [ctxt:= ' $']
            [ELSE]
              [ctxt:= '  ']
            [ENDIF]
            [:4][htxt 11:][:2][ctxt 4:][FIRSTPOSITION :PositionWidth][:2][LASTPOSITION
:PositionWidth][:4][fieldname 20:][:2][arryct 3:][:2][setct
3:][:2][NUMBEROFCHOICES][:2][NUMBEROFANSWERS][:2][fieldpath]
          [ENDIF]
        [ENDIF]
      [ENDSETLOOP]
    [ENDARRAYLOOP]
  [ENDFIELDSLOOP]
[ENDBLOCKPROC]
[ENDPROCEDURE][*WriteAllFields]
```

3.1.1 Result 1

Output from the WRITEALLFIELDS modified code is:

```
set11     1  1  set1     1  1  6  6  set1[1]
set12     2  2  set1     1  2  6  6  set1[2]
set13     3  3  set1     1  3  6  6  set1[3]
```

set14	4	4	set1	1	4	6	6	set1[4]
set15	5	5	set1	1	5	6	6	set1[5]
set16	6	6	set1	1	6	6	6	set1[6]
set21	7	7	set2	1	1	3	6	set2[1]
set22	8	8	set2	1	2	3	6	set2[2]
set23	9	9	set2	1	3	3	6	set2[3]
single	10	10	single	1	1	1	6	single
set17	11	11	set11	1	1	1	2	set11
set18	12	12	set12	1	1	1	2	set12
set131	13	13	set13	1	1	1	6	set13[1]
set132	14	14	set13	2	1	1	6	set13[2]
set133	15	15	set13	3	1	1	6	set13[3]
set134	16	16	set13	4	1	1	6	set13[4]

3.2 Step 2

Read the above listing with SAS to generate RENAME statements. The UNIQUENAMES for the fieldname SET1 are set11 through set16. There is also a fieldname called SET11. So, when Blaise generates the UNIQUENAME for fieldname SET11, it assigns the name SET17. Similarly fieldname SET12 gets the name SET18. This can be very confusing for the analyst who needs to use the extracted datafile.

To be consistent with the original FIELDNAMES, the following RENAME was produced by manipulating the variable attribute listing using SAS. Each ”set” variable was renamed with an underscore and an s (_s). Each array variable was renamed using an underscore and an a (_a). The variables that were assigned deceiving UNIQUENAMES were renamed back to the original FIELDNAME. Comment code was added to display the fully qualified FIELDPATH clarifying the reason for the RENAME of the variable. Where FIELDNAME = UNIQUNAME, the rename is unnecessary and no code is generated.

3.2.1 Result 2

```
RENAME
      set11  = set1_s1 /**set1[1] **/
      set12  = set1_s2 /**set1[2] **/
      set13  = set1_s3 /**set1[3] **/
      set14  = set1_s4 /**set1[4] **/
      set15  = set1_s5 /**set1[5] **/
      set16  = set1_s6 /**set1[6] **/
      set17  = set11 /**set11 **/
      set18  = set12 /**set12 **/
      set131  = set13_a1 /**set13[1] **/
      set132  = set13_a2 /**set13[2] **/
      set133  = set13_a3 /**set13[3] **/
      set134  = set13_a4 /**set13[4] **/
      set21  = set2_s1 /**set2[1] **/
      set22  = set2_s2 /**set2[2] **/
      set23  = set2_s3 /**set2[3] **/
;
```

3.3 Summary, example 2

For this example, the SAS.CIF was modified to extract a number of variable attributes from a Blaise data model. SAS was used directly to read the variable attributes and generate SAS statements that further modify the SAS dataset. Here, a RENAME statement was generated to clarify the variable names and make them correspond more closely with the original Blaise

fieldnames. A further extension of this technique might be to produce a list of variables to drop from a dataset with a DROP statement.

4. Conclusions

Use of a combination of Cameleon tools and other software can generate code to map or recode variables to make the final dataset easier to use. Extracting data may require the mapping of questions and variables or specific types of recodes. Furthermore, variables can be created and the attributes can be changed or adjusted to satisfy specific requirements.
Automating these procedures is critical because

- ✓ the underlying questionnaire might change during development and production
- ✓ the questionnaire might have multiple SET type questions and/or nested blocks that are time consuming to identify manually
- ✓ mapping these fieldnames to analytical names manually is error-prone and these errors will be hard if not impossible to detect
- ✓ generating the toggles can be tedious and again, is error prone

The major benefit to automating these procedures is that the data is treated consistently and with fewer errors, if any. The savings in time and cost can be substantial and the end product will be robust and accurate.

5. References

Blaise for Windows 4.1 Cameleon Reference

6. Acknowledgements

The authors acknowledge input from various staff with the Research Computing Division of RTI International. The additional ideas of these people helped develop the concepts discussed in this document. The people involved are Lilia Filippenko, Gilbert Rodriguez, and Rita Thissen.

Error Reporting and User Interfaces for Post-collection Data Editing

Pavle Kozjek, Statistical Office of the Republic of Slovenia (SORS)

1. Introduction

At Statistical Office of Slovenia, Blaise is used for integrated data collection and editing of CAI surveys, as well as for editing of data from other sources (from high- speed data entry, secondary sources, follow-up of CAI surveys, etc.). The old interactive editing system (GVS-2000) running on a mainframe is no longer supported and is being replaced by Blaise. There are also some old Cobol and PL-1 data editing applications on the mainframe that also need to be replaced as soon as possible. They are not interactive and are organised as time consuming cyclic editing processes with printed lists of errors and re-entered corrections.

Although the GVS-2000 system is now outdated, it used to provide some good and functional solutions for data editing for many years, and users (data editing staff) were satisfied with it. In a new network environment, users expect at least all the functionality of the previous system. Since Blaise is mainly CAI oriented, some functionality needed for batch processing and post collection editing is not originally obtained by the system. The paper describes our approach how to add this functionality.

2. Data editing system on SORS LAN: some development issues

In the past, probably the most obvious problem of the data editing system at SORS LAN was the absence of standard generic procedures to obtain control and overview of the editing processes and to allow different types of users preparing, maintaining and using data editing applications in the same way. It was therefore absolutely necessary to define proper standards and conventions to avoid tailor-made solutions for each single survey. We were aware that for instance some CAI surveys (e.g. LFS, HES, etc.) can never completely "fit" into such a standardized system, due to their complexity and special needs, but the majority of other surveys can be placed there.

When developing a LAN data editing system, we followed the example and used the main principles of GEntry, a relatively successful in-house developed system for high-speed data entry. Blaise (version 4.4) was used for interactive and batch editing procedures, and VB for user interfaces. However, data editing is a much more complex process than high-speed data entry, so we were afraid to meet some unsolvable problems during development.

Some of the main expected characteristics of the new LAN editing system are:

- Standardized and transferable LAN environment for data editing (hierarchical system of folders on the server)
- Standard set of main tools (Blaise, VB, etc.)
- Generic and parameterized user interfaces
- Standard naming conventions of:
    - survey applications, based on a codelist of SORS surveys
    - generic programs (for instance; M*code* = DB of data for editing, A*code* = address register DB; *code* = survey code from SORS register of surveys)
    - fields in the datamodel. SORS's metabase (under development) is expected to have a very important role here.

- Standardised flow of data and metadata:

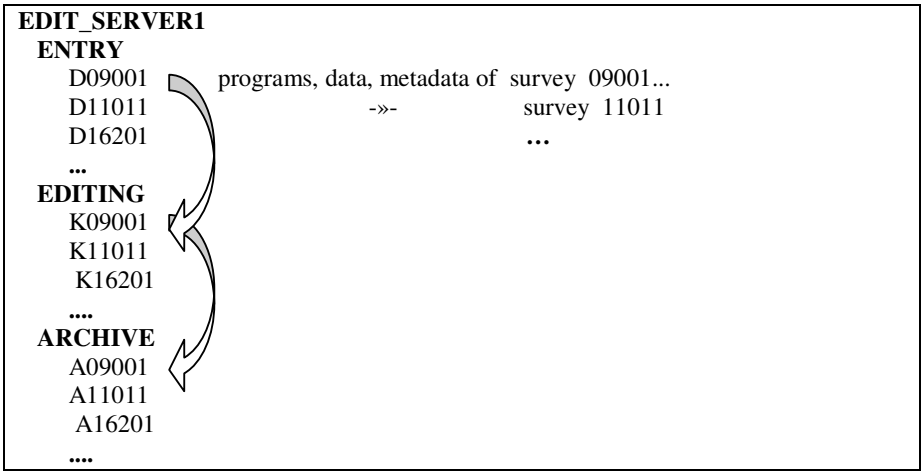


Figure 1: folder structure on server and basic data flow in the proces od fata editing

The ARCHIVE folder is intended for (on-line) storing data and metadata after editing on micro level. Data from this folder can be directly accessed by different users for further processing (statistical-macro editing, imputation, analysis, tabulation, etc.). In a similar way, raw data are archived before entering the editing process.

The speed and intensity of development was limited by the need to support regular statistical »production«, organize training and obtain support for the new staff working with Blaise: application developers and application users. Especially for some developers who were used to work in a mainframe environment, the changes were not easy: first, they had to adapt to environment and development on LAN, and then to study new tools. As the main (but not the first) data editing pilot project, the application was developed for the monthly Industrial Production Survey, which is a typical representative of a large group of surveys where some additional functionality was required. Some characteristics:

- Many and complex checks on a relative small number of fields
- Many reference files (often including data from more than one previous period)
- In some cases, mixing of microediting and statistical editing
- »Dynamic« reference files (derived from the last version of edited data)
- Adding new (additional) batches of raw data during the editing process

The Industrial Production Survey was prepared and tested for editing in Blaise already about five years ago, with Blaise III. But the production really started in January 2003, with Blaise 4 and VB interfaces, when environment on LAN at SORS became capable to support more complex processes.

3. Error handling and reporting issues

The Blaise system is mostly and traditionally used to support surveys, where data collection and editing are integrated in the process of computer assisted interviewing (CAI). It is based on interactive work of an interviewer.

When SORS decided to use Blaise as the main LAN data editing system, one of the frequently asked questions was if Blaise can obtain all data editing functionality of the old mainframe system. If something already used suddenly becomes unavailable, the users will not be very happy. The mainframe GVS-2000 system, based on Rapid RDBMS, was a kind of reference, with the relatively good and automated coverage of complete survey data editing process. Improved error handling and reporting for post-collection data editing on LAN was one of the first and most important needs. Blaise itself is very much oriented to interactive work, so we added some solutions developed fot the needs of post-collection data editing:

- An array of Boolean values was added to each editing data model to save and handle each single error
- Each error in the survey editing application was numbered with its unique error code to optimize searching, maintenance and documentation, and to report data correctness at any moment
- A list of errors (checks and signals) is reported on screen constantly and independently of cursor position, using DEP watch window
- A number of standard batch templates were prepared, to generate and show different reports about errors

With error codes added (as an array of fields) into the survey datamodel, batch reports and analyses of errors in data are quite easy. Some more time was needed to find a good solution for error reporting in interactive mode. Here is an example of the Blaise screen with a list of errors in DEP watch window for survey M09001 – Industrial Production:

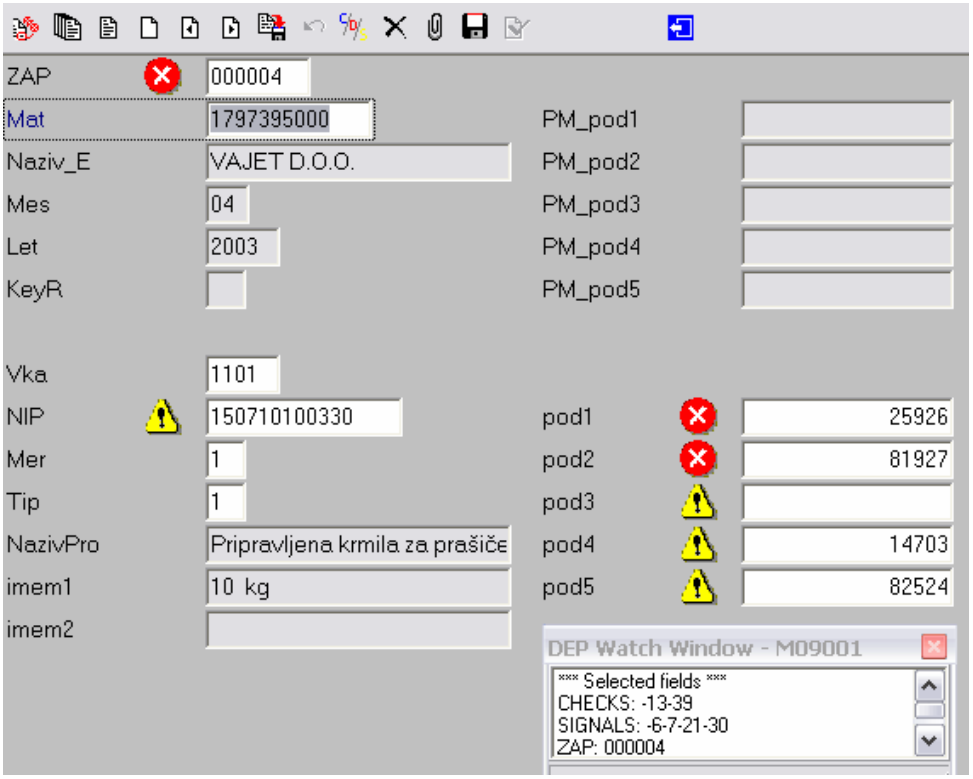


Figure 2: Data editing screen with list of errors

Values of auxfields CHECKS and SIGNALS (list of error codes) and field ZAP (primary key) in DEP watch window are calculated with every press of Enter, so a data analyst can see the effect of corrections immediately and at one place. There are also other settings and options (layout, hot keys, etc.) adapted to this mode of data editing.

With these additions, error reporting was already much closer to user's needs. Nevertheless, we still could not find an easy way to search the database for specific errors in an interactive way. Instead of this, a generic batch module was prepared. A user enters a selected error code into interface and Manipula setup produces a list of units with this error, and brings the report to the screen. Users adapted to it, but from the point of view of functionality it is not completely satisfying, and we are still looking for a simple interactive solution.

4. User interfaces

Generic user interfaces were found as one of the most important components of the new SORS data editing system. They are key for access and control of the complete editing process. They were designed in permanent cooperation between their developers and users. We followed the principles that the interface should be self-descriptive and as simple as possible for use and maintenance.

A special **interface for application developers** is usually not a part of the system, since developers are supposed to use tools directly from their environment. But in our case, when developers are still learning their new tools and new environment, the interface for development was found extremely helpful and useful. Through the menu system it supports:

- Direct access to development environment, based on parameters (code of survey, developer, etc.)
- Copying all templates and standard setups (modelib, depmenu, etc.) to survey development folder, to begin the new application
- Generated import and export of data
- Implementation - including the application into the production environment

Probably this interface will become obsolete (or less important) when developers become more familiar with the new environment. But it has proved to be a good start for new developers.

Different is the generic **interface for application users** (analysts-data editing staff) which is a normal part of the production system. The current version already covers almost all needs of users of about 40 surveys. Almost - because with few surveys there are still some very special processes, which can not be triggered by buttons in the interface. But at the end, with complete overview of task and functions in editing processes they will be included as well.

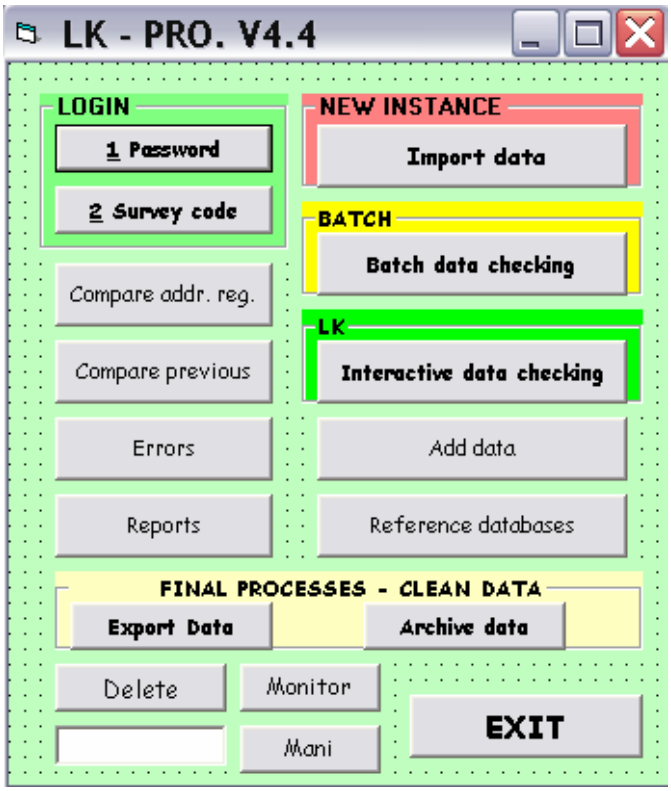


Figure 3: Data editing user interface

Access to survey applications is defined by parameters entered into the user interface: password, survey code, period (instance) of survey, names of reference databases, etc. The password also defines the level of user: survey administrator or data editor-analyst. Some buttons (e.g. import data, archiving) are active for administrator only. Other parameters (e.g. /g - get form; /P2 – data editing page layout) are always fixed. After the login procedure, the user begins with importing ASCII files (data, references, etc.) into Blaise, runs batch checking and then continues with interactive review and correction. Blaise (and other Windows) executables are called through the VB Shell function, using Blaise command line parameters. Example:

```
Shell (potDEP & potLK & "M" & SCode & " /g /p2 /t2 /Ydsn  /e /!" & potLK & "
/m" & potLK & "depmenu.bwm"), 2
```

Figure 4: Commands behind Interactive data editing button

Figure 4 shows commands behind the *Interactive data editing* button: call of survey data editing application in user environment. Parameters: *potDEP* is path to DEP, *potLK* is path to the survey data editing folder, and *SCode* is the numeric survey code. Get form mode, data editing page layout and data editing - dynamic checking (/g /p2 /t2) are default settings in the data editing system, and clean records are not brought on the screen (Ydsn). Depmenu and modelib files are designed for the common needs of data editing and can be adapted for the single survey if necessary.

In practice up to 20 users (data editors) work interactively on a survey database, editing their own part of data defined by key values. We tried to avoid mixing batch and interactive processes as much as possible, so batch checking is performed on complete database at the beginning, and later only when necessary.

It is very important that with this user interface we are able to cover different combinations of data entry and editing, manipulate and transform between different types of data models (record is form or record is row in a table: transformation is generated) and (only by entering different parameters) support data editing on different servers and locations.
This is enabled since the complete editing procedure is defined as a transferable module, which can be used at any place in the statistical information system.

5. Conclusions

The system for post collection data editing at SORS is successfully replacing the old mainframe data editing system. Error reporting and smooth access to the system, obtained by user interfaces, are important components of the integrity, and both were the result of in-house improvements, enabled by knowledge and experience from the past years. I have to say that some hints from our Blaise colleagues from other countries were also very helpful.
At the moment most important functions are covered and new surveys are "added" mostly without the need for any changes in the system. This functionality allows us to choose between modes of data editing in some surveys: integrated or separately, after data capture. This also means better flexibility and better availability of human resources: a job can be spread over different data entry and editing departments. However, what is most important is that the system succeeded to make the first steps to establish common in-house standards of development and usage of data editing applications.
Parallel with this, we try to build a similar system for CAI surveys, which is not an easy job, since it is hard to put so different surveys in a shell of standards and conventions. And with both systems, the question of efficient and automated use of metadata is still opened.

6. References

- Edgar, M., and Turner, M. J., Software Quality Framework, Meeting on the Management of Statistical Information Technology, Geneva, 17-19 February 2003
- Katic, M., Klanjscek, M., and Kozjek, P., Management and Monitoring of the Statistical Process and Impact on Data Quality, Meeting on the Management of Statistical Information Technology, Geneva, 17-19 February 2003
- Kozjek, P., Blaise Generator for High Speed Data Entry Applications, Proceedings of the 6th International Blaise Users' Conference, Cork, 2000
- Kozjek, P., Blaise for Post-collection Data Editing - Building general data editing system based on Blaise, Proceedings of the 8th International Blaise Users' Conference, Copenhagen, 2003
- Pierzschala, M., and Manners, T., Revolutionary Paradigms of Blaise, Proceedings of the 7th Blaise User's Conference, Washington D.C., 2001
- SORS and Statistics Sweden, Feasibility study on the architecture of information systems and related equipment issues, Study implementation and Hardware/Software Specification for Tendering, September 1997
- Sundgren, B., An Information Systems Architecture for National and International Statistical Organizations, CES/AC.71/1999/4, Meeting on the Management of Statistical

The new German Microcensus starting in 2005, organisational and technological aspects, the use of Blaise

1. Introduction

Roughly one percent of the German population (roughly 800,000 persons) is surveyed each year in the context of the German Microcensus. The goal of the statistics thereby gathered is to obtain extensive information on the structure of the population, on family contexts and new forms of co-habitation, on the economic and social situation of households, on schooling and vocational training, on the nature of the professions exercised, on the living conditions, on health, etc. The questionnaire is very comprehensive, complex, and many plausibility tests are carried out. The selection of the persons surveyed takes place according to a mathematical sample procedure in which the basis of the random selection is an area sample. Each household selected is surveyed in four consecutive years, the questionnaire also being fully completed once more for the second to fourth rounds of questioning.

The Microcensus is the most important set of German statistics in which BLAISE is used.

The survey is carried out by interviewers, but respondents may complete a classical questionnaire by themselves and mail it to a Statistical Office. The interviewers enter the answers in a classical paper questionnaire or in a Blaise questionnaire on a laptop. Both variants are possible until the 2004 survey. The classical questionnaires are collated in a Blaise program in the Statistical Offices. There are no telephone questionnaires; Internet questionnaires are not yet being used.

The questioning has so far taken place once yearly on a specific date; in most cases, this is a date in May. The interviewer gets the list of addresses (his/her jobs) once per year in the shape of a paper list. The Blaise application both for the laptops and for the Statistical Offices is put together in the Federal Statistical Office, sent once per year to the Land Statistical Offices, where it is installed on the interviewers' laptops. The interviewers store all the surveys on their laptops. At the end of their work, they return the laptops to the Land Office. There, the data are assembled. This procedure has all the well-known technical and organisational disadvantages.

A new Microcensus Act will apply in Germany from 2005 onwards. The interviews will no longer be implemented on a specific date, but spread over the whole year. Also, the list of questions can change during the year. Results and evaluations must be available in advance, so-called rapid notifications are called for. The interviewers receive their jobs twice monthly, and also have to send the completed interviews directly to the Statistical Office. The Act forces the Microcensus to go "online" in terms of technology.
The following are the most comprehensive changes and tasks which have to be completed by January 2005:

- Almost all interviews will be carried out by laptop. Many more laptops will be used, will have to be managed and maintained; many more electronic interviews must be processed and combined.
- The data will be sent daily via data telecommunication from the interviewer to the Land Office.
- The interviewer receives new program updates and new data models by data download; these have to be installed and managed. Data conversions must be carried out.
- The interviewer receives his/her jobs via data telecommunication; certain management and administrative functions, such as information on remuneration, will be carried out online.

These comprehensive changes and additions will naturally be introduced gradually, first the electronic data transfer, the transfer of the jobs and the update function. The management and administration functions will be added later. We intend to do this via the update function.

2. Experience and solutions of other countries

The problems and tasks are not new; similar challenges have to be solved with many sets of statistics, and extensive software packages already exist in other countries. A demonstration of the solution used in the Netherlands was very helpful for us. Especially the suggestion to have a clear separation between logistics (interviewer management) and statistics (the questionnaire), was very important. We were able to take a great number of tips from the essay "Automatic Upgrade of Production Blaise Instruments" from the IBUC 2003 in Copenhagen. A good project structure is very helpful for automatic updates on a laptop with automatic data conversion. I put forward ideas for this in Washington in 2001: "Organisation of an extensive BLAISE project". Also in the context of our development environment for the Blaise application, we had developed some tools which were highly helpful to realise automatic updates.
Because of special organisational contexts in Germany, in particular because of the federal structure, we have unfortunately not been able to take on all the abovementioned solutions.

This short essay indicates a few design-related decisions which are more organisational and less technical in nature, and which need to be taken in the concept and introduction of an online survey using laptops, and sketches the solutions we achieved. Special reference is made to the use of Blaise.

The interviewers we employ have very little knowledge of how to use laptops; it must be possible to operate the application with no technical understanding.

3. Program updates, data conversion on the laptops

If it is necessary because of error correction or because of changes and additions in the question program to alter data models in the production operation, the question arises as to whether it is better to convert the data on the laptop or to send them to the server, convert them there and send them back with the new model. The organisational effort for the server solution appeared to us to be too big, and we hence carried out conversions on the laptop. Greater security and the possibility to intervene would favour the server solution.

Our laptop application is structured as follows:

```
Microcensus - Blaise ---------------- Sequential control
                                |
                                |--------- Blaise – MC
                                                |
                                                |--------- Logistics
                                                |--------- Questionnaire
                                |
                                |---------- Blaise-MC-temp
                                |
                                |---------- Blaise – MC – V001
                                :
                                :
                                |---------- Blaise – MC – Vn
                                |
                                |---------- Zip – Archive
```

Sequential control: Here one finds all objects for the update and data conversion programs, protocols and interfaces – data of the update programs for the Blaise application.

Blaise-MC: The current application! The work is done in this directory.
The directory is strictly sub-divided into the areas logistics and questionnaire.

Blaise-MC-temp : Directory for the implementation of the updates and the data conversion

Blaise – MC – V001 to Blaise – MC – Vn : Backup directory of the individual versions so that in the event of an error it is possible to work in the backup version. The version management of Blaise is used.
Zip – Archive: Directory to "unzip" the download file with the updates!
Some compression programs which support the .zip format are not secure. Errors in the directory of the Zip archive are not recognised. For this reason, we send a file with all filenames and paths and examine whether all the files have really been transferred and unzipped.

It is certainly also possible always to work in the directory of the version, but one then has to copy it prior to an update. Always working in the Blaise-MC directory has no logical but practical advantages. The data conversion may not be repeatable; in case of an error one has to pay attention here. There are several possibilities at this point; for us maximum security was the most important thing.

Since data models get new fields and are added to, it only makes sense to implement the updates in the order of the envisioned versions. We have not yet provided for a direct jump from version V001 to V005, as described in the abovementioned IBUC 2003 essay.

The update should be implemented such that the interviewers can always continue to work with all conceivable errors. This was a moot point; one could also put forward the opinion that if an error has occurred the interviewer does not carry out any interviews until the correction has been carried out.

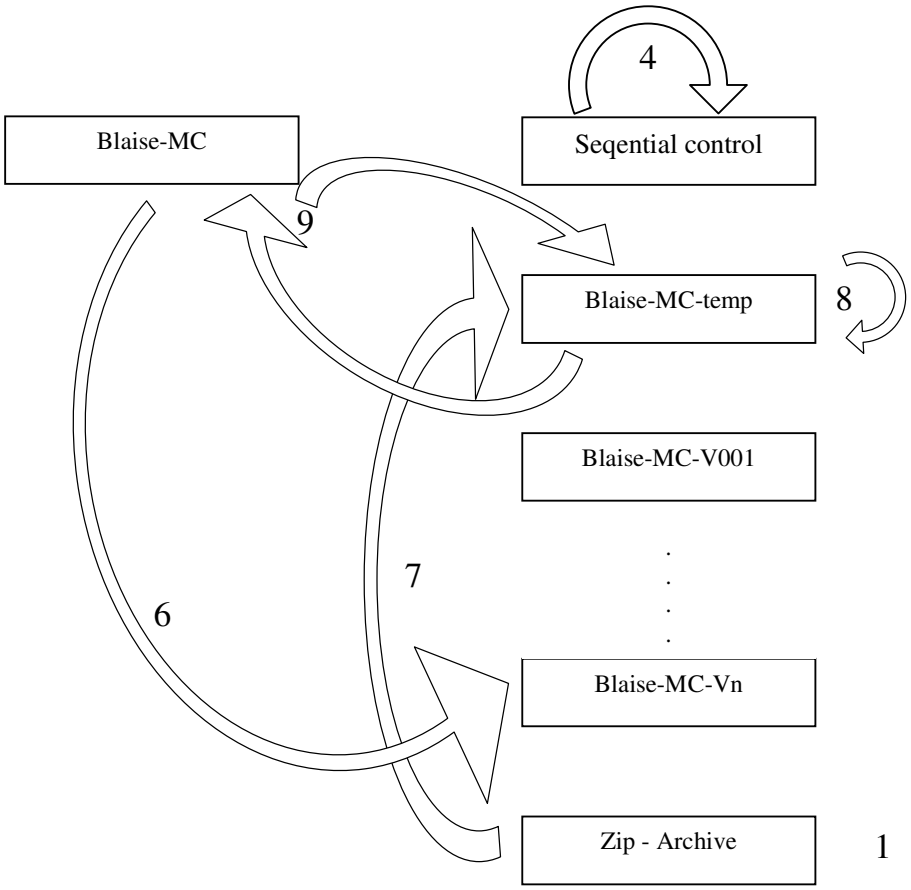
The update process is implemented in the following steps:

1. Unzip and examine the download file
2. Interpret the command file/the command script
3. Implement sent programs and procedures
4. Update the Sequential control directory, and hence the update procedure
5. Examine whether the last update was implemented
6. If the last update process was error free, the Blaise-MC directory is copied to Blaise-MC-temp and to the backup directory of the corresponding version. If the last update process was faulty, the backup directory is copied to the Blaise-MC-temp directory and to laise-MC.
7. The new objects are copied from the Zip-Archive directory to Blaise-MC-temp. If data conversion is necessary, this is implemented in the Blaise-MC-temp directory:

```
-----------|----Data models
           |----Data models – old
           |
           |----Data
           |----Data old
```

Copy of the data and data models in the Data old and Data models old directories. Installation of the new data model in the Data models directory, conversion of the data from Data old to Data

8. Copy of the Blaise-MC-temp directory to Blaise-MC
9. done

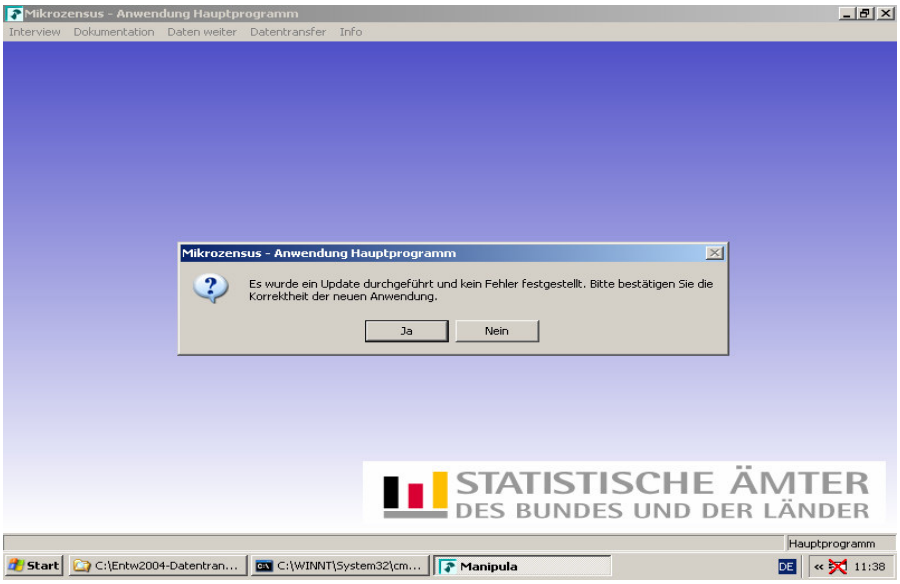


After implementing the update, the interviewer launches the application. The launch program reads the protocols and decides in which directory the interviewer continues to work:

If every thing is OK, it works in the new version.
If an error occurs at 1-7, it works in the Blaise-MC directory in the old version.
If an error occurs at 8, it works in the backup directory, or the backup directory is copied into the Blaise-MC directory. If an error occurs, the protocols can be sent to the Land Office.

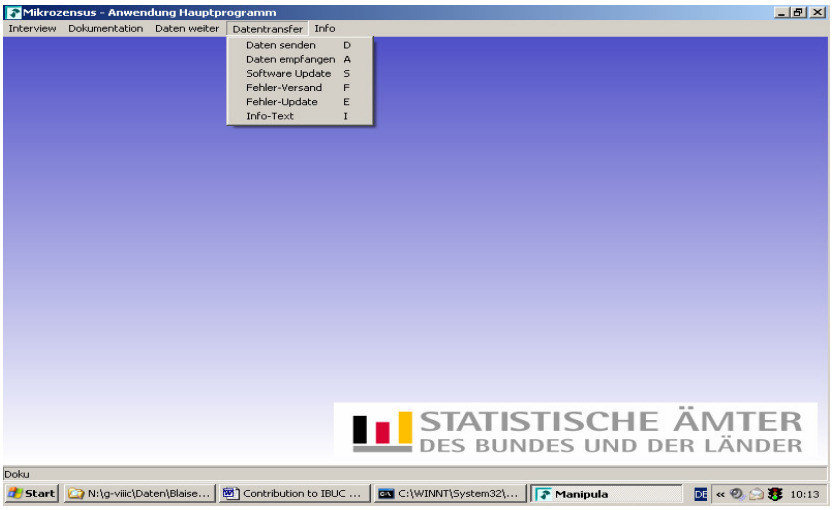
The update of the server application takes place in the same manner.

The checking feature of the protocol files can only deal with technical errors. If a quite fundamental logical error is present in the data model which prevents further interviews being carried out, the interviewer can change to the backup directory when exiting the application via the following question:



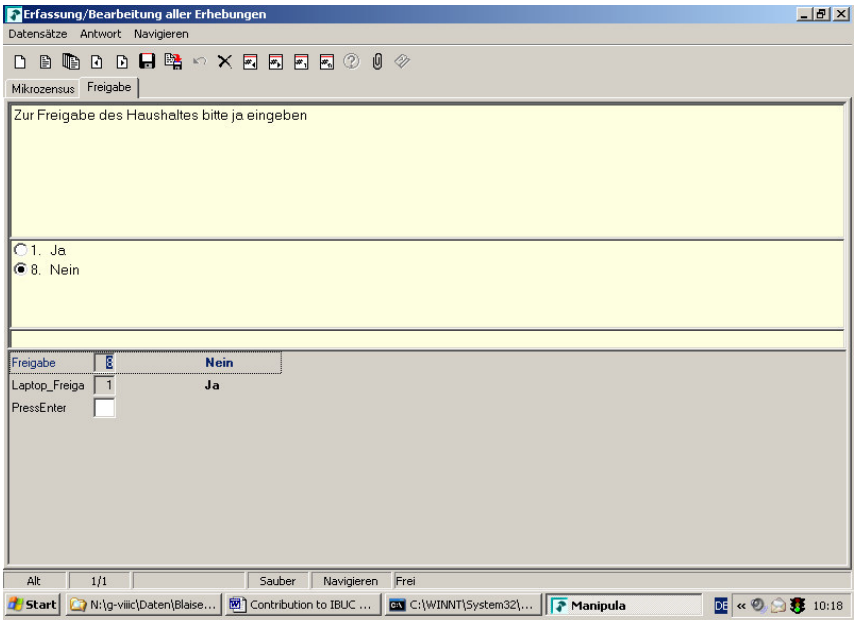
The interviewer receives the information that a file download has taken place when he/she launches his/her daily data transfer. He/she can launch the download and implement the update. We opted to separate the three functions since the download/update procedure may take some time. It is however also possible to implement everything in one function.

The Manipula menu looks like this:



4. Data upload

It is planned for the interviewer to transfer their data to the Land Office daily so that backup copies are available in the Land Office. Backups on another medium, such as a diskette or CD-R, are no longer to be necessary (or only in exceptional cases). It is possible for the interviewer to have to visit a household several times until the interview is complete. It can also happen that the interviewer passes the case to the Land Office and the case is further processed there. In order to avoid "spread" data collection here, the interviewer concludes the interview by entering a release character:



Once the release has been confirmed, the interviewer can only read the information, but can no longer change anything. This is possible in Blaise, but rather laborious.
All fields have to be placed on the route once with ".ask" and once with ".show". We would prefer to be able to set within a data model via a condition whether the sentence is opened "read-only" and not only via a parameter of the command line. Is this possible? Release must naturally be confirmed.

The data are forwarded as follows:

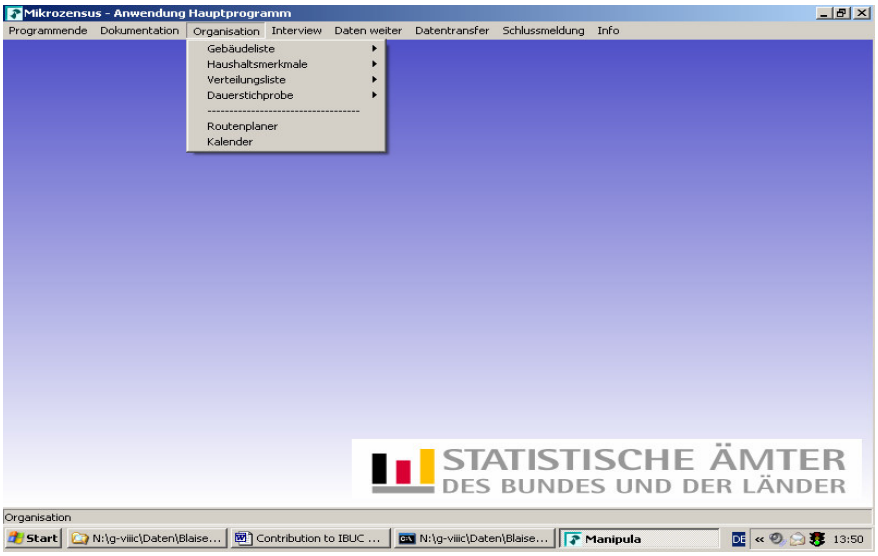
All data for the download are compressed in a Zip archive; the file names are given a date stamp, the version and an interviewer ID. A batch procedure is created which opens a data telecommunication connection. This takes place as standard with Rasphone; a specially-programmed browser is opened which only permits data to be sent; the data are sent via this browser, a receipt file is returned, the connection is closed. The Blaise application evaluates the receipt file and issues the message "Data transfer was successful" or "Unable to transfer data".
Every thing is controlled via the application's Manipula menu; the interviewer does not have to be able to operate any other programs.

Various addresses can be used in our browser via an INI file to upload the data and download the file. This makes it possible for an update file to be available on a central server for all interviewers in Germany, whilst the interviewers carry out the data upload with their Land Statistical Office. The interviewer does not notice at all that he/she are working with two addresses.

5. Management and deployment of the interviewers

The jobs are sent to the interviewers 24 times per year via a data download. This list contains the addresses of the buildings which the interviewer is to survey for the first time, and information on the households from the previous years which are to be surveyed for the second, third or fourth time. These data are fed into organisation files and processed. Since for reasons of data protection no addresses may be stored in the interview, address management must take place in the organisation files. The interviewer enters information important for billing, such as the mileage covered, in a separate file and sends it to the Land Office with the data.
It has to be possible on the laptop to delete a falsely sent work package. An error may well occur here if one has 3,000 interviewers. The old must be replaced by a new one.

The organisation data are processed in the same Maniplus menu as the interview:



It was considered to provide the interviewers with route planners, electronic maps and geocoded addresses to help them find the buildings and households, but this is not going to take place until later.

6. Management project, controlling project

Blaise is used to enter the data and for the organisation files, Manipula programs being deployed in order to control the application. Not all other functions (steps) which are downstream to data entry and are important to implement the survey are not carried out in Blaise. For this, in the Land Statistical Offices, other database-orientated solutions, in some cases also mainframe applications, are used. The data are exported from Blaise and then imported into the special applications.

This includes the following steps:

sample selection
formation of the work packages (Map&Market)
reminder procedure
interviewer management
building selection
evaluation and preparation of tables
etc.

7. Bascula

However we in turn carry out compensation for the missing answers and the extrapolation for the evaluation/preparation of tables with the Blaise tool Bascula in the Land Statistical Offices. The data are exported from the Blaise application, imported to Bascula together with the benchmark tables, where the material is added to by the extrapolation factors and exported. The Bascula programs are created by the specialist department in a dialogue; these are then launched as Manipula programs (function export) with weight – section in the batch.

Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey

Mark Pierzchala, Debra Wright, Claire Wilson, and Paul Guerino / Mark Pierzchala, Mathematica Policy Research, Inc.

In this paper we discuss the design implications involved in collecting and processing data via three disparate survey modes: mail, Web, and computer-assisted telephone interviewing (CATI) for the National Survey of Recent College Graduates (NSRCG). We provide a brief background of the study and outline the design and development process. Also discussed are the similarities and differences between the modes, examples of questions that posed particular challenges, and survey management implications. We conclude with lessons learned and advice for future multi-mode surveys of this nature.

1. Background and History of the NSRCG

The NSRCG is sponsored by the National Science Foundation (NSF) Division of Science Resources Statistics. It is one of a family of three NSF sponsored surveys designed to collect information to estimate characteristics of the population of scientists and engineers in the United States. Together, the three surveys constitute the Scientists and Engineers Statistical Data System (SESTAT). The first NSRCG (then called the New Entrants Survey) was conducted in 1974. Since then, subsequent surveys have been conducted about every two years. The 2003 NSRCG is the 16th survey in this series and was conducted in recent fieldings as a CATI-only survey. Mathematica proposed to cut costs and increase response rates for the 2003-2004 NSRCG by utilizing three modes of data collection. The survey was designed as a mail survey with a Web option and CATI follow-up for non-respondents.

1.1 Purpose of the NSRCG

The main purpose of the NSRCG is to provide estimates of the number of recent science, social science, and engineering graduates, and their recent employment experiences, salaries, and demographic characteristics. The first section of the questionnaire includes questions about the graduate's education, including their primary and secondary majors, degree types, degree dates, and the institutions from which their degrees were received. The middle sections ask about the graduate's employment situation as of the reference date (October 1, 2003), including detailed questions about their employer and type of work, as well as questions about their job satisfaction, and how closely related their job is to their field of study. The final section of the questionnaire includes key demographic questions.

1.2 Survey Population and Study Modes

The 2003 NSRCG utilized a stratified, two-stage sample design. In the first stage, a nationally representative sample of 300 colleges and universities was selected. During the second stage, individuals who graduated from the selected institutions with a master's or bachelor's degree in science, social science, engineering, or a health related field between July 1, 2000 and June 30, 2002 were randomly selected within cohort (year of graduation), type of degree received (bachelor's or master's), and major field of study (science, social science, engineering, and health related field). In order to provide reliable estimates of female and minority scientists, these groups were sampled at higher rates.

Graduates were sent an advance letter and an address verification form in September of 2003 explaining that they had been selected for the survey and asking them to provide updated contact information. A paper questionnaire and cover letter were sent in early October. Reminder postcards were sent two weeks later to non-respondents. After four weeks, non-respondents were sent another paper questionnaire and a cover letter offering an option to complete the survey on the Web. Each sample person was assigned a unique username and

password that was included in the mailing with instructions for completing the survey online. A post-it note with the pre-printed message "Try something new. Respond by Web!" was attached to the paper questionnaire to encourage Web participation. Sample persons who had not yet responded by Web or paper were contacted by telephone beginning in early February. Three e-mail reminders containing a link to the Web survey were sent over the course of the field period – four weeks after the onset of CATI data collection, twelve weeks later, and then two weeks before the close of the data collection period. Clicking on the link took the graduate to the Web survey and automatically filled in the login id and password.

1.3 Comparison of Single versus Multimode Administrations

An experimental group of 2,000 graduates were not offered the mail or Web survey option. Since previous rounds of the survey had used CATI as the primary mode of data collection (only sending a paper questionnaire if the respondent requested it), this group acted as a control group to assess the effectiveness of the multi-mode design.

1.4 One Blaise Instrument, Database, and Survey Management Scheme

One Blaise instrument was designed to collect CATI and Web data and was used to enter data from paper questionnaires. This approach to data collection offered several advantages including: it reduced programming effort by eliminating the need to program three separate instruments; all data were written to a single database which eliminated the need to merge databases after data collection (e.g., partial responses from Web could also be incorporated into and completed in CATI); and survey management was simplified.

2. Review of Attributes of Modes

Paper, Web, and CATI surveys differ in several fundamental ways. Differences include the way that questions are worded and displayed, data definition, and in some instances question order. Much of these design differences are the result of issues related to presentation, manner of responding, segmentation of the questionnaire, the dynamic versus passive nature of the questionnaire, administration (self versus interviewer), pace of interview, the medium of the interview, and the training of the person recording the responses. We discuss each of these issues and how they affected our design considerations briefly below.

2.1 Visual versus Aural Reception

One of the most obvious differences between CATI, paper, and Web surveys is their channel of presentation. CATI surveys are presented aurally, while paper and Web surveys are presented visually. The CATI respondent is totally dependent on the linear presentation of the survey by the interviewer. In Web and CATI modes, the respondent can rely on the layout of the questionnaire or browser screen to understand the entire content of a question with sub-items at a glance. The different channels of presentation also decrease the level of standardization across modes, as the CATI scripts need to be modified to reduce the cognitive complexity of responding and fit conversational norms. Section 4.1 below gives an example of how one question was worded differently based on its reception.

2.2 Oral versus Written versus Typed Transmission

The manner in which participants respond differs by survey mode, as well: CATI respondents speak, mail respondents write or check, and Web respondents type or click. Because each channel has the potential to affect respondent behavior, care must be taken when designing each mode. For example, the self-administered questionnaire (SAQ) is generally considered the best method to collect information on socially undesirable behaviors such as illicit drug use (Aquilino 1994) and the number of lifetime sexual partners (Tourangeau and Smith 1997) because it offers the greatest level of privacy. To be sure, these effects are moderated by respondent level of trust, whether the SAQ is self-administered or interviewer-administered, and

the presence of family members during the interview (Aquilino, Wright, and Supple 2000; Wright, Aquilino, and Supple 1998; Aquilino 1994). Thus in designing a multimode survey such as the NSRCG, researchers are faced with how best to collect comparable information across channels that are public (i.e., spoken) and thus more prone to social desirability bias and private channels. This may involve offering more assurances of confidentiality in CATI or integrating such technologies as telephone data entry to record the frequency of sensitive behaviors in a live telephone interview.

2.3 Segmentation

Whereas segmentation has been traditionally viewed as a known side effect of CATI and CAPI surveys (Groves, Berry, and Mathiowetz 1980; Pierzchala and Manners 2001), Web surveys can also limit one's ability to examine the survey instrument as a whole. Since there are no intervening interviewers or supervisors in the Web mode, the concept of segmentation must be expanded to include *the respondent* when discussing multimode surveys. CATI surveys lie on one end of the overview-segmentation continuum in that survey participants do not have the ability to determine how responses affect future questioning, and their memories must serve as the method by which they access former responses. Conversely, paper respondents are able to scan the entire questionnaire before responding and therefore have complete access to previous responses. Where Web surveys fall on this continuum depends on programmatic decisions. For example, Web surveys that that present just one question per browser screen would be considered very similar to a CATI survey on the continuum. On the other hand, Web surveys that present several or many questions per browser screen and allow users to navigate through previous responses and future questions freely would be considered more similar to paper questionnaires.

2.4 Dynamic versus Passive

CATI and Web modes are dynamic modes in that routing, question wording, edits, and computations can all be adapted, executed, and enforced during the survey administration. This dynamism introduces several advantages, such as tailoring questions to the respondent, skipping unnecessary questions, following up on suspicious situations (e.g., no bachelor's degree recorded), and proceeding through difficult sections in very methodical ways to ensure no important data are missed. The passive paper mode relies on visual cues such as arrows, column headings, and the like to elicit correct responses. Dynamic displays may lead to further segmentation. For example in question B3 below (section 4.2.1), the year of retirement appears on an ensuing browser screen only if the retirement box is checked on the previous screen. In question A18 (section 4.2.4) the fact that graduate expenses may or may not always be asked was one reason the undergraduate and graduate expenses were handled on different browser screens.

2.5 Self Administration versus Interviewer Administration

Web and paper modes are self-administered modes while CATI is an interviewer-administered mode. Question wording may differ based on this attribute (see 4.2.1 below) as great care is made to make question text clear in any mode. Further, the interviewer is an intermediary and can be the source of error. On the other hand, the interviewer can explain, persuade, and otherwise facilitate the successful completion of the item or interview. Section 4.2.4 below gives an example where the interviewer-administered mode may aid in the collection of sensitive data.

2.6 Pace of the Interview

In addition to the differences described above, CATI, paper, and Web modes also vary in terms of the pace of the interview and in the determination of the pace. The pace of a self-administered questionnaire is wholly driven by the respondent's ability to comprehend the question, access the appropriate information, and respond. While the survey researcher can

design the questions and instrument to reduce comprehension and retrieval times (see, for example, Mangione 1995), they have no control over the process once the survey is in the respondent's control. The pace of the CATI mode is determined first by manner of question construction (e.g., don't repeat question stems every time) and secondly by implicit negotiation between the interviewer and the respondent. On the other hand, the pace of the Web mode can vary greatly between respondents, due solely to the hardware and software infrastructure between them and the Web survey server. Poor connect speeds can introduce several-second delays between browser screens and contribute to premature exits. One way to try to compensate for this possibility is to add more than one question at a time to one browser screen.

2.7 Medium

The medium of the CATI interview is the phone, the medium of the mail questionnaire is paper, and the medium of the Web survey is the browser. As opposed to other media, many aspects of the browser-as-medium are out of the control of the survey organization, including whether or not cookies are allowed, presence of a pop-up blocker, screen resolution, color rendering, processor speed, and the quirks of each individual Web browser. The need to accommodate this variable environment encourages a "lowest common denominator" approach to Web browser display, which in turn makes it more difficult to match the browser display to paper. Section 4.2.4 gives an example of a question where browser could not match the display of a paper questionnaire.

2.8 Trained versus Non-Trained Recorders

At MPR, CATI interviewers are highly trained and capable of executing a variety of complex actions such as handling edits, navigation, and correcting answers. On the other hand, the paper and Web respondents are untrained so the media must be clear and virtually foolproof. Web and paper respondents cannot be given the full range of operability that the CATI interviewer has. In Web mode, for example, the respondent would not be expected to be able to handle a consistency edit or to engage in navigation beyond one browser page at a time. (In some other surveys such as required, repeated, and frequent surveys, it would be possible to train a paper or Web responder in higher order actions.)

3. Multimode Instrument Design and Development

The development of the data collection instruments and CASIC modes was done in succession starting with the paper questionnaire, followed by a conversion to CATI, then adaptation to the WEB mode, and finally further modifications to accommodate data entry and follow up and completion of some cases.

3.1 Development of the Paper Questionnaire

The paper questionnaires for all three SESTAT surveys are reevaluated before every administration in a separate SESTAT coordinating contract from NSF currently held by MPR. The main objective of the coordinating contract is to develop standard instruments and methods between the surveys. In addition to the paper questionnaires, MPR developed web guidelines, proxy guidelines, and coding manuals among other products. From the standpoint of MPR's instrument development effort for the 2003 NSRCG, the already-existing paper questionnaire was taken as an established starting point.

3.2 Development of the CATI Mode

The Blaise instrument was first programmed with the CATI survey in mind. The paper questionnaire provided the basis for the CATI instrument routing and response options. However, the question text from the mail survey needed to be adapted to an interviewer-administered format. In addition, the computer-assisted technology of CATI meant the telephone survey could take advantage of dynamic text fills, enforced routing, calculations, and

edit checks. Since the CATI survey was the first step in translating the paper questionnaire into a computerized mode and involved moving from a self-administered to an interviewer-assisted mode, the transition from paper to CATI proved to be the most complex design challenge.

NSF provided guidelines based on previous iterations of the survey, that specified question text and any routing or response options deviations that were specific to the CATI survey. In addition to specific minor wording changes to take into account the interviewer administered nature of the CATI survey, in some instances a question was added that did not appear in the paper version. For example, if the response to "Were you working for either pay or profit during the week of October 1, 2003?" was "No", the follow-on question "Were you not working for any of the following reasons: you were a student on paid work study, you were self-employed and did not get paid that week, you were on vacation from work or traveling while employed, you were on paid sick leave, personal leave or other temporary leave, or you were on a job that did not pay but had other benefits." was asked to make sure the respondent counted these experiences as employment. In the paper version this was simply reflected as an instruction.

In addition, some items were split into sub-parts to accommodate the aural nature of the CATI survey. For example item B14 was split into four questions in CATI (including a filter) to identify whether the type of employer was private, a government agency, self-employed, or other before obtaining more detailed information (Figure 1). In any CATI text displayed below, the interviewer is required to read bolded text; text that is not bolded is optional.

Figure 1 CATI Succession of Yes / No Sub-Items

B14_1	**Was your principal employer during the week of October 1, 2003 a private company or individual, a government agency at any level, were you self-employed, or did you work for some other type of employer?**	**Yes / No**
(If self-employed) B14_2	**Were you self-employed in your own not incorporated business, professional practice, or farm?**	Yes / No
(If private company) B14_3	**Was that a private for-profit organization or individual paying your wages, salary or commissions, or a private not-for-profit, tax-exempt, or charitable organization?**	Yes / No
(If government) B14_4	**Was that local government, such as city or county government, state government, U.S. military service, active duty or commissioned corps, such as USPHS or NOAA, or U.S. government as a civilian employee?**	Yes / No

The paper questionnaire asked B14 in one question as shown in Figure 2.

Figure 2

B14. Which one of the following best describes your principal employer during the week of October 1?

Mark (X) one answer.

1 ☐ SELF-EMPLOYED in own NOT INCORPORATED business, professional practice or farm

2 ☐ SELF-EMPLOYED in own INCORPORATED business, professional practice or farm

3 ☐ A PRIVATE FOR-PROFIT organization or individual working for wages, salary or commissions

4 ☐ A PRIVATE NOT-FOR-PROFIT, tax-exempt, or charitable organization

5 ☐ Local GOVERNMENT (e.g., city, county)

6 ☐ State GOVERNMENT

7 ☐ U.S. military service, active duty or Commissioned Corps (e.g., USPHS, NOAA)

8 ☐ U.S. GOVERNMENT (e.g., civilian employee)

9 ☐ Other – *Specify*

The NSF guidelines in conjunction with the mail version were used to program a CATI instrument that would function well as a stand-alone telephone survey, but that would maintain comparability to the paper question and permit data entry of returned mail surveys.

Fills were also used in the CATI version to simplify questions for respondents. For example rather than asking "Between the most recent degree you earned prior to October 1, 2003 and October 1, 2003, did you enroll or take any courses at a college or university?" the CATI version calculated the most recent degree the respondent reported earning prior to October 1, 20023 and filled it so that the item read "Between the {bachelor's / master's} degree and October 1, 2003, did you enroll in or take any courses at a college or university?"

Unlike the paper version of the survey, the CATI program prevented routing errors and out-of-range responses. The computer-assisted nature of the CATI survey also meant that errors could be minimized during data collection by including range, consistency, and logic checks in the survey. To take advantage of this capability, edit checks were programmed into the CATI instrument, which required interviewers to verify responses for particular items. These checks were based on the Westat CATI instrument and methodology report from the previous iteration

of the survey. In addition, checks were built in for items that were considered "critical" by NSF. These checks required interviewers to repeat certain questions, stressing the importance of obtaining a response to the item, if the response "don't know" or "refused" was given. Without responses to these items, the interview could not be considered complete and the respondent would need to be called back later in an attempt to collect the missing data.

Instrument testing was particularly intensive at this stage since the CATI was the first computerized version of the survey. Testing was focused on making sure that the CATI instrument skips matched those in the paper version and that all additional CATI items, fills, conditional text, edit checks, and calculations were operating correctly.

3.3 Development of the Web Mode

Web development was generally based on a set of guidelines that were issued by NSF and intended to ensure comparability across contractors conducting the three SESTAT surveys. These guidelines addressed such issues as background color, font, placement of navigational buttons, the use of links, and inclusion of special pages that were not part of the actual instrument. Examples of such pages included the login screen, the survey information screen, a quit screen for users who chose to suspend before finishing the survey and a final, or "good-bye" screen. The guidelines ensured that all contractors had a common approach to handling such issues as whether to display question numbers, whether to use progress indicators, and how many questions to display on a screen.

CATI specifications provided the initial basis for programming the Web survey. The instrument was subsequently modified in response to usability testing findings and client review. The usability testing protocol followed either a *talk aloud* or a *retrospective* format. In the talk aloud format, an observer and the participant were in the same room while the participant completed the survey online. Participants were asked to express any thoughts or reactions they had to each screen as they progressed through the survey, and occasionally, the observer would ask more directed questions about particular features or screens in the instrument. This format had the advantage of enabling observers to capture each participant's impressions of the survey as they occurred.

During the retrospective testing sessions, the participant completed the Web survey without interruption while observers watched the participants' screen from a remote computer in a separate room. Because participants were not being watched or interrupted while completing the survey, this approach had the advantage of being more natural, while demonstrating how long it would typically take respondents to complete the survey. Once the participant completed the survey, the observers would review each screen with the participant to uncover potential problems or issues.

Usability testing findings led to changes in layout, question numbering, and question wording. Regarding layout, we tried to assess the best way to display questions that were difficult to fit on a single screen due to lengthy instructions or numerous response choices. For example, based on participants' feedback, we determined that splitting a question with 14 sub-items across two browser screens was preferable to displaying the entire question on one screen with little white space or with a scrolling browser page. Usability testing was also instructive in determining the best way to format instructions for individual questions so that respondents would be most likely to read them.

We also investigated issues related to question numbering during usability testing. Although some Web surveys do not display any question numbers in an effort to simplify the page, we determined that question numbers were necessary for two reasons. First, if respondents contacted the help desk about a problem they were having with a particular question, the question number would make it easier for them to communicate which question was posing a problem. Second, when errors or problems were identified during testing, displaying the question numbers made it easier to document which questions were affected.

Although displaying the question numbers on the Web made good sense for the aforementioned reasons, this decision also raised other challenges relating to sequence. Because some questions were asked in a different sequence in the Web survey than on the paper questionnaire, we were faced with the decision of whether to maintain the original question number, resulting in questions sometimes appearing out of sequence on the Web, or whether to re-number the Web questions so they appeared in sequential order. If we chose the latter option, the Web question numbers would not coincide with the paper question numbers. Through testing, we learned that participants rarely paid attention to the question numbers, so we opted to maintain the paper question numbers within the Web mode. A note on the instructions page explained that some questions might appear out of sequence, and some questions would be skipped entirely if they were not applicable, as determined by previous answers. This explanation and approach to question numbering seemed to work well in facilitating identification of questions without introducing unnecessary confusion.

Usability testing also helped us to refine CATI-based question wording that proved awkward in the Web environment. For example, the first verification question in the CATI screener was "Are you {name}?" Usability testers reported that the question seemed too conversational for a Web survey so the wording was changed to read, "Is your name {name}?" in the Web mode.

3.4 Development of Data Entry

The instrument was adapted to accommodate data entry resulting in changes in routing and application of edits, and implementation of verification.

There are subtle differences in instrument routing between data entry and the other modes that is necessary to facilitate data entry. The data entry capability allows heads-down data entry; that is the operator is not required to look up at the computer screen in order to capture data. In some spots in the instrument, CATI and Web modes can determine the answer to a subsequent question and auto-fill and skip over that item. If, however, the item is present on paper, the operator has to be able to enter it, even if auto-fill and skip are programmatically possible. Otherwise, heads-down operators could soon find themselves out of sync with the instrument as they enter what they see on paper.

Data entry operators were required to verify that a blank response had been left missing by the respondent. In addition, items governing branching required a valid response. If a routing item was blank and its value could not be determined from other values, the entry for the case was suspended and held for follow up or data retrieval.

To facilitate data entry, a so-called 'fan edit' was conducted for each paper form before data entry. Mail survey respondents could commit a variety of errors (such as leaving on-route items blank and recording off-route information; recording ranges instead of a single numeric value; and checking two or more values for a "mark one" question). Rules were pre-defined for each possible respondent error that the data entry instrument would not be able to handle and cleared up ahead of time based on NSF guidelines for data editing. The 'fan edit' concentrated only on problems that could affect data entry, it was not a complete manual hand edit; that is, many edits were reserved for machine editing downstream after data entry.

The staff that conducted the fan edit performed the data entry. They were capable of stopping data entry on a case to clear up errors that may have been missed in the fan edit. This method of reducing the time spent on the fan edit and allowing some errors to be cleared up by the data entry operator was implemented with the aim of reducing time spent on the overall process.

Verification was required for 5% of cases. To accommodate this, the instrument randomly selected cases based on the system time when the case was brought into memory. These cases were flagged for verification and were then accessible for double entry. Selected cases were completely verified for all items. This was accomplished by conducting double data entry for

each item with immediate detection and resolution whenever a difference in values was detected. The second operator was the adjudicator. For any difference, the first entered and the second-entered values were displayed and the adjudicator could choose one or the other, or enter yet a third value if necessary.

4. Collection Instrument Similarities and Differences Between Modes

In this part of the paper we discuss specific examples of issues we encountered in designing one instrument to collect data in three modes. We were faced with several challenges due to differences in data definition question order, routing, and the mechanics of coding state, education, and occupation; as well as browser display issues, auto-fill and skip-over items, the number and order of collection of university degrees, and the re-asking of critical item questions across modes.

General conventions we adopted to handle these issues include the formulation of question text and question format, handling of non-response, implementation of edits, acquisition of Critical-Complete items, and implementation of remarks. Some questions were treated the same for all modes, some were treated the same for two modes and differently for the third, and some were handled differently between all three modes. Additionally, some questions were present in only one or two modes.

Appendix A illustrates question-specific similarities and differences between modes for the NSRCG instrument. A visual scanning of Appendix A illustrates that accommodating differences between modes for each question was done for about half the questionnaire. Some of these differences were driven by globally applied conventions. Other differences stem from inspection and assessment of each question.

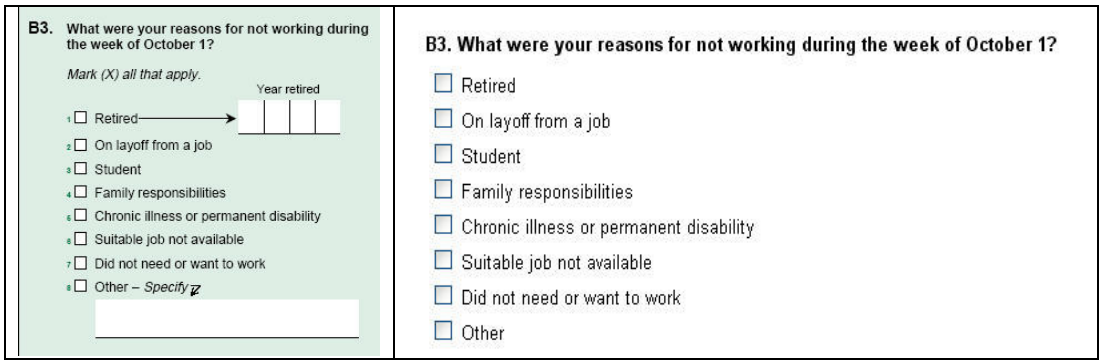
While this paper focuses more on differences between questions between modes, a nice example of a question (A14) where all three modes are handled similarly, is given in the companion MPR paper *System Implementation for a Blaise Multimode Web, Cati, and Paper Survey* by Hart, Foster-Sardenberg, and Okada. A14 is rendered as a series of Yes/No sub-items in all three modes.

4.1 Question Text

Different features and capabilities of the three modes typically dictated whether questions had to be worded differently or whether they could use the same format. Two major distinctions that affected question text formulation were whether the mode was self-administered (Paper/Web) or interviewer-administered (CATI) and whether text could be dynamically modified (CATI/Web) or was passive (Paper).

For the self-administered modes, several questions used a “Mark all that apply” response format whereby respondents would select their responses from a displayed list. Paper and Web respondents would mark check boxes to select all applicable responses as shown in Figure 3 for question B3.

Figure 3 Example of Similar Displays on Paper and for Web Modes



To translate such items into an interviewer-administered environment, the question stem was re-worded and the interviewer presented each response as a separate question in a series, to which the respondent replied yes or no. For example, question B3 used the following wording in CATI (Figure 4, all required-to-read text is in bold):

Figure 4 CATI Succession of Yes / No Sub-Items

B3_1	**For which of the following reasons were you not working during the week of October 1, 2003? Please answer yes or no for each of the following. Were you not working because you were retired?**	**Yes / No**
B3_2	Were you not working because you were **on layoff from a job?**	Yes / No
B3_3	Were you not working because you were **a student?**	Yes / No

NOTE: Not all sub-items are not shown.

Whereas paper and Web used identical question wording for some items by virtue of their common attribute of self-administration, Web and CATI followed the same wording for other items that rely on dynamically filled text. For example, Question B12 (Figure 5) asks about the number of people working for the principal employer. In Web and CATI this question had custom fills depending on the answer to B14, (which was asked before B12 in these two modes but after B12 in the paper mode). B14 provides a classification of employer type (see Figure 2 in section 3.2 above).

Figure 5 Dynamic versus Static Text in Question B12

CATI and Web modes	**Counting all locations where {you operate / this employer operates}, how many people work {for your business / for your principal employer}? Your best estimate is fine.**
Paper mode	Counting all locations where this employer operates, how many people work for your principal employer? Your best estimate is fine.

4.2 Question Format

Differences among the three modes affected the format and layout of questions as well as question wording. In this section we give specific examples of questions that required different formats between modes.

4.2.1 Education and Occupation Coding

Because the main purpose of the NSRCG is to collect information regarding the educational and employment experiences of recent graduates, correctly categorizing and coding primary majors and occupation titles is of key importance to NSF. Coding of education and occupation was executed dynamically in CATI and Web in the same way while coding on the paper questionnaire was done altogether differently because of its passive nature and need to save space. NSF provided pre-defined coding dictionaries for mapping educational majors and occupation titles to numeric codes. These dictionaries were linked to the CATI and Web instruments so that verbatim responses could be auto-assigned a code during the interview. For example, the response "mechanical engineering" mapped to code 735. If the respondents' major field of study exactly matched one of the major fields listed in the educational dictionary, then the field of study was automatically assigned the appropriate code designated in the educational dictionary. If the verbatim major or occupation title did not map to a major or occupation in the coding file, respondents were asked a full or partial set of branching items to classify the response (see Appendix C).

Because of space limitations and the static nature of paper, a hierarchical branching scheme for determining occupation and education was not possible. On paper, respondents were asked to write in their verbatim major and occupation and then refer to the back pages of the questionnaire to look up the three-digit code that most closely matched their response. They were then asked to write this code in the space provided (see Appendix D). This method had the advantage of allowing respondents to view all of the majors and occupations and select the code that was the best match, but some respondents left the code blank or entered more than one code. In addition, some respondents entered codes that did not match the verbatim responses.

4.2.2 Degree Grid Differences

Collecting education background is also of key importance. On paper, this information was collected in a degree grid that allowed respondents to fill in information concerning their first bachelor's degree, and their most recent and second most recent degrees (see Appendix B). The dynamic nature of CATI and Web allowed the Section A degree grid to be handled much more systematically than in the passive paper medium that also had space constraints. Section A contains a degree grid roster where all respondent degrees of Bachelor's level or above are entered. The paper questionnaire allows a maximum of 3 degrees as this is all that can fit on one page.

While the vast majority of respondents have less than 3 degrees, previous rounds of the survey have shown that the respondent could have earned more than 3 degrees. For this reason, CATI and Web modes allow up to 9 degrees. After information for each degree is recorded CATI and Web modes probe for additional degrees, listing already-recorded ones. These two modes also probe for a bachelor's degree if one is not recorded.

On paper, none of the column headings refer to the sample degree (which was the degree which determined the respondent's eligibility for the study). Indeed, as the paper questionnaire has no sample-degree information attached or included, the respondent is free to enter degrees in any order, and there is no guarantee that the sample degree will even be recorded. It was often difficult to programmatically determine if the sample degree was reported on the paper form, or if reported, where it was included, because the sampling frame information and respondent-supplied information for the same degree did not always agree. In CATI and Web, the screener confirms sample degree information and this degree becomes the first degree recorded in the grid.

Thirteen items are collected for each degree, divided into 5 subsections. On paper, the subsections are collected in the order of: a) School name and location; b) month and year; c) type of degree; d) verbatim text of primary field of study and second major; and e) code for primary and second major (verbatim-verbatim | code–code). For CATI and Web, the sub-sections are collected in the order of: c) type of degree; b) month and year; a) school name and location; and then d) and e) intermixed (verbatim-code | verbatim-code). The reason for the difference in the order of collection for degree information is to accommodate the most natural flow and wording of each mode.

4.2.3 Salary

Another example of differences in question layout is salary. In collection of salary, interviewer-mediation in CATI allows probing items to follow Don't Know or Refusal for a reluctant respondent. All three modes followed the same format for the salary question, offering respondents an open-ended field for entering an exact dollar amount. The CATI instrument used the same wording as Web and paper, but the CATI mode allowed interviewers to probe for a salary range in an effort to avoid missing data on this sensitive item. If the respondent answered Don't Know or Refused to the salary question, the interviewer asked whether the salary was above or below $35,000 a year. If the respondent answered this question, the interviewer would try to get the respondent to report the salary within a ten thousand dollar range. Although this approach resulted in mode differences with salary ranges being collected in one mode only, the trade-off of reducing item non-response was thought to outweigh any drawbacks of the mode differences.

4.2.4 Borrowed versus Still-Owed Educational Expenses

All three modes also handled educational expenses differently due to passive versus dynamic media, interviewer- versus self-administered modes, and space limitations for both paper and browser. For example, question A18 asks about types of financial support received for both graduate and undergraduate degrees. The Web (Figure 6) and paper (Figure 7) modes used a "Mark All" for data definition while CATI (Figures 8 and 9) used a series of Yes / No sub-items for the 9 response possibilities. In this case, visual presentation was particularly important for the self-administered modes in order to make clear the distinction between financial support for undergraduate versus graduate degrees. In the Web mode, we found that putting the two sub-questions in consecutive browser screens could be confusing because the screens were virtually indistinguishable, with only one word difference ("undergraduate" or "graduate"). However due to browser limitations and the fact that the Web mode handled the display of graduate expenses dynamically, we placed undergraduate and graduate questions on separate screens.

Figure 6
Web
mode

2003 National Survey of Recent College Graduates

A18. Thinking of any undergraduate degrees you completed before October 1, 2003:
From which of the following sources did you receive financial support?

Check all that apply

- ☐ Financial support from parents, spouse, other relatives, not to be repaid
- ☐ Loans from the school you attended, banks, federal or state government
- ☐ Loans from parents or other relatives
- ☐ Financial assistance from your employer
- ☐ Tuition waivers, fellowships, grants, scholarships
- ☐ Assistantships or work study
- ☐ Earnings from employment
- ☐ Other - specify

Previous | Suspend | Continue

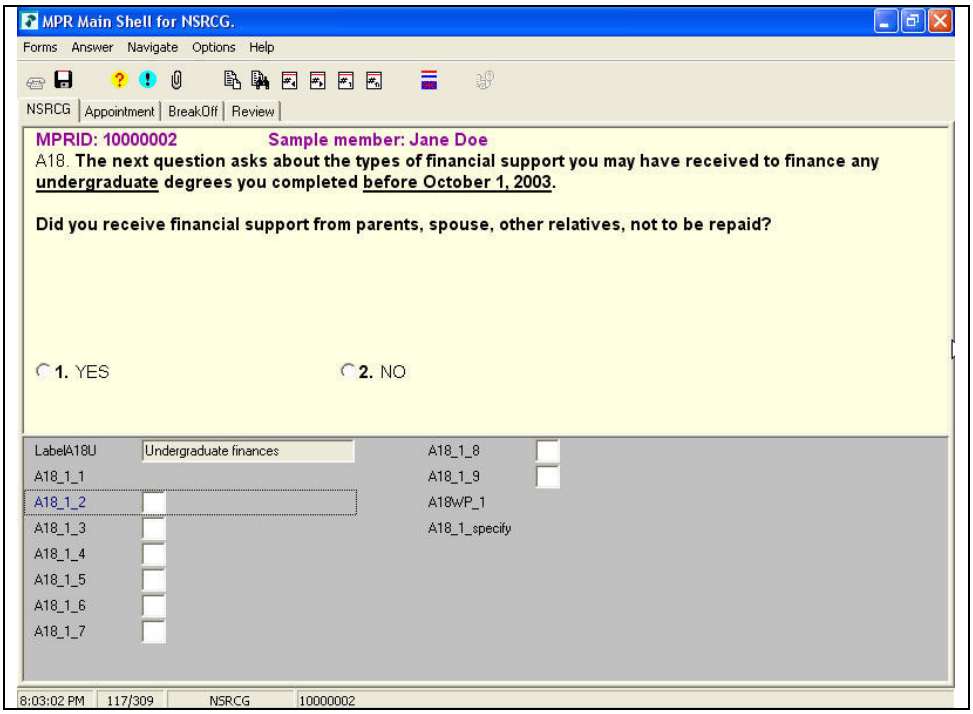
Figure 7
Paper
mode

A18. The next question asks about the types of financial support you may have received to finance any undergraduate or graduate degrees you completed before October 1, 2003.

Mark (X) all that apply for each undergraduate and graduate column.

		Undergraduate	Graduate
1	Did not earn a degree at this level	1 ☐	1 ☐
2	Financial support from parents, spouse, other relatives, not to be repaid	2 ☐	2 ☐
3	Loans from the school you attended, banks, federal or state government	3 ☐	3 ☐
4	Loans from parents or other relatives	4 ☐	4 ☐
5	Financial assistance from your employer	5 ☐	5 ☐
6	Tuition waivers, fellowships, grants, scholarships	6 ☐	6 ☐
7	Assistantships or work study	7 ☐	7 ☐
8	Earnings from employment	8 ☐	8 ☐
9	Other – *Specify*	9 ☐	9 ☐

Figure 8 CATI mode



Note: A similar screen appears for graduate expenses if a graduate degree was recorded.

Figure 9 illustrates the question sequence for the first 3 CATI questions.

Figure 9 CATI Succession of Yes / No Sub-Items for A18

A181_2	**The next question asks about the types of financial support you may have received to finance any <u>undergraduate</u> degrees you completed <u>before October 1, 2003</u>.** **Did you receive financial support from parents, spouse, other relatives, not to be repaid?**	**Yes / No**
A181_3	The next question asks about the types of financial support you may have received to finance any <u>undergraduate</u> degrees you completed <u>before October 1, 2003</u>. Did you receive **loans from the school you attended, banks, federal or state government?**	Yes / No
A181_4	The next question asks about the types of financial support you may have received to finance any <u>undergraduate</u> degrees you completed <u>before October 1, 2003</u>. Did you receive **loans from parents or other relatives?**	Yes / No

Notes: Not all sub-items are shown. A similar set of sub-items followed in CATI for graduate degree financing.

Question A19 provides another example of some of the challenges inherent in translating a question that was developed for paper (Figure 10) into one that works equally well in the CATI and Web modes. Having 4 columns of data points side-by-side on paper, this question posed one of the biggest challenges during Web development. The browser display (Figure 11) was again divided into two consecutive screens (undergraduate and graduate) but this time with two columns each while the CATI version was rendered with 4 numeric (currency) questions.

Figure 10
Paper mode

A19. **The next question asks about the TOTAL amount you have borrowed to finance undergraduate and graduate degrees you completed before October 1, 2003, and how much you still owed as of October 1, 2003.**

Mark (X) one answer in each undergraduate and graduate column.

	UNDERGRADUATE		GRADUATE	
	Total Amount Borrowed	Amount Still Owed as of October 1, 2003	Total Amount Borrowed	Amount Still Owed as of October 1, 2003
1 Did not earn a degree at this level	1 ☐	1 ☐	1 ☐	1 ☐
2 None	2 ☐	2 ☐	2 ☐	2 ☐
3 $1 - $5,000	3 ☐	3 ☐	3 ☐	3 ☐
4 $5,001 - $10,000	4 ☐	4 ☐	4 ☐	4 ☐
5 $10,001 - $15,000	5 ☐	5 ☐	5 ☐	5 ☐
6 $15,001 - $20,000	6 ☐	6 ☐	6 ☐	6 ☐
7 $20,001 - $25,000	7 ☐	7 ☐	7 ☐	7 ☐
8 $25,001 - $30,000	8 ☐	8 ☐	8 ☐	8 ☐
9 $30,001 - $35,000	9 ☐	9 ☐	9 ☐	9 ☐
10 $35,001 or more	10 ☐	10 ☐	10 ☐	10 ☐

Figure 11
Web mode

2003 National Survey of Recent College Graduates

A19. The next question asks about the TOTAL amount you have borrowed to finance undergraduate degrees you completed before October 1, 2003, and how much you still owed as of October 1, 2003.

Total Amount Borrowed	Amount Still Owed as of October 1, 2003
○ Did not earn a degree at this level	○ Did not earn a degree at this level
○ None	○ None
○ $1 - $5,000	○ $1 - $5,000
○ $5,001 - $10,000	○ $5,001 - $10,000
○ $10,001 - $15,000	○ $10,001 - $15,000
○ $15,001 - $20,000	○ $15,001 - $20,000
○ $20,001 - $25,000	○ $20,001 - $25,000
○ $25,001 - $30,000	○ $25,001 - $30,000
○ $30,001 - $35,000, or	○ $30,001 - $35,000, or
○ $35,001 or more	○ $35,001 or more

Previous Suspend Continue

Contact Help Desk Frequently Asked Questions (FAQs) Instructions

A drawback of splitting the question across two screens on the Web survey was that respondents who moved through the question quickly would not notice the distinction between the two screens. An advantage of the Web and CATI modes was that the question about graduate degrees was not asked if the respondent had already indicated that the highest degree earned was an undergraduate degree.

4.3 Allowing and Not Allowing Don't Know, Refusal, and Empty Values

Non-response is defined here as a Refusal or Don't Know response, or as leaving an on-route field EMPTY. The CATI mode required a non-empty response for every on-route question but allowed Refusal or Don't Know. The Web mode did not present or allow Refusal and Don't Know but the respondent could leave answers empty. The passive paper questionnaire allowed the respondent to answer, skip questions, or record a Refusal or Don't Know. The difference between modes with respect to RF, DK, or EMPTY resulted in mode-dependent interpretation of these kinds of non-responses. CATI is the only mode that can reliably differentiate between a

Don't Know and a Refusal. In Web mode an empty answer can mean either Refusal or Don't Know but it cannot mean an inappropriate skip. On paper, an empty may mean Refusal, Don't Know, or an inappropriate skip. A ramification of these response/non-response conventions between modes is that routing specification depends on a combination of both mode and particular data values.

4.4 Data Definition, Range, and Consistency Edits

Data definition edits (allowable values) are enforced during administration for CATI and Web and after the fact for paper-collected data. Range edits (univariate suspicious answers within allowable range) and consistency edits (multivariate comparisons between two or more items) were applied only in CATI mode during data collection. As with most social surveys, the mere fact of keeping the respondent on route and within data definition eliminates the vast majority of problems (Baker, Bradburn, and Johnson, 1995, discuss this in a CAPI context; O'Reilly, Hubbard, Lessler, Biemer, and Turner, 1994, discuss this for self-administered audio- and visual-CASI). Thus data quality in the Web mode is high as long as the respondent answers each presented item. But if the Web respondent left an item blank, some questions would not even appear. For example, if the Web respondent refused to answer the first question in Section B, they skipped all of Section B and went directly to Section C. It is technically possible to introduce consistency edits into the Web mode, but practically speaking this would work only for items that appear on the same browser page. However, even this kind of Web edit was avoided for fear of dissuading the respondent from continuing.

In paper, if the respondent leaves an item blank, data editors may be able to back-code the item based on subsequent answers. If on paper a respondent answered inappropriately (e.g., multiple responses for a code one, or a range for a value) the manual inspection would apply well-defined rules to allow the most appropriate value to be entered.

4.5 Question-Level Remarks

CATI and data entry allowed for remarks at each question, but the Web respondent did not have the option to enter remarks (this was an explicit design decision). For CATI and paper there was the possibility of review of remarks that may help with coding or determination of proper response, but this is not a possibility for the Web.

4.6 Screener

Both the CATI and Web mode included a brief screener to assess whether the correct person had been contacted and was eligible for the survey. Information obtained from the sampled schools was used in text fills to verify that graduates had attended the sample institution or an affiliated campus, that they had received a bachelor's or a master's degree there, and that they had received their degree within one or the two eligible time frames (2001-2002). If a graduate failed to confirm any of this information, the interview ended and the case was coded as ineligible.

It was not practical to include these items on the paper questionnaire. Rather, eligibility determinations needed to be made during post-processing. The degree grid was reviewed to assess whether the degree the graduate had been sampled for had been reported. The institutions, degree types, degree dates, and majors were compared to the information provided by the schools. Cases that had not included a degree that matched on these variables were coded as ineligible.

5. Additional Capabilities

The instrument was adapted to execute post-collection follow-up of critical items and unfinished Web cases, and in the last days to conduct an abbreviated interview if the respondent would not agree to a full interview.

5.1 Critical Item Follow Up

The NSRCG includes several questions that are regarded as critical items, or items that are of particular importance to NSF. There were two types of critical items. *Critical-complete* items were required for the case to be considered complete. That is, all critical items required a valid response; a Refusal or Don't Know did not count. *Critical-callback* items were of secondary importance. A response was not required for the case to be considered a complete, but an attempt to obtain these data was made if the case was called for critical-complete items. The term *critical item* is used below to refer to both critical-complete and critical-callback items.

In both CATI and Web modes critical-callback item non-response was followed up with a repeat question giving the respondent a second chance to enter the value. However, if the respondent still chose not to give a response, the interview continued with the hope of retrieving critical items at a later time. In this case, the instrument was programmed to detect that a critical item was missing and would flag the case for data retrieval. While paper did not have the possibility of prompting for critical-complete items during administration, cases missing critical items were flagged during data entry. For all modes however, there was the possibility of determining the value of some critical-complete items from other collected data such as verbatim responses (e.g., job description). However, it was not always possible to programmatically determine if a case had a critical-item problem, and data inspection was often necessary.

5.1.1 Critical Item Data Retrieval, Melding of Mode With CATI Platform

The NSRCG multi-mode instrument was programmed to facilitate critical-complete item retrieval. This allowed interviewers to re-contact critical-item cases via CATI (no matter what mode the original data collection was in) and follow-up on the missing critical items only. This also ensured that any newly determined critical items (e.g., due to a new path opening up) would be collected at the same time.

All critical item retrieval was interviewer-administered using the CATI capability of the instrument. However, the internal mode of processing was maintained. This meant that while the presentation to the interviewer resembled CATI, the routing and data definition of the originating mode was maintained. If the critical items could be retrieved, then the case was given a final status code corresponded to the originating mode.

Critical items occurred at scattered places in the almost-complete record (these are noted in column 1 of the Appendix A table). At the start of the case, the interviewer could view a summary of all critical items using a Blaise edit-summary dialog. The interviewer secured cooperation through the normal CATI contact screens. Once contact was made and cooperation secured, the interviewer pressed the End key to get to the next critical item in the order of appearance in the questionnaire.

5.2 CATI Follow Up of Partial Web Responses

A similar follow-up was conducted for partial Web responses. Web partial cases could be put into 3 broad groups: 1) the respondent had logged into the Web survey but few or no items had been answered, 2) the respondent had answered some items on the Web but broke off in the middle of the case, and 3) the respondent had almost completed the Web survey. Cases in the first two categories were shifted to pure CATI mode. Any data previously entered by the respondent on the Web were maintained and displayed in the CATI mode. This allowed the interviewer to go through the CATI introductory screens and then press the End key to arrive at the next appropriate unanswered question. From there, the interview could proceed normally as a CATI case.

The almost-complete Web cases were handled in two ways. In some of these cases, the respondent had merely neglected to end elegantly or to enter the last few questions. They may have exited through the browser rather than through the Web software. In these cases, staff could enter the case, navigate to the last question, and the case would be coded as complete. The other almost-complete cases could be considered critical-complete item cases and were handled as such.

5.3 Abbreviated CATI Interview

In the last 7 days of the survey, interviewers could offer to conduct an abbreviated interview on a CATI case if the respondent indicated lack of time for a full interview. They could go to a parallel tab and indicate a switch to an abbreviated mode. The instrument would at that point ask all critical items but require only the critical-complete items. However, interviewers could still collect all data if the respondent agreed to a full interview.

6. Blaise Features and Programming Techniques

The NSRCG Blaise instrument relied on native Blaise system features, (extensions of) commonly used programming techniques, and clever uses of existing language constructs to achieve the flexibility needed for this multimode survey.

The Web capability was achieved with the C2B system from CentERdata from the University of Tilburg in the Netherlands. C2B is a 3rd party add-on that makes use of the Blaise API. MPR is appreciative of the opportunity to use C2B to achieve the Web mode. CentERdata provided several new features and excellent support for this survey.

Limitations in browser technology, or the need to achieve some browser displays, occasionally forced adjustment to Blaise-language syntax and blocking conventions. Overall more Blaise blocks were declared than would otherwise be the case.

The standard Blaise windows interface was used for CATI and data entry, while the C2B browser interface was used for the Web. The fact that both interfaces were generated, not hand drawn, meant that the cost of producing the dual interface was very low. The extremely rich Blaise windows interface with its navigational, remark, and other capabilities, and its natural hooks to Blaise CATI management, was necessary for the interviewers and data entry operators to efficiently achieve their tasks. The browser interface was intended to be used on a wide variety of browsers with varying connect speeds, and can be considered to be a lowest-common-denominator display and collection solution, which while appropriate for this Web collection, has far less capability than the Blaise windows interface.

A field called *ModeOfProcessing* kept track of mode of administration. This tracking was maintained independently of actual Web or windows platform, and independently of Blaise keywords such as CATI, CAPI, and CAWI. As a result, for example, we could follow up an original Web case while in Blaise CATI and yet keep the Web routing and data definition for the case.

In order to allow empty values in both paper and Web modes, the Blaise EMPTY was used for all subject-matter fields. In CATI, edits were used to enforce an answer for all fields. These edits were turned off for follow up of paper, CATI, or Web; follow-up-specific edits were activated to collect critical item information. Double data entry and verification were achieved through Blaise-language programming. The Blaise LANGUAGES feature was used to declare a Web language separate from the CATI language.

The unique Blaise checking mechanism executed constant re-evaluation of route, computation, and answer throughout the case; that technology and the related End key and pop-up edit dialogs, were essential in the capability of conducting critical item follow-up and the abbreviated interview.

6.1 Multimode Survey Management

CATI management relied on Blaise out-of-the-box capability with significant MPR additional infrastructure to implement the MPR outcome coding scheme, links to *MPR Survey Manager*, modifications to case delivery, and appointment handling among other added features. *MPR Survey Manager* is an overall survey management system based on .NET technology and the SQL Server database system. *MPR Survey Manager* was used for locating, mailings, producing reports, and sending of email reminders among other survey tasks. MPR extended CATI management to account for three simultaneous modes of survey administration in one instrument and in one database. This included an expanded status coding system, adapted rules for sequencing status codes through attempts and outcomes, the ability to switch cases between modes, withholding cases from the call scheduler or putting them back into the scheduler depending on specific events in any of the three modes. Maniplus was used extensively as a supervisory review system and to implement data entry and critical-item follow-up control systems.

7. Lessons Learned and Advice

Following is a list of the three most important lessons learned from the development of the NSRCG multimode instrument, and a suggestion for a most fruitful area of future research.

7.1 Specify Early For All Modes

While each mode of administration must be considered on its own merits with respect to the proper conduct of the survey, such a multimode CASIC instrument is far easier to develop if CASIC specification formally takes into account all three modes before programming begins. Otherwise you may constantly revisit all the source code to accommodate it to additional modes.

Decide early in the process which browsers, connect speeds, and client configurations the Web mode should support. In a general population survey (such as the NSRCG) with a true Web alternative, it may be necessary to support a large number of situations. In order to achieve browser displays that will work for all respondents, you may not achieve rich presentations that are typically found in the workplace. Section 508 guidelines may also limit browser display options. (Section508 links are in section 9.1.)

7.2 Leave Time for Prototyping and Reworking

The rendering of a data collection instrument into such disparate modes as paper, CATI, and Web, is not necessarily a straightforward process. Adjustments to each mode are likely in order to achieve 'cognitive equivalence' between the modes in the mind of the respondent. These adjustments may include changes in question text, data definition, routing, fills, and so forth. Thus it may be necessary to leave time for prototyping and assessment, and reworking of the instrument. As each mode is implemented, retesting of previously implemented modes is required to make sure existing capability is not broken. It is likely that conducting a multi-mode Web survey will be new to most or all staff; the survey sponsors, the project staff, the programmers, and the operations personnel. It is necessary to keep open the lines of communication throughout the development and survey-conducting process. As this capability is developed in the survey industry, all staff involved should expect that there will be revision and re-consideration of implementation, and that this is a natural learning process that takes time.

During development of the instrument, we conducted extensive testing to ensure that the routing was correct, that skips and fills were working properly, and that updates did not introduce other problems. Over the course of the field period, we made revisions as necessary, resulting in over 20 iterations of the production instrument. All revisions were made in a test environment to

make sure the live instrument was not damaged. Following each revision, we tested all three modes (CATI, Web and the data entry instrument) to make sure that changes in one mode didn't adversely affect the proper functioning of another.

7.3 Consider Effects on Survey Management

Survey management must be capable of handling all three modes simultaneously. In a survey such as NSRCG, it is the respondent who has the choice of mode. The call scheduler must be adjusted to account not only for calling outcomes, but also be able to turn off cases that have just been accessed or completed in Web mode, or a paper questionnaire that has just been checked in. There are times when a CATI respondent expresses a preference for another mode and the CATI management system must release the case immediately to the Web mode. At the same time, a follow-up with such respondents should be scheduled in case they fail to complete in another mode. There are numerous details associated with efficient multimode survey management. This is not trivial and is something you must plan for, develop, and test before production. You should be flexible enough to be able to emphasize one mode over another as the survey progresses.

7.4 Research is Needed in Staging Modes of Administration and Incentives

A fruitful area of research in such a multimode survey is the proper staging of the modes given the nature of the survey and the survey population, including the use of incentives. The possible combinations are numerous ranging from simultaneous release of all modes to using one or two modes to follow up a primary mode. A young, internet-savvy population should be approached differently than a mostly-elderly population for example. The use of incentives per mode should also be investigated including differentials between modes. The degree to which one mode substitutes for another is also worthy of research. The reasons to offer multiple modes are to accommodate respondent preference, to increase convenience, to target specific populations, to reduce costs, to accelerate data collection (Web allows this), and to raise overall response rate. The ideal product of such research would be a parameterized cost function that could be applied to CATI/Web/paper surveys to help determine the optimal mix and staging of modes.

8. Summary

The conduct of the NSRCG in paper, CATI, and Web in Blaise was a challenging yet successful and rewarding experience. Many of the challenges were in the nature of a learning experience and nearly all developed solutions can be applied to future surveys. While it is more costly to develop a three-mode instrument than a one-mode CATI instrument, it should be far less costly than conducting the three modes in different software. Additionally, survey management capability must be expanded, yet this is still far simpler than combining survey management across separate data flows. The use of one instrument does encourage that specification for all modes be done up front. Further it is methodologically advantageous that specification of all modes and uses be done in concert, because differences and similarities between modes become the result of explicit decision, not an artifact of the use of different software.

9. References

Aquilino, W. (1994). "Interview Mode Effects in Surveys of Drug and Alcohol Use: A Field Experiment." *Public Opinion Quarterly.* 58:210-240.

Aquilino, W., Wright, D., and Supple, A. (2000). "Response Effects Due to Bystander Presence in CASI and Paper-and-Pencil Surveys of Drug Use and Alcohol Use." *Substance Use & Misuse* 35: 845-867.

Baker, R., Bradburn N., and Johnson, R. (1995). Computer-assisted Interviewing: An Experimental Evaluation of Data Quality and Cost. Pp. 413-431 in *Journal of Official Statistics*, Vol. 11, No 4, Statistics Sweden.

Groves, R., Berry, M., and Mathiowetz, N. (1980). Some Impacts of Computer Assisted Telephone Interviewing on Survey Methods. Pp. 519-524 in *Proceedings of the Section on Survey Research Methods*. Alexandria, VA: American Statistical Association.

Hart, L., Foster-Sardenberg, A., and Okada, Y. (2004). System Implementation for a Blaise Multimode Web, Cati, and Paper Survey. *Proceedings of the 9th International Blaise Users Conference.* Gatineau, Canada: Statistics Canada.

Mangione, T. (1995). *Mail Survey: Improving the Quality*. Thousand Oaks, CA: Sage Publications.

O'Reilly, J., Hubbard, M., Lessler, J., Biemer, P., and Turner, C. (1994). Audio and Computer Assisted Self-Interviewing: Preliminary Tests of New Technologies for Data Collection. Pp 197-214 in *Journal of Official Statistics*, Vol. 10, No. 2, Statistics Sweden.

Pierzchala, M. and Manners, T. (2001). Revolutionary Paradigms of Blaise. Pp. 8-26 in *Proceedings of the 7th International Blaise Use's Conference*. Washington, D.C.: Westat, Inc.

Tourangeau, R. and Smith, T. (1997). Collecting Sensitive Information with Different Modes of Data Collection. Pp 431-453 in Couper, M., Baker, R., Bethlehem, J., Clark C., Martin, J., Nicholls, W., and O'Reilly, J. (Eds.) *Computer Assisted Survey Information Collection*. New York, New York: John Wiley & Sons.

Wright, D., Aquilino, W., and Supple, A. (1998). "A Comparison of Computer-Assisted and Paper-and-Pencil Self-Administered Questionnaires in a Survey on Smoking, Alcohol, and Drug Use." *Public Opinion Quarterly.* 62:331-353.

9.1 Section 508 Website Links

Home page: http://www.section508.gov/

A summary of the law: http://www.section508.gov/index.cfm?FuseAction=Content&ID=3

Appendix A: NSRCG Instrument Differences Between Modes

This table displays differences between modes excluding question text differences.

Question	Paper	CATI	Web	Notes
Section A – Education Background				
Model row	P	C	W	**1/**
Screener		C W		ID verification and eligibility **2/**
A1	P C		W	Browser display and routing
A1a	P C		W	
A2_1	P C			Blaise coding for state / territory
A21			W	Browser drop-down for state / territory
A2_2	P C		W	Blaise coding for country vs. browser string type
A3	P C W			
A4	P C W			
A5_1	P C		W	Browser display and routing
A5_2	P C		W	
A6Branch		C W		Education coding. **C/W:** Auto-code on verbatim and/or ask series of branching questions. **P:** Enter 3-digit code.
A6Code	P			
A7	P C W			
A8	P C W			
A9	P C W			
A10	P C W			
A11	P C W			
A12	P C W			
A13Branch		C W		Education coding. **C/W:** Auto-code on verbatim and/or ask series of branching questions. **P:** Enter 3-digit code.
A13Code	P			
A14_1	P C W			
A15	P C W			
A16	P C W			
A17a **CB 3/**	P			**Paper:** Allow 3 degrees. - Sub-blocks in order of a-b-c-d-e. - Education coding: Enter 3-digit code for primary and secondary major. - Degrees entered in any order. * CC for sample degree, CB for other degrees
A17b **CC ***	P			
A17c **CC ***	P			
A17d	P			
A17e **CC ***	P			
A17c **CC ***		C W		**CATI/Web:** Allow 9 degrees. - Sub-blocks in order of c-b-a-d/e-d/e. - Education coding: Auto-code on verbatim and/or series of branching questions for primary and secondary major. - First degree is sample degree and is transferred in large part from the screener. * CC for sample degree, CB for other degrees
A17b **CC ***		C W		
A17a **CB**		C W		
A17d		C W		
A17e **CC ***		C W		
A18U	P	C	W	**CATI:** series of Yes/No. **Paper/Web:** Mark all. **CATI/Web:** Auto-fill and route past if no degree at level. **Paper:** Fields on route regardless of degree level.
A18G	P	C	W	
A19UBorrow	P	C	W	**CATI:** series of 4 numeric (currency). **Paper/Web:** Mark all. **CATI/Web:** Auto-fill and route past if no degree at level. **Paper:** Fields on route regardless of degree level. **Web:** Side-by-side browser display.
A19UOwe	P	C	W	
A19GBorrow	P	C	W	
A19GOwe	P	C	W	
A20	P C W			
A21	P C W			
A22	P C W			
A23	P C W			

Section B – Employment Situation				
B1 **CC**	P C W			
B1Convert		C W		**C/W:** Re-ask question for DK/RF (C) or if blank (W).
B2 **CC**	P C W			
B2Convert		C W		**C/W:** Re-ask question for DK/RF (C) or if blank (W).
B3	P W	C	P W	**CATI:** Series of Yes/No. **Web / Paper:** Mark all
B4	P C W			
B5	P C W			
B6Branch		C W		Occupation coding. **C/W:** Auto-code on verbatim and/or ask series of branching questions. **P:** Enter 3-digit code.
B6Code **CC**	P			
B7 **CB**	P C W			
B8	P C W			
B9	P W	C	P W	**CATI:** Series of Yes/No. **Web/Paper:** Mark all
B10	P C W			
B11	P C W			
B14CATI **CB**		C		**CATI:** B14 placed between B10 and B11. **CATI:** Series of Yes/No. **Web/Paper:** Mark all
B12	P C W			
B13	P C W			
B14 **CB**	P W		P W	**CATI:** Series of Yes/No. **Web DE:** Mark all
B15 **CB**	P	C W		**CATI/Web:** Auto-detect whether employer is educational institution.
B16 **CB**	P	C W		**Paper:** On-route even if B15 not answered to allow B15 back-code.
B17 **CB**	P C W			
B18 **CB**	P W	C	P W	**CATI/Web:** May auto-code based on B17. **Paper:** B18 and B19 on route if B17 blank.
B19 **CB**	P W	C	P W	
B20	P C W			
B21Branch		C W		Occupation coding. **C/W:** Auto-code on verbatim and/or ask series of branching questions. **P:** Enter 3-digit code.
B21Code **CC**	P			
B20Desc		C W		**CATI/Web:** Occupation description
B22	P C W			
B23 **CB**	P C W			
B24	P C W			
B25	P C W			
B26	P	C W		**Paper:** No auto-fill and skip. **CATI/Web:** Possible auto-fill and skip past.
B27 **CB**	P C W			
B28 **CB**	P	C W		**Paper:** No auto-fill and skip. **CATI/Web:** Possible auto-fill and skip past.
B29	P C W			
B30	P C W			
B31	P C W			
B32	P C W			
B33	P C W			
B33Detail		C		**CATI:** Salary categories if B33 = NonResponse
B34	P C W			
B35	P C W			
B36	P	C	W	**CATI/Web:** Can auto-code and route past based on previous answers.
B37	P	C	W	**P:** On route if B36 blank to allow back code to B36. **C/W:** Possible auto-code and skip based on previous answers.
B38	P C W			
B38Detail		C		**CATI:** Salary categories if B38 = NonResponse
Section C – Other Work-Related Experiences				
C1	P C W			
C2	P C W			

C3	P C W			
C4	P C W			
C5	P C W			
C6	P C W			
C7	P C W			
C8	P C W			
C9	P C W			
C10	P C W			
Section D – Demographic Information				
D1	P C W			
D2	P			**P:** Working full time; Working part time; Not working.
D2_1		C W		**CATI/Web:** Was spouse/partner working
D2_2		C W		**CATI/Web:** Working full time; Working part time.
D3	P	C W		**P:** Allow on route if D2 empty for possible back code.
D4	P C W			
D5	P C W			
D6	P W	C	P W	**Web:** Accommodate side-by-side browser display. Paper follows Web.
D7 **CC**	P C W			
D8	P C W			
D9	P C W			
D10	P C W			
D11	P C W			
D12	P C W			
D13	P C W			
D14	P C W			
D15	P C W			
D16	P C W			
D17	P	C W		**CATI/Web:** May auto-fill and route past first or second most important reasons. **Paper:** Always on route. May back code to D16.
D16	P C W			
D19	P C		W	Blaise coding for state / territory
D19_			W	Browser drop-down for state / territory
D19_1	P C		W	Blaise coding for country vs. browser string type
D20	P C W			
D21	P C W			
D22	P C W			
D23	P C W			
D24 **CB**	P C		W	**Web:** Browser display
D24Convert		C W		**CATI/Web:** Ask if D24 blank.
D23	P C W			
D26	P	C W		Routing difference due to (lack of) auto-fill.
D27	P	C W		Routing difference due to (lack of) auto-fill.
D28	P C		W	**Web:** Browser display
D29 **CB**	P C W			
D31	P C W			

1/ Rows usually represent question numbers not items. For example, B27 has 14 parts, each a separate item in a CASIC instrument. At times, rows represent sub-questions where there are differences at this level.
2/ The screener is a complex block with sub-blocks with some differences between Web and CATI modes.
3/ CC: Critical-Complete item. CB: Critical Callback item.

Appendix B: NSRCG Degree Grid on Paper

A17. **The next few questions ask about the degrees you received before October 1, 2003. Starting with your most recent college or university degree, please provide the following information for each degree you have at the bachelor's level or higher.** *If you have more than three degrees, report your two most recent degrees and your first bachelor's degree.*

MOST RECENT DEGREE	SECOND MOST RECENT DEGREE	FIRST BACHELOR'S DEGREE
a. From which school did you receive your most recent degree?	**a. From which school did you receive your second most recent degree?**	**a. From which school did you receive your first bachelor's degree, if not previously reported on this page?**
College or University Name	College or University Name	College or University Name
Department	Department	Department
City/Town	City/Town	City/Town
State/Foreign Country	State/Foreign Country	State/Foreign Country
b. In what month and year was this degree awarded? Month / Year	**b. In what month and year was this degree awarded?** Month / Year	**b. In what month and year was this degree awarded?** Month / Year
c. What type of degree did you receive? *Mark (X) one answer.*	**c. What type of degree did you receive?** *Mark (X) one answer.*	**c. What type of degree did you receive?** *Mark (X) one answer.*
1 ☐ Bachelor's degree (e.g., BS, BA, AB)	1 ☐ Bachelor's degree (e.g., BS, BA, AB)	1 ☐ Bachelor's degree (e.g., BS, BA, AB)
2 ☐ Master's degree (e.g., MS, MA, MBA)	2 ☐ Master's degree (e.g., MS, MA, MBA)	2 ☐ Master's degree (e.g., MS, MA, MBA)
3 ☐ Doctorate (e.g., PhD, DSc, EdD, etc.)	3 ☐ Doctorate (e.g., PhD, DSc, EdD, etc.)	3 ☐ Doctorate (e.g., PhD, DSc, EdD, etc.)
4 ☐ Other professional degree (e.g., JD, LLB, MD, DDS, etc.) – *Specify*	4 ☐ Other professional degree (e.g., JD, LLB, MD, DDS, etc.) – *Specify*	4 ☐ Other professional degree (e.g., JD, LLB, MD, DDS, etc.) – *Specify*
5 ☐ Other – *Specify*	5 ☐ Other – *Specify*	5 ☐ Other – *Specify*
d. What is the primary field of study and second major (if applicable) for this degree?	**d. What is the primary field of study and second major (if applicable) for this degree?**	**d. What is the primary field of study and second major (if applicable) for this degree?**
PRIMARY FIELD OF STUDY	PRIMARY FIELD OF STUDY	PRIMARY FIELD OF STUDY
SECOND MAJOR	SECOND MAJOR	SECOND MAJOR
e. Using the FIELD OF STUDY list on pages 18-19, choose the code that best describes the primary field of study and second major (if applicable) for this degree.	**e. Using the FIELD OF STUDY list on pages 18-19, choose the code that best describes the primary field of study and second major (if applicable) for this degree.**	**e. Using the FIELD OF STUDY list on pages 18-19, choose the code that best describes the primary field of study and second major (if applicable) for this degree.**
Code for Primary Field of Study	Code for Primary Field of Study	Code for Primary Field of Study
Code for Second Major	Code for Second Major	Code for Second Major
NOTE: Education codes range from 601-995	NOTE: Education codes range from 601-995	NOTE: Education codes range from 601-995

Appendix C: NSRCG Education Coding Branching on Web

A17d. Please indicate your second major for the master's degree you earned at in 2001:

psych

A17e. We need to assign a standardized educational code to the field of study you just listed. From the list belov please select one category that best describes the field of study for this degree.

- ○ Health
- ◉ Psychology, social sciences or history
- ○ Natural science, engineering, math or computer science
- ○ Education
- ○ Business
- ○ Humanities or arts
- ○ Or another field?

A17e. More specifically would you say...

- ◉ Psychology
- ○ Social sciences or history
- ○ Social work

A17e. More specifically would you say...

- ◉ Clinical
- ○ Counseling
- ○ Educational
- ○ Experimental
- ○ General
- ○ Industrial/organizational
- ○ Social
- ○ OTHER psychology

Appendix D: NSRCG Education Coding on Paper

Paper: Field of Study Verbatim and Code

A22. What was your primary field of study for this degree?

PRIMARY FIELD OF STUDY

_____

_____

A23. Using the FIELD OF STUDY list on pages 18-19, choose the code that best describes the field of study for this degree.

CODE ___|___|___

NOTE: Education codes range from 601 to 995

Paper: Partial image of Field of Study Code List

FIELD OF STUDY (Continued)

Languages, Linguistics, Literature/Letters

760 English language, literature and letters
771 Linguistics
772 OTHER foreign languages and literature

Health and Related Sciences

781 Audiology and speech pathology
782 Health services administration
783 Health/medical assistants
784 Health/medical technologies

Psychology

891 Clinical
892 Counseling
704 Educational
893 Experimental
894 General
895 Industrial/Organizational
896 Social
897 OTHER psychology

System Implementation for a Blaise Multimode Web, CATI and Paper Survey

Leonard Hart, Amanda Foster-Sardenberg, Yuki Okada /Leonard Hart, Mathematica Policy Research, Inc.

This paper addresses the systems implementation underlying the use of Blaise to collect and process data in three modes: CATI, Web, and mail. Using a single Blaise instrument, data are collected or data-entered in three modes simultaneously into a common database. This approach makes it more convenient for the respondent, requires less programming, eases data management, and results in simpler and more effective survey management. However, the integration of these three disparate modes presents several challenges for underlying systems. We outline challenges and their solutions concerning the programming of the Blaise multimode instrument, the rendering of the web mode, communication of web respondents with the Blaise database, and security.

1. Introduction

In the fall of 2003 Mathematica Policy Research, Inc. implemented its first true multi-mode Blaise CATI, Web and data entry instrument using one dataset. While Mathematica Policy Research had experience conducting multimode CATI/Web surveys across a combination of software platforms, data were not collected in one dataset in real time. Conducting multimode CATI/Web surveys in different platforms presented challenges such as keeping survey management in sync between the modes, making sure the latest data were in all datasets, maintaining various data-integration scripts, and producing valid reports representing true survey status. The shift to a one-system approach introduced new challenges such as re-thinking security to connect web respondents to the same database used by CATI and data entry. Other issues such as maintaining system performance and Blaise programming techniques are also discussed.

2. Challenges

Implementing the new unified architecture involved reevaluating Web security to meet the new demands of an open link to our internal network and the evaluation and testing of many software platforms. We installed new, more robust hardware to help meet these demands. Additionally, we learned several programming languages and we developed a new philosophy on how the instrument would be constructed and how questions would be displayed for the different modes. Not the least of these challenges involved bringing people together throughout the company to work together on implementing the new architecture.

Our first major step in overcoming this new data collection format was laying out a plan. This meant bringing key people together from each relevant group from the very beginning. This proved to be very helpful as it allowed everybody to become familiar with the concept of our integrated architecture and to speak up if certain known factors could impact other aspects of the project. An excellent example of this was bringing in the network security people at the beginning. As project suggestions were made that could affect security, the network staff was able to address the issue on the spot or suggest better methods of accomplishing the goal. They were able to maintain the right direction security-wise, even as people broke off into subgroups to work on their part of the project

3. Security and Systems Infrastructure

In this section, we briefly discuss the setup of hardware and software infrastructure that was done with the primary goal of ensuring MPR systems security as well as that of respondent data.

3.1 Security

The first major task we had to overcome was how we were going to have one secure dataset accessible from within the company network while allowing Internet respondents access to their data record from this same dataset at the same time. For propriety reasons we cannot go into a lot of details here but will hopefully give you some insight into some of the things we did to overcome these obstacles. The first rule of thumb about security is not to talk about it. All too often people write about their security setup such that users and hackers are given proprietary information or unnecessary clues about the network setup.

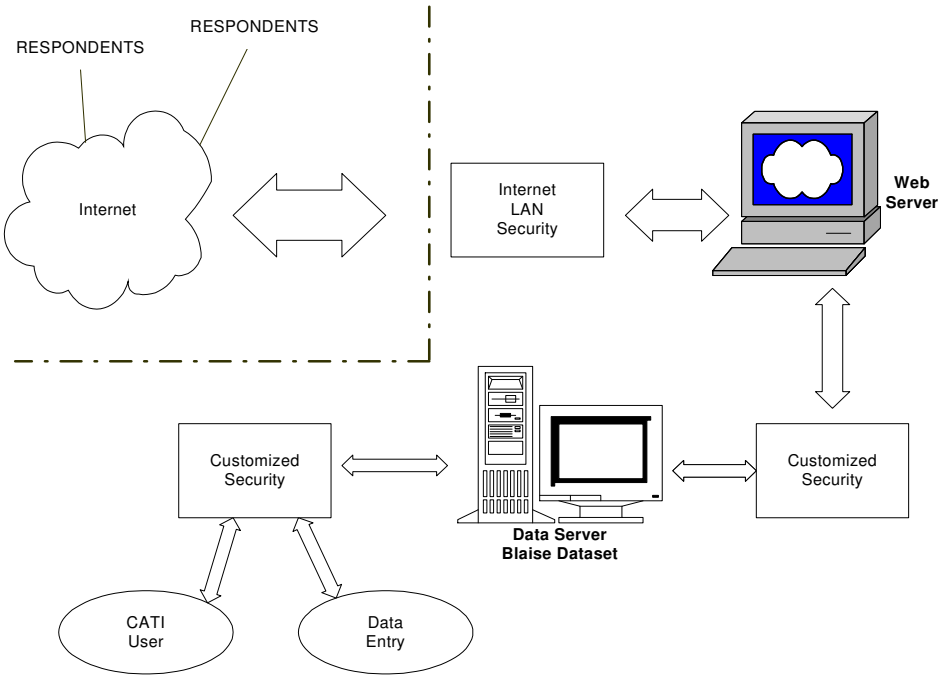
Setting up the new security for the web survey took some thinking and testing, but our past experience with other Web surveys helped us greatly to build a new and more secure system. For instance, we consulted with survey people about characters that should be excluded when generating passwords and how to manage passwords during the web survey. With network people, we coordinated our work for setting up the network configurations, web and data servers and their location, secure port and authentication, and getting the SSL certificate in place. Developers made sure that the data, passwords, and login IDs were kept secure and settings for the web server were correct and locked down. In addition, we decided to delete the first page where the respondents enter their login IDs and passwords before starting the survey so that the respondents or other people would not be able to go back to the first page and reinitiate the survey.

3.2 Hardware

Once the initial security theory was established the next step we took was looking at how the security arrangement would affect performance. This was a major concern considering our security implementation used several pieces of hardware and software to address our security needs. On top of this we wanted a redundant hardware/software solution in case something failed so the survey would still be available. Again we used our past experiences with Web surveys to look at what worked and did not work.

Since the previous Web servers were several years old we upgraded them to the latest technology. The new Web servers had dual processors, dual power supplies and Redundant Arrays of Inexpensive Disks (RAID) hard drive system. Since the price of hardware has decreased greatly the past couple of years we were able to buy some of the latest technology for just a few thousand dollars. The redundancy built into these new servers helped meet part of our goal to have a robust system. However, we knew that the powerful Web servers themselves would not buy us enough speed considering the security we were planning, and so we had to look at other areas. We were able to increase performance by offloading some services onto other servers. We knew this would help but we were concerned about putting more traffic on the network as machines talked with each other. We tackled this problem as a two-step process. The first step was to make sure the communications between the various machines took the shortest route between machines. The second part of this development and probably our biggest speed improvement was installing gigabit connections. We cannot say enough about the gigabit connection and how it increased the speed. The following diagram shows how everything came together.

Figure 1



3.3 Systematic Testing and Deployment Procedures

Once the new hardware, software, and security were in place testing became a very important part of the process. Testing was probably the most critical, ongoing, and multi-phased part of our NSRCG system-management process. It was critical to involve not only technical developers but also subject-area experts from the project in the testing effort. This was true regardless of the degree of specification detail and completeness, as it is the subject-area experts who are in contact with the client and thus who are in the position to best define how the instrument should work and whether any existing written or de facto specifications require modification.

In order to streamline the testing process with minimal production downtime, we had three servers on which we operated: the Alpha, Beta, and production servers. A standard procedure was developed and documented for deploying the updated instrument in each of the three environments, successively. This included not only development of backup and deployment scripts but also of testing scripts for CATI and Web to be used to check and review instrument upgrades simultaneously across modes. Development of basic testing scripts for each mode allowed for efficient and streamlined validation of basic functionality during instrument upgrades.

Once the functionality had been validated and the update approved, procedures were followed for the systematic deployment of any new updated instrument into production. Both initially and throughout, developers experimented with configuration settings on the Alpha server in order to tune performance and modify the interface for the web survey. After developers went through the checklist and tested the web survey on the Alpha server rigorously, the developers then moved the instrument to the mock-production Beta server. On this server, the environment, (e.g., server settings and folder structures) was identical (or nearly so) to that of the production server.

Project personnel could test once the web survey was deployed to the Beta server, and the version that they tested was generally either identical to the one on the production server, or else only one iteration ahead, in preparation for imminent production updates. Our testers performed extensive display and routing testing, as well as the mandatory Section 508 compliance testing. Only after extensive testing was completed, would the developers move the instrument onto the

production server. This systematic testing and deployment process minimized redeployment issues, and it prevented critical instrumentation and data problems that could otherwise have occurred. Minimizing downtime is a foremost concern for any survey, but particularly for web surveys that are active 24 hours a day, 7 days a week. Properly implemented and maintained, a systematic testing and deployment process allows for adequate input by the Project testers regarding instrumentation errors or oversight, as well as robust safeguarding with multiple rollback points.

3.4 Browsers

People use many different browsers and different versions of the same browser. There are reports stating that Internet Explorer has about 95% of the browser market but the target group for the NSRCG was recent college graduates. We figured this group of people would be more inclined to use non-IE browsers. Further complicating things was the fact that the sample was international and this meant the possibility of people living in countries where Microsoft products are less widely used. To accommodate these people with the widest varieties of browsers, we tested the instrument using Internet Explorer 5.0 +, Netscape Navigator 4.6 +, Opera, Safari and the newer Netscape and Mozilla/Firefox browser based on Gecko engine. There are hundreds of other browsers in the world but we felt if we address this core group of browsers listed above we would account for 99.99% of the people that have access to the Web.

Part of the HTML syntax for older browsers required some modification but we were able to display the web survey properly for the browsers listed above. To develop a browser-independent web survey was one of our many goals. In order to meet this goal, repetitive and careful rounds of testing across a range of browsers became a standard part of any testing and implementation of new instrument features.

3.5 Software

All three modes of the 2003 NSRCG survey were programmed using Blaise version 4.6 survey software and component pack. For the web survey, we utilized additional software, C2B Internet Service from CentERdata of the University of Tilburg in the Netherlands. C2B Internet Service supports all Blaise features, including don't know and refusal response options, parallel blocks, and Blaise tables. C2B Internet Service is easily configured with both Microsoft IIS and Apache. Using PHP and ASP scripts and C2B commands, the Blaise API was invoked throughout the web survey.

3.6 Languages

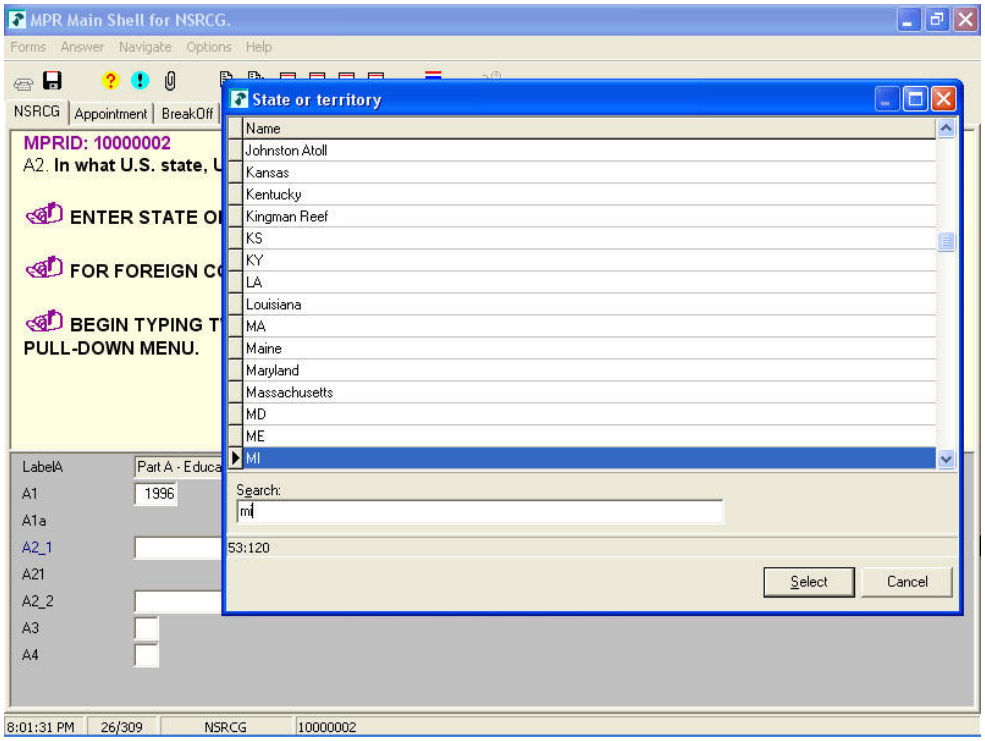
For the 2003 NSRCG web survey, HTML was used in the PHP scripts as well as in many parts of the Blaise source code to display somewhat complicated question layouts properly, including those involving various types of lists and tables. PHP script was mainly used to generate the Blaise questionnaire. JavaScript was also used in the PHP scripts for the client-side scripting. ASP was used for authentication with the LDAP (Lightweight Directory Access Protocol).

4. Programming Issues

There are differences in designing and programming one instrument for multimode CATI, WEB, and data entry, as describe in MPR's companion paper *Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey* by Pierzchala, Wright, Wilson, and Guerino. Here we discuss a few design and programming details that accommodated these differences.

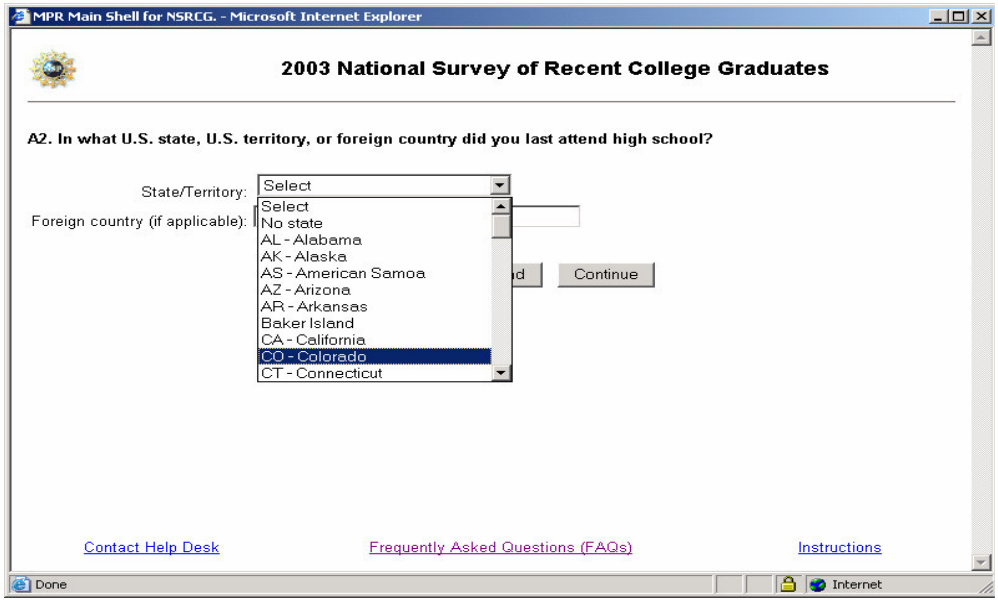
The first-person orientation of the multimode instrument must be kept in mind with regard to questionnaire presentation across modes. For instance, it is very useful for the CATI interviewer and data entry operators to have the capability to use a Blaise lookup table for entering the name of the state (see Figure 2). Interviewers can simply start typing the name of the state and see the lookup table with list of states that starts with the character that the interviewers type. This minimizes typing errors and speeds the interview.

Figure 2



On the other hand, using the drop-down box to display the name of the states to the respondent is more effective in a web survey (see Figure 3).

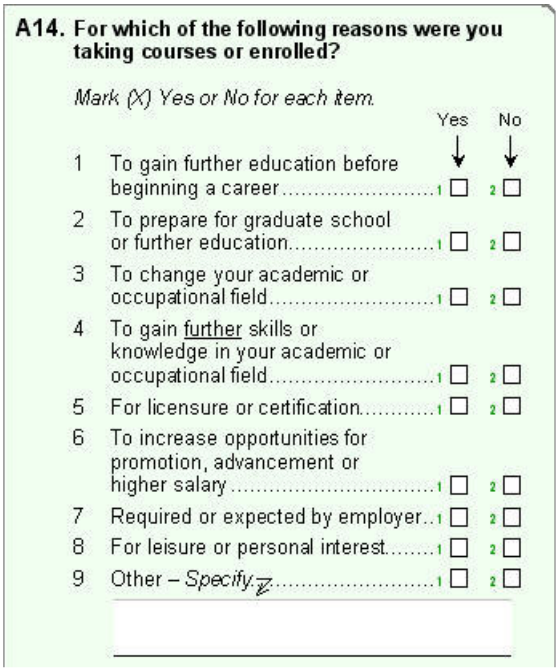
Figure 3



The CATI / paper state lookup was accomplished with a standard Blaise external file reference while the same question in the web mode used an enumerated type.

Even in a question that is cast similarly across the three modes, there are also some differences in programming the instrument for CATI and paper entry versus Web. Question A14 was presented as a series of Yes/No sub-items in all three modes. Figure 4 displays the paper version.

Figure 4



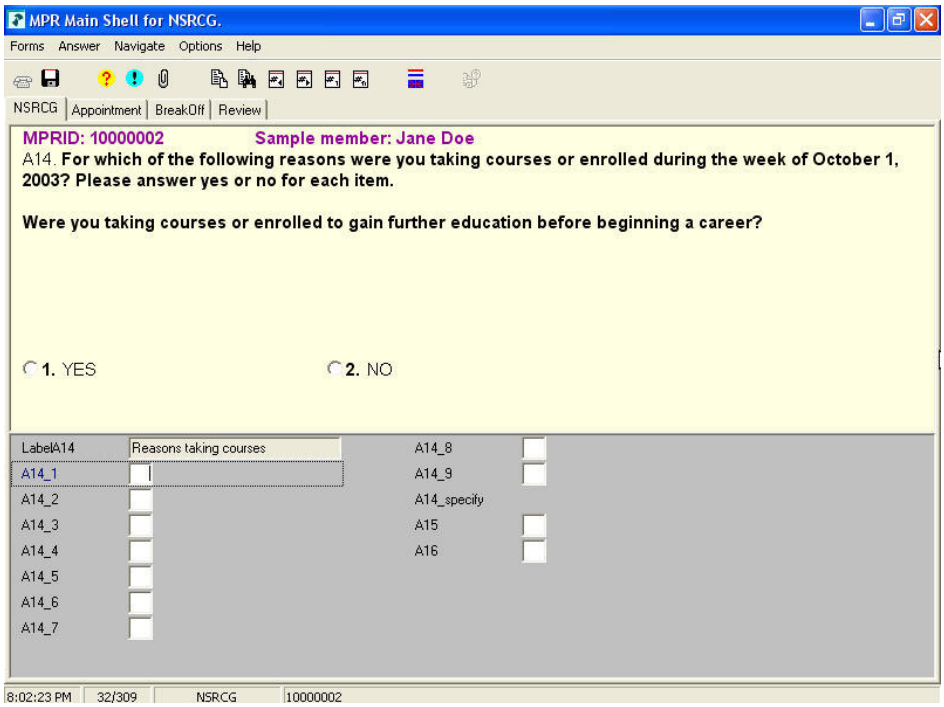
A14. For which of the following reasons were you taking courses or enrolled?

Mark (X) Yes or No for each item.

		Yes	No
1	To gain further education before beginning a career	1 ☐	2 ☐
2	To prepare for graduate school or further education	1 ☐	2 ☐
3	To change your academic or occupational field	1 ☐	2 ☐
4	To gain further skills or knowledge in your academic or occupational field	1 ☐	2 ☐
5	For licensure or certification	1 ☐	2 ☐
6	To increase opportunities for promotion, advancement or higher salary	1 ☐	2 ☐
7	Required or expected by employer	1 ☐	2 ☐
8	For leisure or personal interest	1 ☐	2 ☐
9	Other – *Specify*	1 ☐	2 ☐

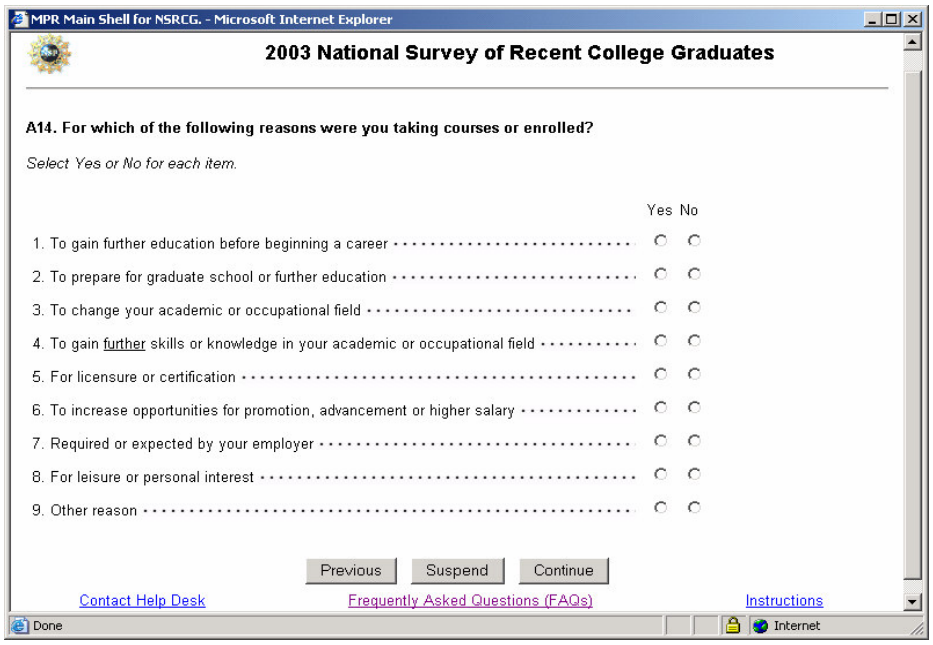
In CATI, interviewers read the group of items one at a time to the respondents; the interviewers saw the question texts in succession (Figure 5), but the data entry cells themselves were still displayed on one page of the Blaise Windows interface giving them the context of the complete question.

Figure 5



Web respondents saw the group of questions in one browser page (see Figure 6).

Figure 6



At times, there are noticeable syntax differences in developing the instrument for CATI and Web. For example, to display the first question in the multi-part web question numbered A14 above, the following Blaise question text is used at the block and within-block level:

```
FIELDS {Block-level, used for web browser heading}
  A14 WEB "<b>A14. For which of the following reasons were you taking
           courses or enrolled?</b>
          @/@/@ISelect Yes or No for each item.@I
          C2BTABLE(<table border=0 width = 80%>)
          C2BTD(<td align=center
          valign=top>)C2BNoJava()C2BBlockAsTable()C2BEnumColumns(2)"
      CATI "" : bSectionA14

FIELDS {Item-level within block}
  A14_1 {Education}
    WEB "<table border=0 width=100%><tr valign=center><td>
         <font size=-1><nobr>1.
         To gain further education before beginning a career
         </td><td width=100%
         background=images/dot.jpg></td></tr></table>"

    CATI "@AA14.@A For which of the following reasons were you taking
          courses or enrolled during the week of October 1, 2003?
          Please answer yes or no for each item.
          @/@/Were you taking courses or enrolled to gain further
          education before beginning a career?"
         / "To gain education": tYesNoList
```

We found that, with the current versions of software we were using, the HTML code must be included in the field text for the web survey in order to display the group of items with a nice, legible format. In general, if more than one question needs to be displayed per page in web survey, we needed to embed the HTML code in the field section for formatting purposes. Otherwise, we could simply develop the web instrument just as we would the CATI instrument.

5. Documentation and Testing

As mentioned above, we began with some of the core technical documents required by any project: detailed multimode instrument implementation instruction and testing checklists. A critical part of this core documentation involved further detail on how to set up and configure the C2B Internet Service on our network to successfully host a web survey. After documenting these processes and procedures, we also realized several additional documents were needed and would be vital to the ongoing success of our multimode survey. These critical documents spanned the gamut in their end-user orientation, from testers to developers to call center staff interfacing with respondents. In sum, these documents serve for reference, emergency, and training purposes, as well as baseline documentation for future web and multimode surveys that include a web mode.

Additional necessary documentation included user-oriented instructions for testers on our error tracking application that is accessed through a function key from within the test instrument. Additionally, we found that we as developers had an ongoing need to develop and maintain our technical documentation in greater detail as the survey evolved. Both for general reference and especially as the web commands became more specialized and survey-specific, we found we required additional documentation on how to embed the C2B web interface commands in the Blaise code, as well as detail on and examples of their specific usage. Some extremely useful technical survey management documentation was also added, specifically for the error tracking of reporting items or various general issues reported by either the call center or project management.

Third, and specific to implementation of the web survey mode, we found that fairly explicit documentation was required for the call center Help Desk staff that received trouble calls directly from web survey respondents. This document had to be geared to the respondents' "frequently-asked questions", or, issues of a technical nature that a respondent might encounter, for example, "how do I enable JavaScript?" or "how do I disable the pop-up blocker?" Moreover, the procedures for first-line troubleshooting and implementation of fixes had to be presented in a detailed and stepwise manner for the Help Desk staff. Backup contact information, and the implicit coordination of corresponding after-hours backup technical assistance (covering both instrument environment and network systems support), was also provided.

To properly test the survey across three different modes, one developer and one tester were assigned for each mode of the survey in order to test, fix, and maintain the instrument throughout its current cycle. Each mode of the survey has its unique flow, and thus it was very important that the testers be knowledgeable in such differences and be able to perform the testing in a thorough and efficient manner. Because the source code is shared by all three modes, it is possible to introduce bugs to other modes when the developers fix the source code in one mode. In multimode contexts such as ours, the importance of proper software versioning practices, mode-specific baseline testing scripts, and thorough testing of each new feature cannot be overstated. After making any changes to the code, staff tested the survey in each mode to verify that routing and display are correct for all three modes. In the early part of the survey, testers and developers both spent considerable time taking mock surveys in all three modes (web, CATI, and Data Entry) at the same time to confirm that modes could operate simultaneously. As our procedures and knowledge of the survey matured, it was possible at times to complete testing for an instrument update within an hour or two.

6. Systems Maintenance and Management

The instrument was updated for changes from time to time during live production. When changes were made to the instrument, designated testers conducted thorough testing in all three modes before the instrument was updated in production. While on one hand, updating the instrument for a CATI-only survey can be done before or after the hours of operation at the call

center; on the other hand, updating the instrument for a multimode survey with web is not as simple. Respondents can and may be taking the web survey at any time even more so for international respondents. As such, we analyzed the log files from the web server and C2B Internet Service to determine what time was best for interrupting the web survey in order to update the instrument and reboot the servers. We found that web activity was lowest between 2 AM and 7 AM every day; thus we usually updated the instrument early in the morning to avoid interrupting the web respondents. An update that did not require a data definition change could be accomplished in 2 seconds. In this situation, we did not bother to block access to web respondents, we would wait for the Blaise lock file to disappear and would copy 3 files into place virtually instantly. On the other hand, if a data definition change was required, the web access was blocked for a few minutes, with a nice explanatory screen, while the data conversion took place. We also found that activity is usually low on Saturday morning, so we routinely rebooted the servers at that time to clear memory and other resources. Call center staff would log onto a test web case every morning to ensure the web server was running.

We routinely checked that all hardware, including the servers and batteries, were properly maintained and fully functional. As part of this check, we monitored logs for any warnings, as well as the ambient temperature in the server room.

A batch file was run every night to back up important files other than database. The following files are saved in zip files and stored in the designated backup location: PHP scripts, image files used for the web survey, and any log files that are generated by C2B Internet Service and the web server.

7. Other Issues

Call scheduling for CATI was extended to take into account Web logins and completions and cases that were completed on paper and checked in. Cases that were accessed through the web had their record locked in the Blaise database and the Blaise call scheduler automatically excluded them from delivery. If a CATI respondent requested to complete by web, the interviewer would exit the case with the proper notation, and the case would be turned off from the scheduler for a number of days to give the respondent time to finish on the web. Once the interviewer exited the case, it was immediately available for web access to the respondent, including any data already collected. Any complete case, whether completed in CATI, Web, or paper, was denied to the respondent in the web mode. Cases that were started in web, but never completed, were followed up in CATI towards the end of the survey.

Operational, project, and client reports were derived from internal status codes and history files maintained by the instrument. Survey management data dumps were produced every overnight to be available to operational and project staff for investigation of survey management issues. For special investigations, Blaise history and log files, audit trails, C2B log files, and web server logs were used.

Other issues the technical team had to deal with were running an overnight process and linking to *MPR Survey Manager,* an overall survey management system. *MPR Survey Manager* was used for locating, mailings, sending email reminders, checking in paper questionnaires, and other things.

8. Summary

Bringing together three disparate survey modes into one Blaise instrument and database was a challenging technical exercise that was ultimately successful through careful planning, prototyping, and testing. Maintaining open and honest lines of communication between all MPR staff was an essential prerequisite for overcoming a variety of challenges. The fact that security was never compromised, that performance was always maintained in all modes, and that data were collected and processed successfully attests to the effectiveness of the effort.

9. Reference

Pierzchala, Wright, Wilson, and Guerino (2004). *Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey*, Proceedings of the 9th International Blaise Users Conference, Gatineau, Canada: Statistics Canada.

Mode effects and maintenance of data quality in a CAPI/CATI survey

Tim Burrell, Office For National Statistics,

1. Introduction

All ONS computer assisted social surveys use Blaise. As most ONS social surveys are carried out as CAPI, the standard is to design the questionnaire instrument with the ease of the face-to-face (FTF) interviewer in mind. The exception to this standard method of CAI design is the Labour Force Survey (LFS).

The LFS is a rotating panel survey containing five quarterly waves. The first wave is always CAPI, in part because the UK does not have an address or person register with phone numbers. Follow-up waves are usually carried out via CATI unless a telephone number is not given or available at the first interview. This survey is designed for the ease of telephone interviewing. The CAPI design is constrained to suit CATI interviewing. For example, there are no show cards in either mode.

The Annual Population Survey Boost Sample (APSB), like the LFS, has a panel element, in this case four annual waves. The majority of first wave APSB interviews are administered via CAPI. However, where a telephone number can be traced in advance (about 25% of cases) first wave interviews are carried out via CATI. All second and subsequent interviews are carried out via CATI unless the respondent insists on face-to-face interviewing.

The LFS caters for both CAPI and CATI within a single Blaise instrument designed for CATI. Other ONS social surveys have a different (ONS standard) single instrument structure which is more suited to CAPI. For the APSB, a single questionnaire instrument was designed which is flexible enough to use both modes of interviewing and aims to maintains data quality.

The LFS annually-independent sample components comprise, with the APSB, a major new output for the UK called the Annual Population Survey (APS). The APS is designed to measure key census-type variables annually between decennial censuses. It is a precursor for a more ambitious survey, larger in sample and number of variables, called the Continuous Population Survey (CPS) which will start in 2008. The CPS is described in another paper at this conference (Bumpstead, 2004).

In an era where response rates are falling on all household surveys, ONS are looking at ways of increasing or at least maintaining current response rates. One of these methods is to make more effective use of telephone interviewing. Although telephone interviewing is commonly thought to yield lower response rates than face-to-face interviewing for the same survey, it may have more subtle effects when viewed as part of a strategy for dealing with the entire portfolio of surveys. In this perspective, telephone interviewing for appropriate surveys or parts of surveys may provide more face-to-face interviewer time in the field for surveys for which this may make a critical difference to response. In addition, ONS seeks more of the traditional benefits of telephone interviewing: reduced costs.

This paper will briefly describe the structures of the current CATI-oriented LFS questionnaire instrument and the current CAPI-oriented ONS standard instrument structure. It will then examine the design requirements for a new survey instrument suitable for both modes of interview. It will also look at differences arising in how the questionnaire is structured for CAPI and CATI; at how cases can be moved seamlessly between the two modes as real-life situations demand; and at minimising the impact of interview mode on data quality.

2. Structures of ONS social surveys

2.1. The Labour Force Survey (LFS)

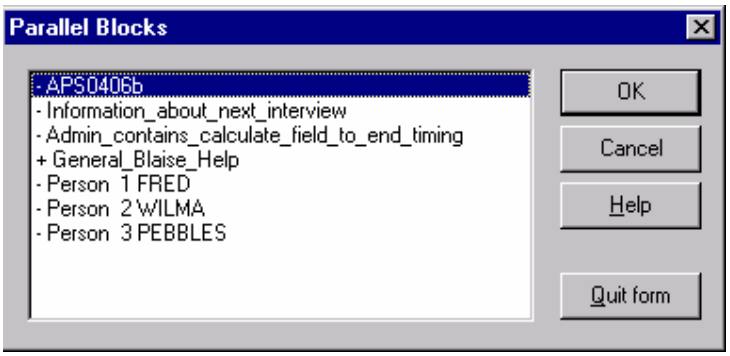
At present, the LFS is the largest single source of survey data available in the UK for monitoring purposes between Censuses. It produces estimates to a level of precision far exceeding any other UK household survey.

The LFS began production in 1990 and was programmed in Blaise I. It was the first ONS survey to adopt CAI. It was, therefore, programmed with simplicity as a strong design driver, given the constraints of 1980s software and hardware. As noted earlier, it was primarily designed to suit a CATI interview as 4 of the 5 waves are usually carried out in that mode. Though CAI tools have advanced and the LFS survey has adopted more up-to-date versions of Blaise (it is now programmed in Blaise 4.6) the essential structure of the instrument has proved highly resilient. Only now, a decade and a half later, do we feel the need to change it.

The design of the LFS questionnaire involves a small household section followed by an individual section for each adult in the household. The LFS type of interviewing structure is called *consecutive* interviewing: where an individual respondent's interview is completed before moving on to the next respondent. Individual interviews are accessible using parallel blocks within the questionnaire. This allows individual interviews to be accessed easily and quickly. At subsequent waves, provision is made to interview the household members in a different order from the first wave interview. Standard ONS social surveys, on the other hand, are designed not to allow access to individual's interviews through parallel blocks. LFS interviews, at a mean of about 30 minutes per household, are shorter in length compared with other ONS social surveys.

Figure 1 shows a screen shot of the box which appears when the option to access the parallel blocks is selected. There are options available to access each person's individual interview. These can be accessed at any stage during the interview, after the household interview section has been completed.

Figure 1.



2.2. Other ONS social surveys

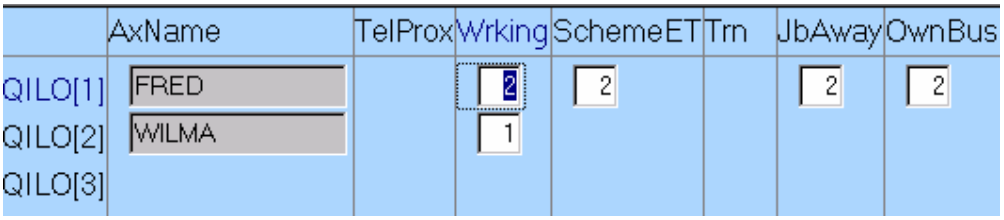
Virtually all other ONS social surveys are carried out via CAPI. For most surveys a field interviewer visits a household and interviews all members who are eligible for interview for the particular survey. These types of interview tend to cover a particular subject (such as expenditure or psychiatric morbidity) in great detail, so they are longer in interview length than the LFS. (Several have a mean interview length of around 90 minutes). In most cases a CATI design is not suitable. For many of them, as for the LFS, all adults in the household are eligible for interview (e.g. for the expenditure survey but not for the psychiatric morbidity surveys). The ONS standard instrument structure provides for interviewing all eligible members of the

household. Surveys which are designed to interview only one person in the household use a simplified version of the standard structure.

Where more than one household member is eligible for interview, the CAPI interviewer will attempt to interview as many of the household members as possible on a single occasion. The individual sections are divided into topic areas and the interviewer covers one topic area at a time asking all the individuals in turn before moving on to the next topic. This method (*concurrent* interviewing) maintains the momentum of the interview and retains the attention of all respondents. There is provision for interviewing household members in any desired order and to complete the household's interviews on another occasion for those members not present when the interview is initiated. Concurrent interviewing, with its frequent shifts between respondents, would not be suitable over the phone (unless information is provided by a single proxy respondent on behalf of all household members).

Figure 2 shows how the Blaise interview screen appears during CAPI using the concurrent interviewing method. The sections are divided into tables so that each section is asked in turn to each individual in the household. In this example, the topic block applies to Fred and Wilma but not to Pebbles (person 3 from Figure 1). When one (the order doesn't matter) household member has completed a section, this set of questions are asked of another respondent before moving on to the next topic. At the design stage for new surveys, much work goes into optimising the topic lengths to allow the movements between respondents and between topics to flow naturally.

Figure 2.



2.3. The Annual Population Survey Boost Sample (APSB)

The APSB began in January 2004. This is a new survey comprising data from all annually independent sample components of the LFS (165,000 achieved households) together with a new survey (called the APS boost sample, APSB) of approximately 65,000 achieved household interviews per annum in selected areas of England. The APSB covers a sub-set of the topics in the LFS. By combining the outputs from these samples, we obtain better local area information.

As mentioned previously, the APSB has an annual panel element. Subsequent waves are carried out by CATI and first waves interviews can be carried out either by CAPI or CATI depending on whether a telephone number is available. This mixture of interviewing modes required the development of a tool to suit the needs of CATI and CAPI, and, bring together the elements of concurrent and consecutive interviewing. It is important when doing this to minimise any mode effects which may occur from the different structures of interview and to also ensure that the data has the same output structure irrespective of the type of interview. In other words, the data should appear in the same order in the Blaise data file whether the interviewing mode was CATI or CAPI.

3. Development of the APSB questionnaire: design approach

When developing the APSB questionnaire, it was an objective to adopt ONS standards for social surveys. This meant developing a questionnaire based on the CAPI format.

CATI interviewers involved in the pilot had difficulties navigating the CAPI-format questionnaire and accessing individuals' interviews. These problems resulted in pauses in conversation which interviewers felt were awkward and unacceptable. Among the criteria for success for the new APSB structure was for the questionnaire to be easily used in both CATI and CAPI formats.

It was also important to develop the mixed-mode Blaise instrument so that the fields were ordered in the same way whether the interview was CAPI or CATI. This would ensure that the data from the two modes of interview could be extracted in the same way with no matching process after the data collection phase.

The questionnaire has a switch variable at the start of the interview to code whether the case is CAPI or CATI. The design of the questionnaire required different routing depending on whether it was being administered CATI or CAPI. Telephone interviews needed to route each block of questions in turn to one individual before moving on to the next person (consecutive interviewing). The routing of the CAPI interview allowed a new set of rules to be followed causing the questionnaire to be broken up into topics which are looped to each individual in turn (concurrent interviewing).

It was important that there should be no negative effect on the common data structure and that each question should be referenced once within the questionnaire. Our solution was a structure of arrays of arrays which would allow these requirements to be met. We were not deterred by considerations of the arguable inefficiency of such a design, which would result in a high proportion of empty places in the data structure. As there was no performance cost, the design was efficient in ways which mattered in practice, enabling us to work with the standards we have developed for instrument design across surveys in ONS.

The design is best understood by examining first the data structure and then looking at how it was achieved at data collection. In the following example, shown in Figure 3, we are dealing with a block called *QPerson* which is repeated for every person in the household and a block nested within it called *QHealth* which is also an array and which represents here any topic block nested within QPerson (i.e. all topic blocks at person level). The example is for a 3-adult household.

If the **CATI** mode is selected the questionnaire uses the person level array to enter data into the first array of the topic (QPerson[1].**Qhealth[1]**, QPerson[2].**Qhealth[1]**). In **CAPI** mode the household members are always routed to the first array of the QPerson block (**QPerson[1]**) and responses are entered into the different topic area arrays. To avoid filling the same fields and a 'display order conflict', the answers are coded into the topic area array person number *plus 1* (QPerson[1].**Qhealth[2]**, QPerson[1].**Qhealth[3]**).

Figure 3: Data structures for CATI and CAPI within a common framework of arrays of arrays as collected

CATI

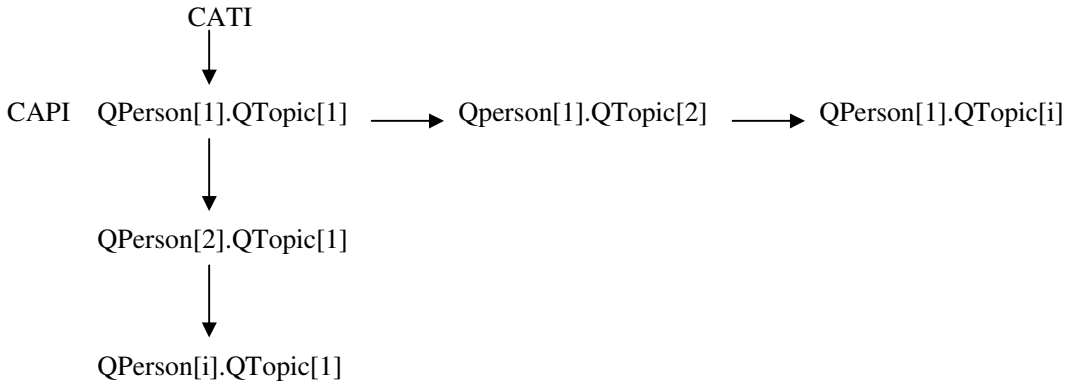
	QPerson[1]				QPerson[2]			QPerson[3]	
	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[4]	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[1]	Qhealth[2]
Mickey	,								
Minnie					,				
Donald								,	

CAPI

	QPerson[1]				QPerson[2]			QPerson[3]	
	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[4]	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[1]	Qhealth[2]
Mickey		,							
Minnie			,						
Donald				,					

To achieve this structure, the interviewers use different routing for CAPI and CATI, controlled by the switch variable. This is represented schematically in Figure 4

Figure 4: The CAPI/CATI data collection matrix for each member of the household



The data collected in the two modes then need to be integrated for output. This is done in Blaise (see below) to the format shown in figure 5.

Figure 5: Data structures for CATI and CAPI within a common framework of arrays of arrays: for output

	QPerson[1]				QPerson[2]			QPerson[3]	
	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[4]	Qhealth[1]	Qhealth[2]	Qhealth[3]	Qhealth[1]	Qhealth[2]
Mickey	**CATI**	*CAPI*							
Minnie			*CAPI*		**CATI**				
Donald				*CAPI*				**CATI**	

4. Development of the APSB questionnaire: implementation in Blaise

The block QPerson, referred to in section 3, is shown below in Blaise and is followed by a detailed explanation of each element within the block. The block shown is an extract from the APSB questionnaire and contains only 2 blocks: one block of health questions (QHealth) and a block of questions about caring for family members (QCarer). Figure 6 shows the block as a whole. The following figures comment in detail on sections of it.

Figure 6: Example of the APSB person block

```
BLOCK BPerson

   INCLUDE "QHealth.HAR"
   INCLUDE "QCarer.HAR"                    {These are the 2 blocks included in
                                            this example}
LOCALS { BPerson }

        LTLoop1          :  INTEGER        {LOCAL used for looping each topic block}

RULES { BPerson }

        IF QSignIn.IntrType = TU OR QTHComp.NumAdult = 1 THEN {Routing for CATI OR
                QHealth[1]                                     one person household}
                QCarer[1]
                ELSE
                IF QSignIn.IntrType <> TU THEN
                        QHealth[1] = EMPTY
                        QCarer[1] = EMPTY
                ENDIF
        ENDIF

        IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1 THEN {Routing for CAPI}
                FOR LTLoop1 := 2 TO DMHSize + 1 DO
                        QHealth[LTLoop1]
                ENDDO
        ENDIF

        IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1 THEN
                FOR LTLoop1 := 2 TO DMHSize + 1 DO
                        QCarer[LTLoop1]
                ENDDO
        ENDIF

ENDBLOCK   { BPerson }

FIELDS { DM }

        QPerson : ARRAY [1..16] OF BPerson
```

The routing for this block is determined at a higher level in the questionnaire. This block is an array and is repeated for each individual in the household. This allows parallel fields to be available for individual respondent interviews during CATI.

This block is split into 3 distinct sections – the include files and locals (figure 7); the rules for CATI (figure 8) and the rules for CAPI (figure 8). Each section is explained in detail below.

Figure 7: Include files and locals

```
BLOCK BPerson

    INCLUDE "QHealth.HAR"
    INCLUDE "QCarer.HAR"                    {These are the 2 blocks included in
                                             this example}
LOCALS { BPerson }

LTLoop1          :  INTEGER                 {LOCAL used for looping each topic
                                             block}
```

The name of the block is *BPerson*. This part of the code contains the person level blocks contain the topic questions which will be asked by the interviewer to each respondent in the household. Each of these blocks is an array. The local is used to loop the topic blocks during a CAPI interview.

The second section contains the routing for CATI.

Figure 8: Routing for CATI

```
RULES { BPerson }

        IF QSignIn.IntrType = TU OR QTHComp.NumAdult = 1 THEN {Routing for
                QHealth[1]                                     CATI}
                QCarer[1]
                ELSE
                IF QSignIn.IntrType <> TU THEN
                        QHealth[1] = EMPTY
                        QCarer[1] = EMPTY
                ENDIF
        ENDIF
```

The code to determine that this is CATI is '*QSignIn.IntrType = TU*'. This set of rules also applies if there is only one adult in the household '*QTHComp.NumAdult = 1*'. The rules are the same as the CATI rules if there is only one adult in the household as the sections will be asked in turn with no need to move on to a second household member.

If the interview is CATI or for a single person household, all topic blocks are completed before moving onto the second person's interview. In this cut-down example, Person 1 is asked the health and caring questions before moving onto person 2.

The interview order for a 2 person household is:

Person 1, topic 1
Person 1, topic 2
Person 2, topic 1
Person 2, topic 2

When a CATI interview is completed, data will be stored (as shown in the .bdb file) in:

Qwhoint.Qperson[i].Qhealth[1].Health

where *i* refers to the person number of respondent. *Qhealth[1]* refers to the first array of the block of health questions. Qwhoint contains the switch variable for CAPI or CATI.

Below is the code used for a CAPI interview of a household containing more than one adult. The code to determine this is: *QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1*

Figure 9: Routing for CAPI

```
IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1 THEN {Routing for
        FOR LTLoop1 := 2 TO DMHSize + 1 DO                CAPI}
                QHealth[LTLoop1]
        ENDDO
ENDIF

IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1 THEN
        FOR LTLoop1 := 2 TO DMHSize + 1 DO
                QCarer[LTLoop1]
        ENDDO
ENDIF
```

To avoid a '*display order conflict*' error the exact same questions as those used during the CATI interview can not be referenced again. With the first array for the topic questions filled, a CAPI interview will populate the second array of questions

Each block is referenced separately with the same set of conditions. During a CAPI interview, it is required that all eligible members of the household are asked one topic before moving onto the next topic. The interview order for a 2 person household is:

Person 1, topic 1
Person 2, topic 1
Person 1, topic 2
Person 2, topic 2

The responses will be coded into (as shown in the .bdb file):

Qwhoint.Qperson[1].Qhealth[i+1].Health

Responses are always coded into the first persons data *(Qperson[1])* and into a different array for the topic which reflects the person number of the respondent plus 1 – as the 1st array has already been referenced within the questionnaire.

This gives the problem that, at this stage, the data is not coded in the same place in the .bdb file if the interviews are completed in different modes. This is rectified by a piece of code at a higher level than the person block which recodes the responses given during a CAPI interview to the same place as they would have been if the questionnaire had been administered via CATI.

Figure 10.

```
FOR LTLoop1:=1 TO DMHSize DO                    {code repeated for each household
                                                 member}
  IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1 THEN     {IF CAPI}

    QPerson[LTLoop1].QHealth[1].QHealth1:=QPerson[1].QHealth[LTLoop1+1].QHealth1
    QPerson[LTLoop1].QHealth[1].GenHlth:=QPerson[1].QHealth[LTLoop1+1].GenHlth

     QPerson[LTLoop1].QCarer[1].Carer1q:=QPerson[1].QCarer[LTLoop1+1].Carer1q
     QPerson[LTLoop1].QCarer[1].Carer2q:=QPerson[1].QCarer[LTLoop1+1].Carer2q

  ENDIF

ENDDO
```

This code is repeated depending on the number of people in the household *(FOR LTLoop1:=1 TO DMHSize DO)* and applies if the interview is CAPI *(IF QSignIn.IntrType = FTF AND QTHComp.NumAdult > 1)*. This recodes the responses collected during a CAPI interview into the place where they would have been coded had the interview been CATI.

For example, looking in detail at the first line of code within the equation:

QPerson[LTLoop1].QHealth[1].QHealth1:=QPerson[1].QHealth[LTLoop1+1].QHealth1

The first half of the equation: *QPerson[LTLoop1].QHealth[1].QHealth1* refers to the place in the data structure where data would be coded during CATI. LTLoop1 refers to the person number of the respondent in the household. During CATI, data is stored in the person numbers block in the first topic array (*QHealth[1]*).

The second half of the equation: *QPerson[1].QHealth[LTLoop1+1].QHealth1* refers to where data is stored from a CAPI interview. During a CAPI interview data is always stored under person 1 (*QPerson[1]*) and under a topic array corresponding to their person number in the household plus 1, as the first array has potentially been filled by the CATI interview (*QHealth[LTLoop+1]*). This piece of code transfers the data from where it is collected via CAPI to where it would be collected via CATI.

This practice allows cases to be transferred between CAPI and CATI without overwriting any information collected at a previous interview in a different mode.

5. Results

After 7 months of interviewing on the APSB, and over 35,000 completed household interviews, there have been no problems with data collection. Cases have been transferred from one mode to another without data loss. Data from the first 5 months of APSB has been analysed and merged with LFS data, showing that the data structure from the CAPI mode of interviewing is consistent with the CATI mode used on the LFS. Interviewers working on either type of questionnaire have not experienced problems.

Analysing results by mode is problematic since the mode selection process involves confounding factors. The LFS has a random sample covering the whole of England. The APS sample is different as it is used to boost the numbers of interviews in certain areas of England to achieve improved local area estimates. This means that the APSB on its own is not a nationally representative sample like the LFS.

6. Conclusions and further work

This method of designing the questionnaire can be viewed as a success as it has allowed cases to be easily transferred from CAPI to CATI and vice versa without the loss or damage to data quality. Interviewers using the different methods have also had no problems. The complexities described here are transparent to them. From the design viewpoint, the complexities have been handled successfully within ONS's design standards and the requirement that any Blaise code should be understandable and understood by all its researchers.

At present, the APSB is the only ONS social survey which uses this design of Blaise instrument. There are very few social surveys which use telephone interviewing as a substantial part of interviewing. Most only carry out telephone reissues if a CAPI interview was not possible. If telephone interviewing becomes more common at ONS – especially with the prospect of the CPS (mentioned above), the design of Blaise instrument used on the APSB may become an ONS standard. Before reaching this stage, research is required into mode effects which may occur through the different methods. We will also need to assess more fully whether this is an efficient method of questionnaire design for the full range of ONS surveys.

Using CAWI in Household Surveys – Findings from Official German Statistics[1]

ANKE GERHARDT, Dr. Thomas Pricking, Statistical Office of North-Rhine-Westphalia
&
BIRGIT KUCHLER, Carola Kühnen, Anette Stuckemeier, Federal Statistical Office Germany

1. Introduction[2]

In view of the steadily-rising Internet use by private households and enterprises in recent years, empirical social and market research has discovered a new survey tool, namely Computer-Assisted Web Interviewing (CAWI). CAWI methods vary from genuine online surveys to online collection of data from respondents' recruited offline. To date, only official economic statistics have benefited from this development. Interviewing in official social statistics is still based either on Paper and Pencil-Assisted Interviewing (PAPI) or on Computer-Assisted Personal and Telephone Interviewing (CAPI/CATI). Hence, the German Federal Statistical Office and twelve out of 16 Land Offices decided jointly to extend the German Household Budget Survey ('Laufende Wirtschaftsrechnungen' or LWR in German) to include a feasibility study on CAWI for respondents recruited offline. The main focus of the feasibility study, referred to as 'Online LWR 2004', is to measure the acceptance of CAWI, as well as socio-economic characteristics and data quality of respondents interested in CAWI. The purpose of this paper is to discuss and summarise the initial findings of 'Online LWR 2004'.

The remainder of the paper is organized as follows: A short overview concerning the purpose and survey organisation of LWR is outlined in section 2. The objective and implementation of the feasibility study are explained in section 3. Section 4 describes the initial empirical results. The final section concludes and highlights the main findings (section 5). The presentation at the conference will include socio-economic characteristics and data quality of respondents interested in CAWI too.

2. Overview of the 'LWR' German Household Budget Survey

The LWR provide yearly cross-sectional information on private households' income and expenditure structures. Socio-demographic characteristics and household equipment with consumer durables will also be collected. The framework of the survey is private households. Since collection of data on households of self-employed persons and households with a disposable net income of Euro 18,000 and more is excluded by law, such households are excluded. The survey based on approximately 6,000 households. Data are collected in written form (PAPI: paper and pencil personal interviewing). The respondent in each household is the head of household. Each respondent is supposed to fill out two questionnaires:

- Questionnaire 'General statements' (Allgemeine Angaben) containing highly standardised questions on households' socio-demographic and socio-economic structures. Households receive the questionnaire at the beginning of each year.

[1] Author to whom correspondence should be sent: Anke Gerhardt, Landesamt für Datenverarbeitung und Statistik Nordrhein Westfalen, Email: anke.gerhardt@lds.nrw.de.
[2] The authors would like to thank Dr. Wolfgang Bandilla, Zentrum für Umfragen, Methoden und Analysen, Mannheim for their valuable comments concerning the actual state of the Online Research as well as Neil Mussett for the accurate proof-reading.

- Questionnaire 'Book of household accounts' (Haushaltsbuch) covering a time slot of three months. During this period, the household has to provide standardised information on monthly sources of income, monthly expenditure on housing and non-standardised information on daily expenditure items such as food, clothes and items of everyday use.

The LWR survey is based on a local data collection process. Hence, each Land Statistical Office takes full, independent responsibility for carrying out fieldwork and data entry. After data entry and data verification, all data will be sent to the German Federal Statistical Office, which will produce a national LWR data set. Hence, the collection of the online questionnaire data is also a local procedure.

We use BLAISE for Windows (B4W) in the German Household Budget Survey (LWR). This is a highly complex method, as it can be used for nine different functions. All these functions can be launched within the BLAISE application. Most of them are based on one of the nearly 60 "Manipula" programs written for this survey.

1. Storage of addresses and other information on households
2. Sampling procedure
3. Management of field work
4. Data entry and verification of data
5. Development, implementation and maintenance of a classification of products (based on Classification of Individual Consumption by Purpose – COICOP)
6. Data processing (computing of differences between income and expenditure, summarising)
7. Categorisation of households of results
8. Export of data to start computing weights
9. Import of weights and export

The basic BLAISE application has been equipped with additional features to import address information from other surveys, verify address data for double records, and define a special export format for generating bulk letters, and so on.

The households participating in LWR are selected as a quota sample. Before drawing the sample, the households will be sorted into 35 stages, constructed on the combination of the quota information. To realize the sampling procedure, the BLAISE application has been enlarged. The sampling procedure is realized as a sequential step-by-step method.

Management of fieldwork is a complex process. Addresses have to be corrected, letters and questionnaires have to be sent, responses and other feedback from the households have to be noted, and reports about fieldwork have to be produced at any time. For this purpose we have developed a central file, known as the "Household File", containing all relevant information.

Data entry and verification are the central features of the BLAISE application. Two types of data are collected: classified data of the questionnaire "General Statements" and income and expenditure data of the questionnaire "Book of household accounts'. Until this year (2004), the latter information was collected four months every year – one month each quarter. Beginning with a new concept for this survey in 2005, we will gather data in the future for one whole quarter from each household. This part of the survey is highly complex because households are

requested to give the information for each member of the household and for every day of the month in detail: costs for housing, travel, price and amount of goods and services and so on.

To verify the data, both parts of the survey have to be integrated – for every household, for every quarter and for the whole year. Verification means that household structure in every month must match either the status at the beginning of the year or the changes documented by the household itself. Verification also means that, for instance, the economic status of a person (or household) must fit the earnings type. Furthermore, the incidence of selected expenditure items is to be checked in relation to households' status – such as if rent is reported by all tenants. In fact, every record contains more than 7,000 fields.

The households report their daily expenditure items in plain text. To process the data, this plain text has to be classified. The list of products can be shown in alphabetical order or as a hierarchically structured list, including a "trigram coding" option. For coding, the BLAISE user will use a short description to find the product, but there is also a long description to differentiate between similar products.

Since 2002, income and expenditure items have been added up to form a monthly household budget (net result). Selected groups of products are totalled. Households have to be classified with respect to socio-economic variables. At the end of data entry, verified data have to be exported to start the weighting procedure. Data sets from all German Länder have to match before weighting. Subsequently, the weights will be imported into the original data set at Land level to publish the result. All these features are realized by a set of "Manipula" programs.

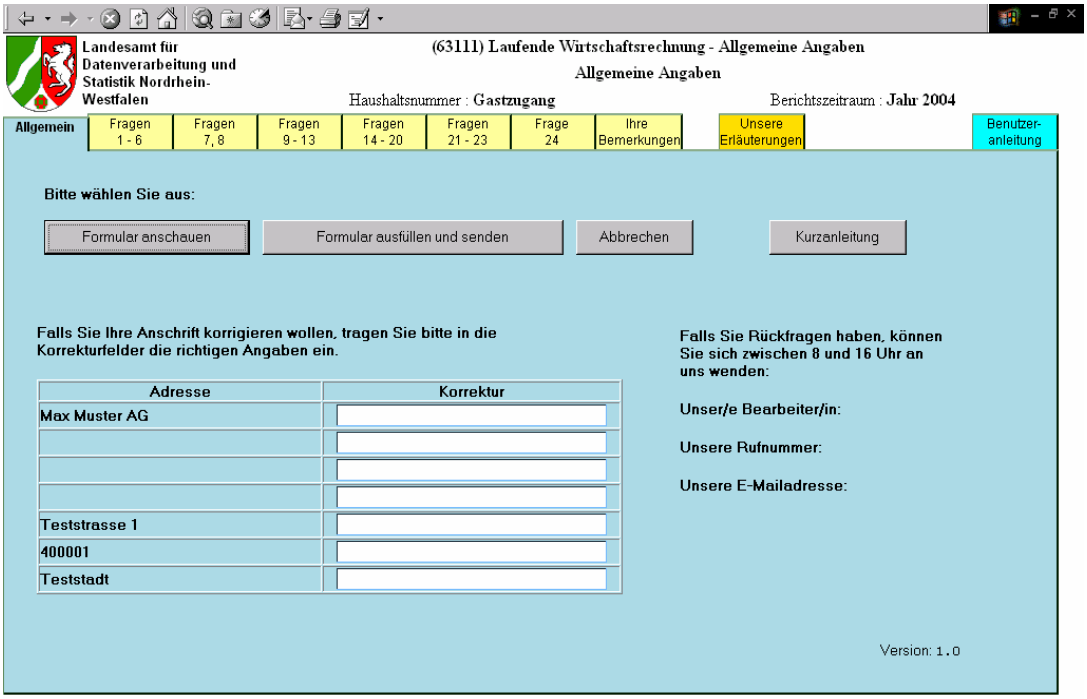
3. The Online LWR survey

In the discussion and planning of an online survey it turns out that the 'General statement' due to the length and structure of questions is suited as online survey. Given that the 2004 Online LWR constitutes the first attempt to introduce online questionnaires in German official social statistics, there is no precedent to help in designing a user-friendly online questionnaire to collect highly-complex data on the socio-demographic and socio-economic composition of households. It was impossible to use examples from market research because such online questionnaires mainly collect simple information on leisure or consumption conduct, and do not involve data checking procedures. Beyond this, LWR data are collected in written form and as such experiences of interviewer about LWR respondents were not available.

3.1 The Online questionnaire

For these reasons as described above, we have copied the widely-used graphic design of online questionnaires of German official business statistics. As the screenshot 1 illustrates design was based on several index cards.

Screenshot 1: Start index of the online form Online LWR 2004



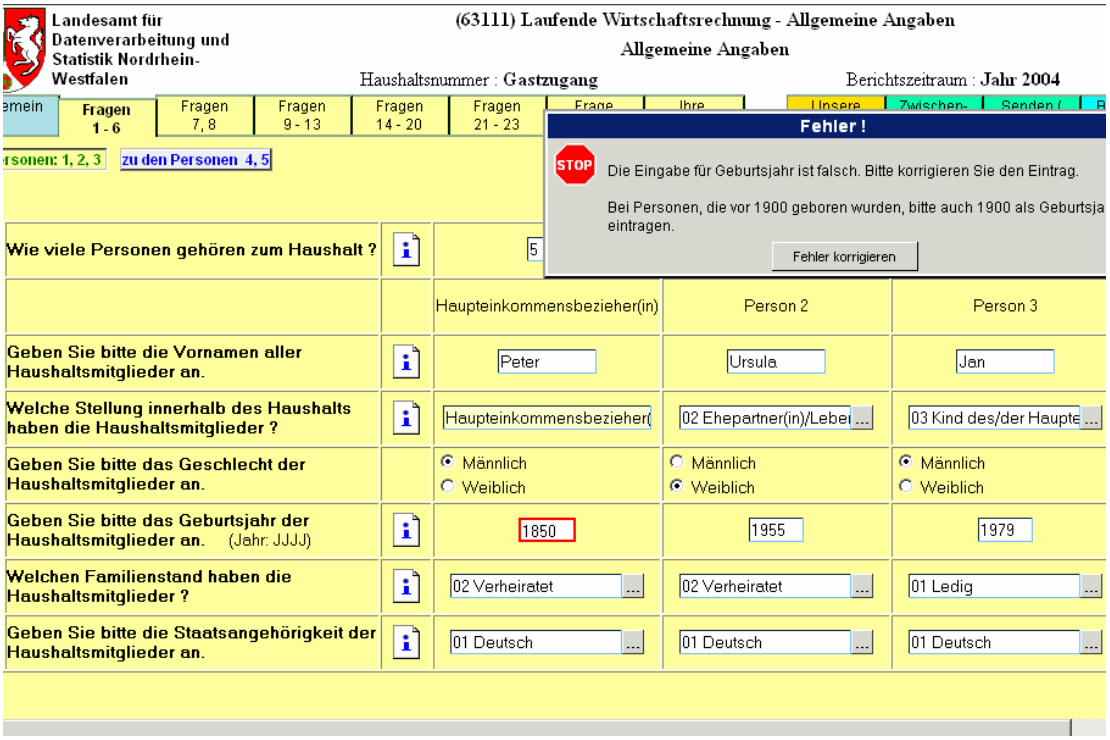
Each index presents a group of questions, accompanied by buttons providing necessary background information, in order to understand the whole intention of the question in case of doubt. Since each index card includes a headline, the principle pursued in completing the online questionnaire should be easily understood, and indeed should be self-explanatory. Once the selection procedure has been carried out, the first index of the online form is displayed. Here, the respondents find the relevant contact address and phone numbers of the Statistical Office. On the same index, they will find their own contact data, which they may amend.

Moreover, respondents can choose between viewing (left button ‘Formular anschauen’) and answering the questions (biggest button ‘Formular ausfüllen und senden’). If they decided to view only, they can browse through the whole questionnaire filling in the fields, but at the end hey can't transmit the data. This read-only option is useful to look for information about the survey and to decide whether any documents or other items are needed to answer the questions correctly. It naturally reduces the risk of transmitting an empty online data set but does not avoid it completely. The solution found was to examine at least formal data verification concerning all answer fields (completed/left empty) before starting the transmission process, and to install a ‘View online questionnaire’ button excluding the ‘Transmit data’ index card. The problem remains as to how to handle an instance in which a household has transmitted more than one online data set. Given the relative lack of knowledge regarding respondent conduct in online surveys, we resolved this question pragmatically, and only registered the count of returns per household, merely considering the first transmission. Further transmissions per household have been stored in a separate file (“cloned file”). The identification numbers of

households sending twice or more often have been listed in a schedule. The BLAISE user who is checking for double records could decide to change the records.

If respondents wish to fill in the questionnaire and send the data, they must select the answering mode (biggest button). After choosing the answering mode, respondents are forced to fill in all fields of one page/ index before it is possible to proceed to the next page/ index. That means that no question can be skipped (see screenshot 2). This mode – used for most questions – makes it obligatory to complete every field on the page.

Screenshot 2: Start index of online questionnaire 'General statements'



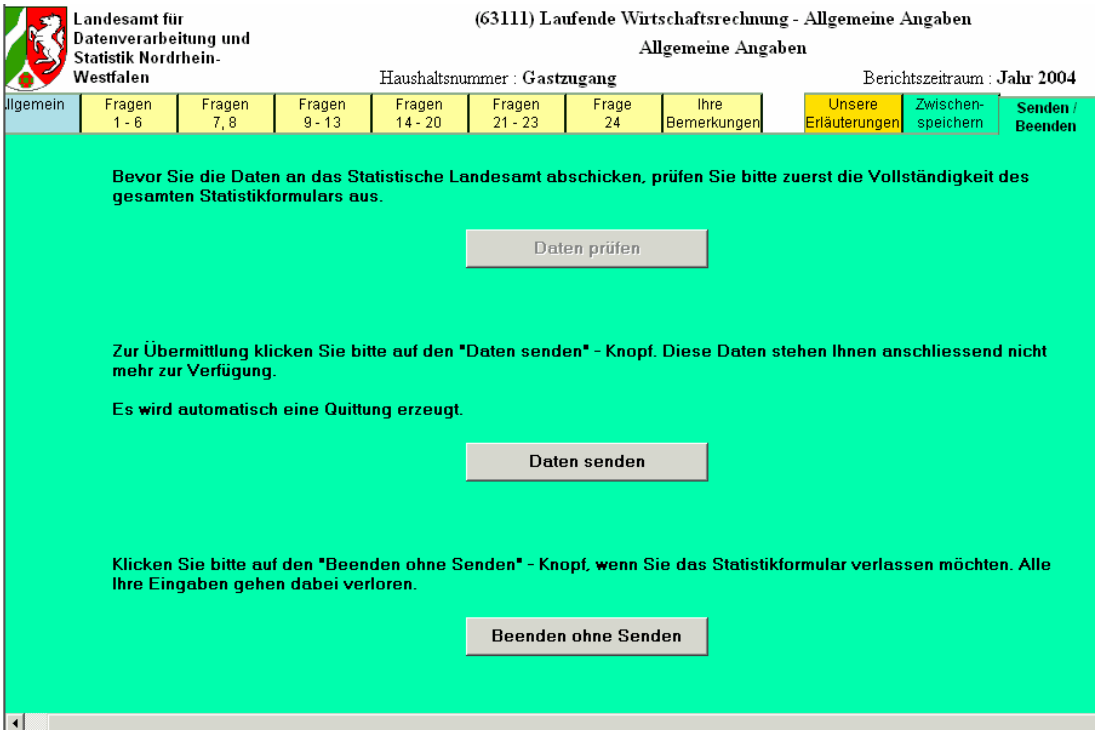
This is a somewhat inconvenient mode since respondents will be drawn back to the page they are leaving. But the greatest challenge was to introduce a data verification procedure during the filling out process without constraining or indeed hampering any individual filling out method or, what would have been even worse, producing unintended data or leading to problems in understanding should the respondent have difficulty understanding the question. Moreover, any additional restriction or verification method risks demotivating respondents due to the shifting of mainly financial resources from the statistical office to the respondent. However, if we introduced an extensive data-verification process during the filling out process, we could not be confident that we would not be creating a new problem by solving an old one: The technical opportunities offered by online questionnaires obviously often run counter to the needs of respondents. For that reason, we did without verifying the content of the data except for two questions (one exception is illustrated above and means that the birth year 1850 should be transformed at least in the birth year 1900 as minimum).

The second verifying process is not illustrated; it was used for the question about household equipment. Here, respondents had to provide information as to whether some selected goods are used by the households. This answer could be passed if at least one product had been named.

However, the main focus lies on measuring the acceptance of online questionnaires as such. Thus, we decided to analyse the quality of the data provided by ‘online households’ and ‘offline households’ once they had been received, in order to obtain representative empirical results as possible on this field.

If the respondents had answered all questions, data transmission could start (screenshot 3).

Screenshot 3: Last index of the online form Online LWR 2004



However, before it was possible to commence transmission, a data check had to be carried out. This check verified the formal correctness of selected data, e.g. if the field for the year of birth contained only numerical information. If problems were detected, the respondent was routed back to the index, a textbox was displayed and the field with the error marked with a red line. The cursor was automatically placed in this field. If there was more than one field to correct, the cursor passed these fields step by step. After amending the data, the respondent could proceed directly to the end of the questionnaire and repeat the check. The data had to be corrected prior to transmission. The reminders could not be ignored.

If the respondent was unable to fill in the questionnaire at once (or if he did not wish to do so), he could use a temporary storage option offered on the next to last index. Of course, this option was not associated with a check for plausibility of the data. At a later dates, when the respondent next logged in, the stored data could be reloaded and be accepted or changed. If he/she decided to transmit the data, it was necessary to go to the last index of the questionnaire to check and transmit the data. Once the data had been transmitted, a receipt for the data was sent to the respondent to store on his/her own computer.

3.2 The online data collecting process

When the households received the participation form via "snail mail", they were also informed about an Internet link where they could take a first look at the online questionnaire. At the next step the households received the individual password information (password and USER ID) via “snail mail”.

The conditions which households must meet in order to take part in the online survey are very simple: They need access to the Internet, browser software and Acrobat Reader to display comments on the questionnaire and helpful information. The "Statistik online" application runs on any system (MS Windows, Mac and so on). The browser needs the following configuration: It has to be enabled to run JavaScript and authorizing by password. Scripting is used to check if the questionnaire has been filled in correctly.

As shown in the description above, the households had to subscribe to use the whole sending option of the web application. If they did not do so, they could view the questionnaire but were unable to transmit their data. The first step in completing the questionnaire was to log on to the application. Then respondents had to choose the statistics and the year for which they wished to provide information. This general feature of the application was offered although it was only possible to select one set of household statistics and to dispatch the first year's data. However, this selection procedure is a part of the application which cannot be disabled.3

Data entry and the transmission procedure can be launched more than once. Respondents can therefore correct their own data as to whether relevant changes have taken place. The data incoming for the second or last time have been stored in a different file. This means that whenever a respondent sent data twice or more, the Statistical Office has two records: the first and the last. Fortunately, it emerges that only few households have sent several returns. The alternative, namely to delete the password after one return, seems overly restrictive within the scope of a feasibility study.

To secure the connection between the respondent’s personal computer and the server, data were sent with 128-Bit encoding and via HTTPS. Some firewalls were installed to protect the web application. The incoming data were immediately forwarded into the protected area of internal data handling.

4. Overview of planned analyses and initial empirical results

Since the data collection process is local, the recruitment and application processes are local too. The scope and forms of recruitment and application often depend on specific local financial resources and organisational structures. In one Land Office, some households were excluded from recruitment due to the age of the respondents because it could be assumed that they do not have Internet access at home. In another Land Office, the password and User ID were given together with the participation form at the beginning of recruitment. Surprisingly, the response rate in this Land Office was above average, and certainly higher than in other Offices. The date on which password information is sent obviously has a major impact on the response rate.

The first purpose of the feasibility study was to obtain empirical information on the acceptance of online questionnaires. We currently have initial information on the response rate and

3 It is required in business statistics because the same respondent (firm) may be obliged to provide information for different years, quarters or sets of statistics.

experience of the technical application. The 12 Land Offices has to recruit 5,000 households of the LWR survey. Out of 5,000 households, 1,028 were interested in the Online LWR 2004 feasibility study (18 percent). Given these response rate results, it was possible to conclude that only one household in five was interested in the online survey. On the one hand, we have to wait for the verified complete data sets of 'General information', containing information on households' Internet access because not all the 5,000 households may have Internet access. However, we already know from other, weighted databases that almost one household in two had Internet access at home in 2003. Comparing these two rates, we can therefore conclude that some households with Internet access decided against the online questionnaire in favour of completing a paper questionnaire and dispatching it via the traditional mail. However, the real response rate in the group of households interested in completing the questionnaire online (1,028) was after all over 80 percent (882 online respondents). When a recruited 'Online-household' did not complete the online questionnaire, the household received a paper version of 'General statements' (as the other 'Offline –Households') via snail-mail.

The second purpose of the feasibility study was to gain insights into the differences in data quality between CAWI and PAPI respondents. "High quality data" means that no additional effort is required to correct or clear the data. To measure this effort, we used the remarks option in BLAISE. Any time a record had to be corrected or completed, the BLAISE user should take note of why this work has been carried out and what effort this involved. The BLAISE user had to differentiate between 'completed/corrected without additional contact with the respondent' and 'completed/corrected after contact with the household'. Such empirical results will be represented at the conference.

The third purpose of the feasibility study was to get information on technical and organisational improvements. Since we could not observe respondents during the answering process, we naturally added an evaluation questionnaire on the 'Book of household accounts' paper questionnaire. The questionnaire contains 17 positive or negative statements on the graphic design as whole, on the principle of index cards, on the technical access conditions, filling out and sending procedures. For each statement, the household could choose between answers (1) applicable, (2) not applicable and (3) no answer. The third answer code could include the household refusing to give its opinion, as well as instances in which a household did not need or did not use the module (such as the possibility to save the given answer on the server). We have received completed evaluation questionnaires from 728 households, in which 687 questionnaires contain valid statements (codes 1 or 2) in the majority of fields.

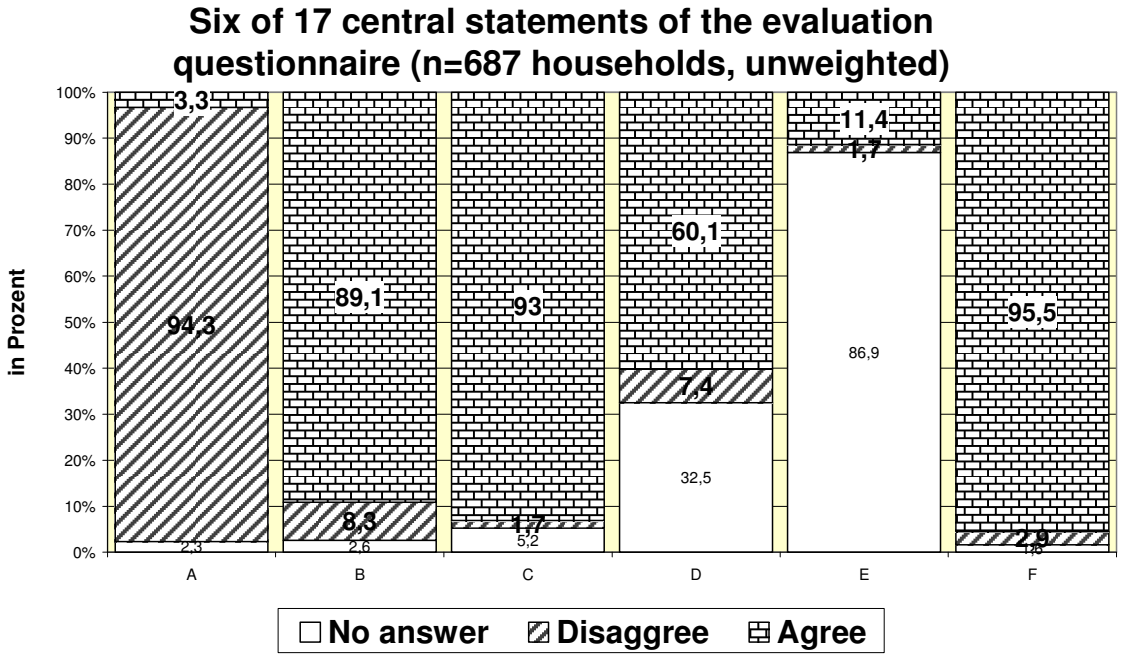
In general, the quantitative analyses of the evaluation questionnaire obtained consistently positive results on graphic design and completion method. More than half of private households gave solely positive answers in all 17 statements. Less than 10 households out of the 687 households reacted solely negatively to the online survey. The majority of households (96 percent) would answer the online questionnaire again. A minority of households reported technical problems as to access or to the transmission process.

Below, we present six of the 17 statements of the evaluation questionnaire illustrating the main experiences of respondents with regard to the technical application of the online survey.

A) I did not need to install additional software to view the online questionnaire.
B) I was able to view the questionnaire without any (technical) problems.

C) I like the index cards in terms of clearness. They make navigation within the online questionnaire easier.
D) The intermediate storage worked well.
E) I was always able to get hold of a contact person on the telephone hotline.
F) I would fill out the online questionnaire again.

The bar - diagram shows that about 95 percent of online respondents did not need any additional software (see bar A).
Bar – Diagram: Statements of the evaluation questionnaire



The majority of households (89 percent) were able to view the questionnaire without any technical problems (see bar B). Only 8 percent did not agree with the statement. The principle of index cards in online questionnaires worked well in household surveys, too: About 93 percent were able to complete the filling out process in this way (bar C). Over thirty percent of households obviously did not use the possibility of intermediate storage, as they neither agreed nor disagreed with the statement. However, 60 percent of online respondents agreed that intermediate storage worked well (bar D). Surprisingly, the rate of online respondents with no valid information on the statement about using contact persons is very high, at 86 percent (bar E). We expected more households to look for help and advice. However, the most important result is represented by the high rate of online respondents who would be willing to take part in the online survey again (bar F). On the basis of these empirical results, it is possible to conclude that the integration of an online survey part in the Household budget survey was successful.

The fourth purpose of the feasibility study was to analyse the socio economic characteristics of online and offline respondents. The recent empirical results will be represented at the conference.

5. Conclusions

From the technical perspective of using CAWI, the question of data checking procedures remains. In comparison with alternative collection methods such as CAPI (Computer Assisted Personal Interviewing), CATI (Computer Assisted Telephone Interviewing) or the classic PAPI method (Paper and Pencil Assisted Personal Interviewing), the major drawback of CAWI is that the first step of data entry is not accompanied by a data checking procedure. On the other hand, a final conclusion cannot be made until the data quality of the online respondents is analysed.

Furthermore, we concluded from remarks made on the evaluation questionnaires that some households used Internet access in public institutions, such as a city library, or on their PCs at work in order to answer the online questionnaire. From the individual remarks made on the evaluation questionnaires, we also learned of a few technical problems which we did not ask about on the evaluation questionnaire. For example, some households were unable to print out the receipt document containing the data that had been transmitted. A few online respondents would have liked to receive confirmation after sending the online questionnaire. It may be that the implementation of such a possibility will further reduce the rate of households sending data more than once.

6. References

ADM, ASI, BVM and D.G.O.F. (2001): Standards zur Qualitätssicherung bei Online-Befragungen. Ausgabe Mai 2001.

Bosnjak, Michael (2003): Teilnahmeverhalten bei Web-Befragungen – Nonresponse und Selbstselektion. In: Axel Theobald, Marcus Dreyer und Thomas Starsetzki (Hrsg.) Online Marktforschung. Theoretische Grundlagen und praktische Erfahrungen. 2. Vollständig überarbeitete und erweiterte Auflage. Gabler Verlag, Wiesbaden, 55-72.

Fischer, Iris (2004): Online- Erhebungen im Binnenhandel und Gastgewerbe. Erfahrungsbericht. In: Wirtschaft und Statistik 1/2004. (Hrsg.) Statistisches Bundesamt, 53- 56.

Pannewitz, Jana (2002): World Wide Web-gestützte Befragungen in der empirischen Sozialforschung. Ein Erfahrungsbericht. WZB Discussion Paper FS I 02-105, Dezember 2002, ISSN Nr. 1011-9523.

Pötschke, Manuela und Julia Simonson (2001): Online-Erhebungen in der empirischen Sozialforschung: Erfahrungen mit einer Umfrage unter Sozial-, Markt- und Meinungsforschern. In: ZA Information 49. Hrsg. Zentralarchiv für Empirische Sozialforschung. Universität zu Köln, 6-28.

Pötzsch, Olga, Jörg Decker und Carola Kühnen (2003): Private Haushalte in der Informationsgesellschaft. Europäische Pilotstudie zur Nutzung von Informations- und Kommunikationstechnologie 2002. In: Wirtschaft und Statistik 2/2003. (Hrsg.) Statistisches Bundesamt, 94- 105.

Pricking, Thomas (2002): Komplexe Anwendungen mit Blaise for Windows: Erhebungsmanagement und Datenbereinigung für die “Laufenden Wirtschaftsrechnungen“. In: Statistisches Bundesamt; Methoden – Verfahren – Entwicklungen, (Hrsg.) Statistisches Bundesamt, 3-13.

Pricking, Thomas (2003): Statistik-Online – Daten via Internet an die Statistischen Ämter des Länder und des Bundes. AWV-Informationen 2/2003: 7-8.

Reips, Ulf-Dietrich (2003): Web-Experimente – Eckpfeiler der Online-Forschung. In: Axel Theobald and Marcus Dreyer and Thomas Starsetzki (Hrsg.) Online Marktforschung. Theoretische Grundlagen und praktische Erfahrungen. 2. Vollständig überarbeitete und erweiterte Auflage. Gabler Verlag, Wiesbaden, 73–90.

Theobald, Axel, Marcus Dreyer and Thomas Starsetzki (Hrsg.) Online Marktforschung. Theoretische Grundlagen und praktische Erfahrungen. 2. Vollständig überarbeitete und erweiterte Auflage. Gabler Verlag, Wiesbaden, 55-72.

Some Ideas On Securing And Processing A Blaise IS Survey

Rob Groeneveld, Statistics Netherlands

1. Introduction

In Blaise IS, two modes of survey are possible: interview mode and form mode. In interview mode the respondent interacts with the survey in much the same way as a respondent who answers a DEP interview. In form mode, the respondent answers all questions and sends back the form to the Blaise IS server. No interaction occurs in form mode, for instance, no routing is possible and checks are limited to range checks. When using interview mode in a lengthy survey, the question arises: can a respondent come back to a partially completed form? This is possible if one accepts a slight security risk. This paper presents a few suggestions on how to keep the interview data confidential and enable resumability. Moreover, a diagram of a possible processing sequence for this kind of survey is presented.

2. Ensuring confidentiality and resumability

We distinguish security and authentication. *Security* means that the data sent over the Internet are inaccessible to others. This can be accomplished using the Secure Sockets Layer (SSL) protocol. *Authentication* means that only the person intended has access to certain specified data. This paper deals with authentication according to a certain method which can be characterised as 'Questionnaire based'. A number of other methods are possible for authentication, including methods using the facilities of the operating system. The advantage of the method presented in this paper is that it only requires an understanding of Blaise and Blaise IS. The disadvantage of this method is that it changes the datamodel — the datamodel is no longer identical to the model used in face-to-face and other types of interviewing.
The Secure Sockets Layer protocol encrypts data transported via the Internet, keeping them confidential. The SSL is an option in Blaise IS which should always be used. If a respondent is able to return to a previously partially filled form, stored in a record on the Blaise IS server, the issue of authentication is of even more concern. If one respondent can return to his form, other respondents can also retrieve that form, unless it is made difficult for them to do so.
We present here a technique whereby a key field is added to each record with individual ID numbers generated randomly, so as to have a key field which is difficult to guess. We suggest using at least 10,000 times the number of respondents as a reservoir from which to draw the ID numbers. The chance to match an ID number by guessing is thus comparable to the chance of typing the correct four-number PIN in an ATM with somebody else's bank card.
The Blaise setup used for the survey is to be extended with ID Number and Password fields. The ID Number becomes the primary key. The password provides additional security.The type TPassword is introduced in order to mask the input with a password character.
At the beginning of the datamodel we add these sections:

```
PRIMARY
  IDNUMBER
TYPE
  TIDNUMBER = STRING[12]
  TPassword = STRING[4]
FIELDS
  IDNUMBER   "Enter the 12-digit number that was sent to you":
    TIDNUMBER
  Password: TPassword
```

The type and field sections can be interspersed with types, blocks and fields of the original datamodel.

3. Generating ID numbers and passwords

We use Manipula to populate the added fields. A Manipula statement to generate an arbitrary string with length 12 containing only digits is the following:

```
FOR I := 1 TO 12 DO
  IDNUMBER := IDNUMBER + STR(RANDOM(10))
ENDDO
```

In which I is an integer auxfield and IDNUMBER is the field with the ID.
As a further enhancement to confidentiality, the password is generated randomly:

```
FOR i := 1 TO 4 DO
  Password := Password + CHAR(97 + RANDOM(26))
ENDDO
```

This command generates a password consisting of four lowercase letters. Both ID number and password are sent to a respondent by email, by normal mail or automatically using the Mail API or the Outlook library. When a respondent starts the survey, he can change his password immediately, so even people who happen to read the letter with the ID number and the initial password cannot reopen his record. The password change can be accomplished with normal Blaise AUXFIELDS and RULES:

```
AUXFIELDS
  EnterPW  "Enter your password":  TPassword
  ChangePW "Do you want to change your password?": (no, yes)
  NewPW    "Enter your new password": TPassword
RULES
  EnterPW
  Password.KEEP
  EnterPW = Password "You entered an incorrect password"
  ChangePW
  IF ChangePW = yes THEN
    NewPW
    IF NewPW = RESPONSE THEN
      Password := NewPW
      EnterPW := NewPW
    ENDIF
  ENDIF
  NEWPAGE { This protects the rest of the questionnaire }
```

The field Password is a datafield stored in the record. It is never asked. The field EnterPW is used to ask the respondent for his password. The check EnterPW = Password tests if the typed password is equal to the stored password. If not, an error message is presented and the password has to be typed again. The next field, ChangePW, and the next page(s) of the interview cannot be entered until the password is correct. If the password is correct, the next question is whether the password must be changed. It is best to change the password the first time the interview is run. If the respondent wants to change his password, the field NewPW is asked. When this field is filled in, the fields Password and EnterPW are updated so the check remains satisfied. The new password is stored in the record and must be entered correctly the next time the interview is opened. Only if the check is satisfied is it possible to proceed to the next page due to the NEWPAGE command. The survey questions proper start from this page. All password fields, normal and auxiliary, are of type TPassword, which is a string of 4 characters (in this case). This type is defined as a password type in the Datamodel Properties, the password character being for instance '*'. A PRINT file called Report.txt is also defined in the Manipula setup which reports the ID Numbers and passwords in a convenient format. The only field in this print file is a simple line of characters:

```
Line1: STRING[255]
```

The Manipula setup writes the Name, ID Number and Password to this field:

```
  Line1 := FORMAT(Name, 20) + IDNUMBER + '  ' + Password
```

Furthermore, some layout features are added to the report file. This file is of course only intended for the person managing the Blaise IS Survey.

Blaise IS must be configured in such a way as to retrieve only records with an existing primary key. This is done in the Data Entry Settings in the Blaise IS Workshop:

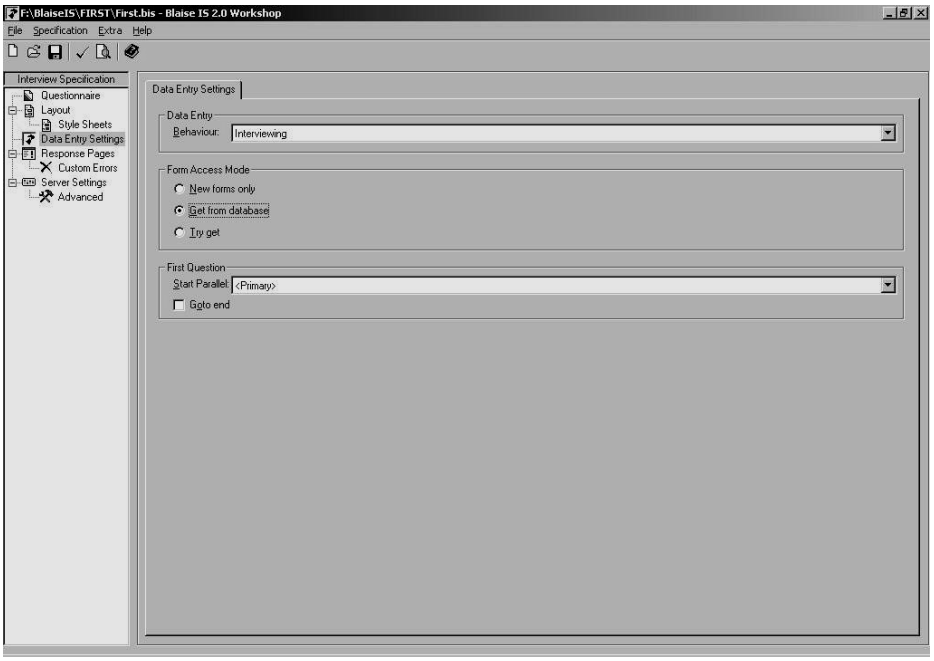


Figure 1. Data Entry Settings in the Blaise IS Workshop

With the option Get from Database the respondent is only allowed to access a form when his key is in the database. If his key is not in the database, he gets an error message.
All records generated with Manipula (empty except for the ID number, password and data available beforehand) are stored on the Blaise IS server in the Data subfolder. Only people who know a valid ID Number and password can access a record. If a respondent wants to leave the survey and come back to it later, he can again retrieve his record using the same ID Number and the (possibly new) password. Another setting is useful in this regard: 'Goto end' in the above screen, when checked, causes the questionnaire to jump to the first unanswered question.
There are a number of other methods for authenticating a respondent of a Blaise IS survey. More information can be found in the Blaise IS Online Assistant.

4. Processing a survey with ID Numbers

Assuming that some respondent data (for instance, registration number, name and address) are already stored in a table in a SQL Server database in the statistical office, the records can be read from this database with Manipula, extended with an ID Number and password as above and stored in a database which is used in the survey in Blaise IS. This can be a Blaise database, but it can also be another database: Oracle, SQL Server, MS Access or a similar system. The database can even be located on another server than the Blaise IS server. In the processing diagram below we assume the records which will receive the answers to the questions are stored in a Blaise database (.BDB file) on the Blaise IS server. The 'original' datamodel is called FIRST, the same model extended with an ID Number and Password is called FIRST_BL_IS.

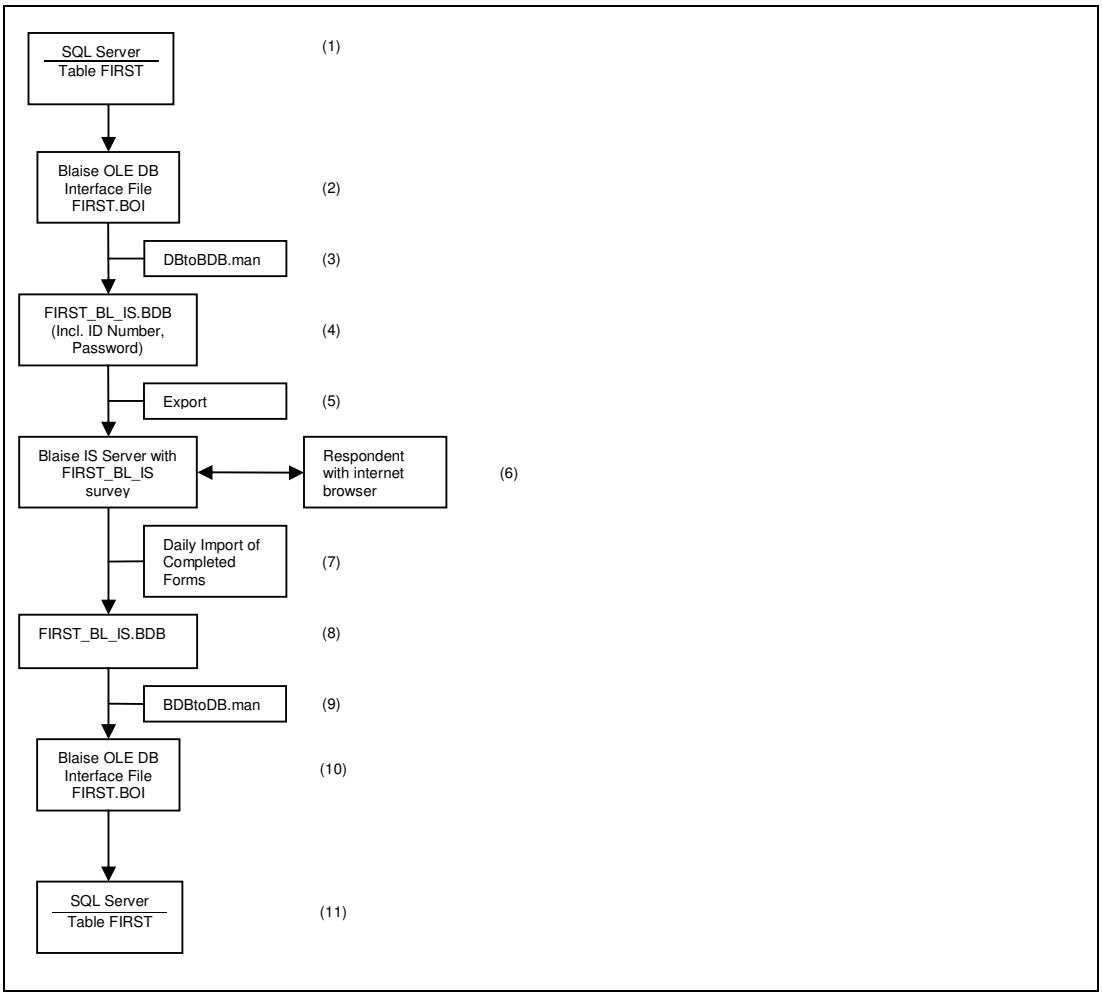


Figure 2. Flowchart: Processing a Blaise IS Survey with an RDBMS

The steps illustrated are:

1. A table on the SQL Server holds data on respondents, for instance, registration number, name and address. Our example table is called FIRST.
2. With the Blaise DataLink a BOI file is constructed which opens this table to processing with Blaise and Manipula.
3. A Manipula setup transports the data in the table to a Blaise database and assigns ID Numbers and passwords at the same time.
4. The result is a Blaise database with ID Numbers and passwords and the name, address and other data from the respondents database.
5. This database is exported (perhaps through a firewall or by email, depending on the ICT configuration of the office) to a database on a Blaise IS Server. This step is not necessary if the file (4) is on the Blaise IS Server.
6. The survey is run on the Blaise IS server. A respondent uses his internet browser to answer the questions in the survey. If necessary, he can come back to his record later.
7. The completed records are moved back to the Blaise database within the firewall and deleted from the Blaise IS server. Incomplete records remain on the server for some limited time, let's say, one week. Complete forms are inaccessible to other respondents once they have been moved. Depending on the configuration, this step can be a Manipula setup or a more elaborate step involving email etc.
8. The records are now on the Blaise database within the firewall.
9. The records are transported back to the SQL Server database, updating the fields in the original table with the data from the respondents, or storing the data in a new table, using

10. the BOI file of step 2.
11. The records with the completed fields are now in a SQL Server table.

5. Summary

We presented some ideas on authenticating users using an addition to the basic Blaise datamodel. Also, we described a possible process for a Blaise IS survey in Interview mode which uses this method. A respondent can access his own record, but not other records. He need not answer all questions in one session, but he can resume the survey using his own partially completed form for some time after first opening it.

Exploring the capabilities of Blaise IS

James O'Reilly, Westat, USA

1. Introduction

While questions remain about the effectiveness of web surveys, interest in web-based surveys has grown among researchers in government, private, and academic organizations. Some clients are pursuing exploratory efforts to learn whether the often discussed challenges--sampling, respondent cooperation, item non-response, self- rather than interviewer-administration, and the vicissitudes of the web environment--can be overcome or at least managed at a level that produces something useful. Others are interested in mixed-mode surveys, usually CATI and web. Still others want to recast existing web surveys: enhancing capabilities or increasing cost effectiveness, or expanding the scope.

Westat has integrated the internet into its survey work for a number of years, including conducting web surveys. Recently, growing demand and changing web research designs stimulated a company effort to examine our web survey processes and technology. One key goal is to evaluate and, where appropriate, implement web surveys using Blaise IS.

Westat's web surveys have been developed mainly using ASP, Microsoft's scripting environment for generating and processing HTML pages. ASP provides a powerful set of capabilities including easy integration with SQL Server and other relational databases. However, along with its power and flexibility, ASP is a procedurally-oriented, relatively low-level development language. This means custom programming and significant maintenance and support challenges.

Westat's ASP development team has built authoring systems to enhance our capabilities--generating much of the ASP code and reusing standard code modules. Still, it is clear that the ASP approach is problematic for some important types of surveys, for example multi-modal and more complex questionnaire designs.

Of course, all these things fit Blaise IS to a tee, in theory. Blaise is intrinsically multi-modal; it handles complexity naturally; its rules engine provide unmatched routing control; and our Blaise experience in CAPI, CATI, and CASI surveys could be applied to web efforts. But is Blaise IS ready to deliver, ready for prime-time with the architecture, scalability, usability, and extensibility required? That's what our effort was designed to address.

Our major activities were:

- User interface testing and experimentation
- Implementing XSL stylesheet extensions
- Exploring instrument development methods
- Developing a training course
- Designing and planning for server-side management and support systems

We will cover the first three of these here. Most of the work was done using IS 2.0 build 161 of 21 November 2003. As we discovered some of the limitation of IS 2.0, we awaited impatiently the release of the Blaise 4.7 beta so we could test some of the large and small enhancements. Since the beta release on July 30th we have been working intensively on version 4.7 and will report on that also.

2. Web interview user interface

In evaluating the user interface (UI) capabilities of IS, the core issue is that the user may often have very limited familiarity with computers, web browsers, and how to interact with them, whereas Windows Blaise UI is designed for and used mainly by trained, experienced

interviewers. We focused first on elements of relatively simple web surveys of individuals, using the NSSE2003 example. We found IS 2.0 handled well scalar items in dropdown lists, enumerated, string, numeric and open text. Also it implemented nicely grouped enumerated series of questions.

The technique implementing the enumerated grouping was a critical improvement to IS. It revealed how, using naming conventions for field response types in the standard Blaise datamodel, it is possible to instruct the XSL stylesheet to change the way one or more items are rendered in the HTML page sent to the browser. We discuss this in a later section.

We found problems with the usability of complex scalar items such as date and time fields. In Windows Blaise edit masks help the user understand what components of the field to enter. Masks are not allowed in the HTML text entry control. Again extensions to the XSL stylesheet are the solution.

Other UI problems in IS 2.0 included:

- Conditional items displayed on the next page, after a trip to the server
- Limited functionality for multiple language interviews

Both of these are fixed in Blaise 4.7.

Establishment survey prototyping raised other issues, including

- Limited flexibility for controlling the table display
- Difficulty with form-like entry of information with many fields on a page and more than one field on a line desireable.

2.1 XSL stylesheet extensions

The architecture of Blaise IS provides great benefits in terms of making the Blaise system's superior features--high-level, structured language using blocks and arrays to implement complex instruments; unmatched rules engine; capable proprietary database; and other elements--available for web interviewing. To do this IS controls the process much more strictly than other web system which typically allow the use of custom JavaScript and other modules to tailor and customize the web application.

The result until recently was that using Blaise IS meant accepting a more limited and dull user interface compared to what one could expect from an ASP or other native web development approach. For some web applications, such as CATI or data editing, with trained staff as users, the tradeoff seems easy to accept. But for many customers the most important part of web survey work is the ability to conduct self-administered surveys—having study subjects do the interview themselves.

For such untrained, voluntary users with widely varying web experience a limited and simple user interface is not likely to be desireable. Expectations of many clients on the look and feel of web applications are for polished, flexible pages.

Fortunately, the IS architecture uses XSLT (Extensible Stylesheet Language for Transformations), usually referred to as XSL. XSL provides a way in IS to extend and refine key functions and interface elements in web surveys. This extensibility potential is different from and in many ways superior to that available in traditional Blaise (DLLs or the BCP).

During a Blaise web survey each time the IS server is ready to communicate with the user's browser, IS uses the API to generate an XML (Extensible Markup Language) document with the appropriate data and metadata on the fields, texts, layout, and other facets of the current situation. The XML document's rich content provides such information as the field's type which allows the stylesheet to process in various ways.

The document is processed by the MSXML parser using the interview XSL stylesheet, biHTMLWebPage.xsl, or a substitute. This process transforms the XML information into the HTML page that is sent to the user.

API > XML > MSXML + XSL > HTML

Figure 1: Blaise IS server steps

As we prototyped surveys in IS 2.0, we identified a number of user interface elements that we needed but could not render with the system as-is. Our engineers studied the system and concluded that we could generate many of the requested elements by adding functions in the interview XSL stylesheet, following the model developed by the Blaise team for displaying a series of enumerated items as rows in a table with the question text in the stub and the enumerated choices as the columns.

Here's an example. In the datamodel TYPE section a TGroupExtent is defined (Figure 2) and applied to the four fields.

```
TYPE
  TGroupExtent = (large "Large extent", Moderate "Moderate extent", Small "Small extent",
    notatall "Not at all")
FIELDS
    BUILD_PREV "Build on previous R&D projects at your company?": TGroupExtent
    ENHANCE_PREV "Enhance the value of previous R&D projects at your company?": TGroupExtent
    BUILD_CURR "Build on other current R&D projects at your company?":TGroupExtent
    ENHANCE_CURR "Enhance the value of other current R&D projects at your company?":TGroupExtent
```

Figure 2: Grouping code

The interview stylesheet has been programmed to check for the answer types of fields, looking for a series of enumerated fields with the same type name and the prefix TGroup. When a set is found the stylesheet instead of displaying the series as individual items in sequence on the page, renders them in a table (Figure 3).

To what extent does your ATP project:

	Large extent	Moderate extent	Small extent	Not at all
Build on previous R&D projects at your company?	○	○	○	○
Enhance the value of previous R&D projects at your company?	○	○	○	○
Build on other current R&D projects at your company?	○	○	○	○
Enhance the value of other current R&D projects at your company?	○	○	○	○

Figure 3: Grouped enumerated display

This model made clear that a wide range of web-standard user interface elements could be added to Blaise IS using only the standard Blaise datamodel syntax and extensions in the XSL interview stylesheet.

We then began to explore the details of implementing this capability as we prototyped instruments and found web interview interface needs that could not be met as fully as we wanted using only the standard Blaise modelib.

2.2 Scales

Using the modelib one can display Likert-type scales horizontally but not with the labels displayed above the points rather than to one side. We requested this of the Blaise IS developers which they quickly provided. In the datamodel one uses a TYPE with a prefix of TScale (Figure 4) and a corresponding function in the interview stylesheet to produce the display (Figure 5).

```
TYPE
  TScaleAmbitious = (MuchLess "Much less ambitious", Less " ", SomeLess " ",equal "Equally ambitious",
    SomeMore " ", More " ", MuchMore "Much more ambitious")
FIELDS
    PROJ_AMBITIOUS "Relative to other R&D initiatives in your industry, how ambitious would you
      say are the overall goals identified for this project?":TScaleAmbitious
```

Figure 4: Likert scale extension code

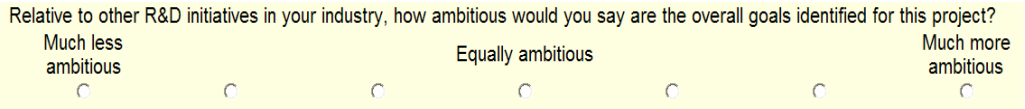


Figure 5: Scale display in IS

2.3 Quantity-Unit

Quantity-unit questions are important to many surveys. We need to know some quantitative information ("How much are you paid?") and the respondent may think of the concept in terms of so much per hour, per week, per two weeks etc. Using standard Blaise with an interviewer, the two elements are different fields and the interviewer may be able to complete both based on the response received to the first part. Or a follow-on question about the unit may be required.

In self-administration this compound concept can be challenging. So we wanted the visual representation to link the two fields more clearly than a two-step dialog displayed vertically on the page. We specified that two successive fields of types prefixed with TQuanUnit would be displayed side-by-side. Our XSL specialist added this to the stylesheet. The datamodel code and display are below.

```
TYPE
  TQuanUnit_Unit = (thousand, million, billion, noRev "None (No revenues last year)")
  TQuanUnit_Quan = 0..999999
AUXFIELDS
      introrev "What were total company revenues for your parent company last year?":string[1], empty
FIELDS
     RevAmount "Reported amount:"/"Reported amount:": TQuanUnit_Quan
     RevUnit   "Amount reported in:"/"Amount reported in:": TQuanUnit_Unit
RULES
     introrev
     revamount
     revunit
```

Figure 6: Quantity-unit code

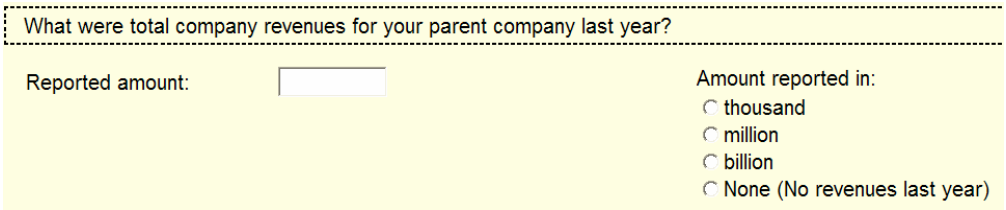


Figure 7: Quantity-unit display

Other user interface extensions added using the stylesheet include:

- Enabling two dual input lines to able to display assets and liabilities in two columns.

Please provide the following information from your company's annual financial report for last fiscal year.
We want a balance sheet format with Assets in a column to the left of Liabilities

Balance Sheet

Total Assets		Total Liabilities	
Current assets		Current liabilities	
Long-term assets		Long-term liabilities	
Owners' equity			

- Altering line spacing to show more elements on a page.

This information refers to the organization directly responsible for performir
verify the accuracy of the information listed below.

Company name	
Division or Unit Name	
Street address line 1	
Street address line 2	
Street address line 3	
City	
State	
Zip	

- Adjust the column widths of grouped enumerated sets so that the widths the same.
- Improved control of the width of text
- Increasing the size of the radio button and check box controls to improve ease of use
- Changing the menu and parallel tab look.

2.4 Applying custom stylesheet extensions

These extensions were added to the default IS 2.0 web interview stylesheet, biHTMLWebInterview.xsl, following XSL programming standards to prevent our references from conflicting or interacting with other functions in the stylesheet. This is done by defining the Westat namespace (Figure 8) so that all variables related to the extensions are identified by the namespace

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:westat="http//www.westat.com" version="1.0">
```

Figure 8: Westat namespace defined

To use the Westat extensions, developers start with the extended interview stylesheet. Working in the IS workshop and IS stylesheet editor the standard instrument customization is done—changing header and footer appearance, fonts, images, and the like—and saving a instrument-specific version of the extended stylesheet.

If the developer wants to change the default settings of the Westat extensions, the IS stylesheet editor can't be used. Instead a text editor or similar tool is needed to access the XSL code and enable or disable specific functions (Figure 10) or change parameters for the functions (Figure 10).

```
<!-- Westat start -->
<!-- OpenSource XSLT flow-control variables - all contributors -->
<xsl:variable name="westat:EnableQuantityUnit">true</xsl:variable>
<xsl:variable name="westat:EnableLineSpacing">true</xsl:variable>
<xsl:variable name="westat:EnableInputLineAlign">true</xsl:variable>
<xsl:variable name="westat:EnableEnumColWidth">true</xsl:variable>
<xsl:variable name="westat:EnableDualInputLine">true</xsl:variable>
<xsl:variable name="westat:EnableTabletStyle">true</xsl:variable>
<xsl:variable name="westat:EnableSpecialEffects">no</xsl:variable>
<xsl:variable name="westat:EnableMenu">true</xsl:variable>
```

Figure 9: Enabling/disabling extension functions

```
<!-- OpenSource Westat -->
<xsl:variable name="westat:DualInputLinePrefix">TDualInputLine</xsl:variable>
<xsl:variable name="westat:QUPrefix">TQuanUnit</xsl:variable>
<xsl:variable name="westat:LSCellSpacing">0</xsl:variable>
<xsl:variable name="westat:LSCellPadding">0</xsl:variable>
<xsl:variable name="westat:FieldLabelWidth">40%</xsl:variable>
<xsl:variable name="westat:FieldNameWidth">40%</xsl:variable>
<xsl:variable name="westat:InputLineWidth">60%</xsl:variable>
<xsl:variable name="westat:TabletRadioButtonWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletCheckBoxWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletButtonWidth">150px</xsl:variable>
<xsl:variable name="westat:TabletButtonHeight">40px</xsl:variable>
<xsl:variable name="westat:HighlightColor">cyan</xsl:variable>
<xsl:variable name="westat:ParallelTabBackgroundColor">yellowgreen</xsl:variable>
<xsl:variable name="westat:ParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:ParallelTabFontFamily">arial</xsl:variable>
<xsl:variable name="westat:CurrParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:CurrParallelTabColor">rgb(120,120,0)</xsl:variable>
<xsl:variable name="westat:CurrParallelTabBackgroundColor">gold</xsl:variable>
<xsl:variable name="westat:MenuButtonBackgroundColor">darkolivegreen</xsl:variable>
<xsl:variable name="westat:MenuButtonColor">yellow</xsl:variable>
<!-- Westat end -->
```

Figure 10: Extension functions parameters

2.5 Technical and organizational challenges of XSL extension

XSL stylesheets in Blaise IS open a significant avenue for extending the already powerful and broad capabilities of the out-of-the-box product. Some key features of this approach include:

- XML and XSL are industry-standard Internet technology being taught and learned widely, and not something restricted to a small niche. Courses, books, and other materials are readily available. So investing, building and applying XSL to web applications in Blaise is feasible at least for a larger organization.
- The XSL language is structured to preclude or minimize the chances of side effects in which one function or extension disrupts standard functions or other extended functions. So, following established development standards, extensions can be added with little risk of corrupting the standard Blaise XSL functions or other third-party extensions.
- Because of the richness of the XML document Blaise generates and passes, the XSL process has access to a large set of elements for its transformations and generation of the page to be sent to the browser. So the range of things that are accessible through the stylesheet is quite broad and substantial.

There's always a catch, of course. In this case it's that the power and potential of XSL for tailoring Blaise IS interviewing pages requires someone with serious technical skills in XML/XSL, and that person will have to become knowledgeable about the Blaise object architecture and its representation in XML.

The web interview stylesheet biHTMLWebPage.xsl for IS 2.0 has 5,335 lines and that in the 4.7 beta has 7,163 lines. Obviously, working in such a complex program requires skill and strong understanding of the details of the XSL and Blaise languages.

Does this mean that for all practical purposes extending IS web interviewing is only realistic for large organizations? This seems likely in most situations. At the same time, based on our work so far it seems that only a limited number of web survey interface elements are needed to make the system able to meet a very high percentage of web interview needs.

While the technical details of XSL programming with Blaise is far beyond the understanding of this writer, the Westat experience so far is that adding functions in the stylesheet is not as difficult and time-consuming a process for the skilled, prepared XSL developer as might be feared. Therefore we might be able to expect that the Blaise team will be willing to evolve its web interview stylesheet to add interface capabilities that users and the team agree are of general use.

Blaise 4.7 includes a powerful example of what can be done with XSL. The specify half of the other-specify is immediately activated and displayed when the "Other" choice is selected.

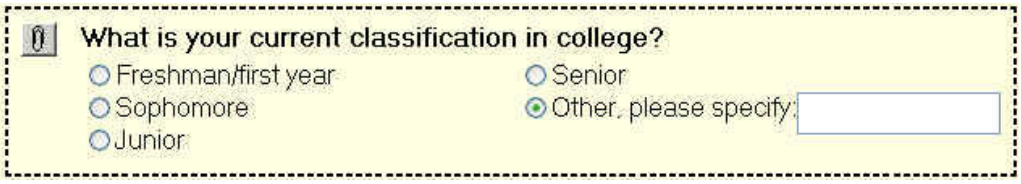


Figure 11: Other-specify implementation in Blaise 4.7

No doubt the Blaise team is as excited as the IS users at the opportunity to continue to expand the usability of IS with such XSL innovations.

2.6 Wish-list of web interview capabilities extension via XSL

Here is a list of user interface extensions of general use that this writer would like to have. They are based on the work of Mick Couper, a leader in research on web interviewing and user interface issues.

Text input areas with labels linked to the text input box.

A currency symbol before or after the field.
$ [_____]

A '%' symbol before or after the field.
[_____]%

Specification: using a prefix of TSymLabel, fields of this prefixed type will get a text label from the field text string "[L|R|A|B][symbol]", e.g.

/"L$" {Place a "$" symbol to the left of the text entry field}
/"R%" {Place a "%" symbol to the right of the text entry field}
/"B(___) ___-_____" {Place the format string "" below the text entry field}

```
TYPE
TSymLabelPay: 1000..100000
 TSymLabelPct: 10..100

FIELDS
Salary "What do you get paid?"/"L$": TSymLabelPay
 Taxes "What percent of your pay goes to taxes?" / "R%": TSymLabelPct
```

Figure 12: Example code for labeled text entry controls

Dates in three fields with separator and labels
__ / __ / ____
mm dd yyyy

Specification: using the prefix TFmtDate, three fields of this prefixed type will be displayed in the order asked with the field text shown below as a label. The default separator symbol would be "/" and could be changed in the stylesheet.

```
TYPE
 TFMTDATEmm:1..12
 TFMTDATEdd:1..31
 TFMTDATEyy:integer[4]

FIELDS
 DOBmm / "mm": TFMTDATEmm
 DOBdd / "dd": TFMTDATEdd
 DOByy / "yyyy": TFMTDATEyy
```

Figure 13: Example code for dates

3. Instrument development methods

The IS web survey instrument development process is a valuable feature of the system. In the Control Centre one codes, prepares, and test datamodels in the same well-integrated Blaise environment. The IS Internet Workshop and Server Manager can be started from the Control Centre tools menu.

Testing a web datamodel is done in two stages. First, the prepared datamodel's static pages can be previewed in the Internet Workshop, enabling rapid checking of the general look and feel of the pages. Testing the active web survey requires more steps—generating the package of specification and files needed for the web server, using Server Manager adding the package to the IIS server, and finally running the survey.

We have found that rapid testing of the structure and rules of a web survey can be enhanced significantly using prepare directives. In the example (Figure 14), when we create the conditional define "webmode" in the Control Centre (Project > Options > Conditional defines > webmode), and prepare it, the NSSE2003 modelib file is used and the LAYOUT section is processed. If we run the datamodel in the Control Centre, its web layout (Figure 15) is not usable for testing.

```
DATAMODEL PrepareDirectiveExample
{$IFDef WebMode}
   {$modelib NSSE2003}
   {$message using modelib NSSE2003}
{$else}
   {$modelib depmode}
   {$message using default modelib}
{$endif}
TYPE
   TGroupOtherClass = (Freshman,Sophomore, Junior, Senior, Other)
   TGroupOtherSchoolType = SET OF (Votec, CommJrColl,Otr4Yr, None, Other)
   TEval = (Excellent,Good,Fair,Poor)

   FIELDS
      CurrClass (1) "@BWhat is your current classification in college?@B" : TGroupOtherClass
      TypeSchl (2) "@BSince high school, which schools have you attended ...B" : TGroupOtherSchoolType
      EvalExp (6) "@BHow would you evaluate your educational experience at this institution?@B": TEval

   RULES
      CurrClass TypeSchl EvalExp

{$IFDef WebMode}
LAYOUT
  FROM CurrClass to EvalExp FIELDPANE LabelList
{$endif}
ENDMODEL
```

Figure 14: Code using prepare directives

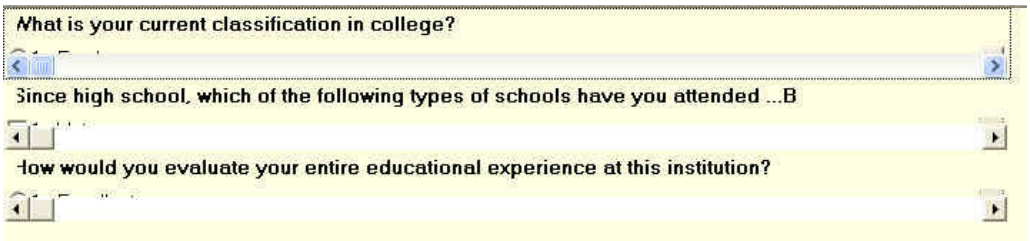


Figure 15: Testing datamodel in webmode

When we change the conditional define to “depmode” (Control Centre (Project > Options > Conditional defines > depmode), and prepare the datamodel, the depmode modelib file is used and the LAYOUT section is skipped, producing a non-web look. When we run the datamodel in the Control Centre, its Windows “dep” layout (Figure 16) is usable for testing.

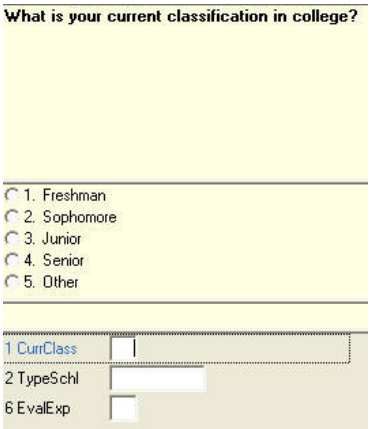


Figure 16: Testing datamodel in depmode

We found prepare directives very useful for rapid development and testing of web surveys. It seems clear that these techniques would also be valuable for multi-mode surveys, allowing the same Blaise source to look and function quite differently in CATI and web.

4. Conclusion

With the key enhancements in Blaise 4.7 now available, we believe IS ready for serious implementations of web surveys. The XSL capability provides a complex and powerful technology for fashioning new or specialized interface elements. The architecture, user interface, and development process all appear suitable to support a wide range of household and establishment surveys.

Exploring the capabilities of Blaise IS

James O'Reilly
Westat

International Blaise Users Conference
Gatineau Quebec
September 2004

While questions remains about the effectiveness of web surveys, interest in web-based surveys has grown among researchers in government, private, and academic organizations. Some clients are pursuing exploratory efforts to learn whether the often discussed challenges--sampling, respondent cooperation, item non-response, self- rather than interviewer-administration, and the vicissitudes of the web environment--can be overcome or at least managed at a level that produces something useful. Others are interested in mixed-mode surveys, usually CATI and web. Still others want to recast existing web surveys: enhancing capabilities or increasing cost effectiveness or expanding the scope.

Westat has integrated the internet into its survey work for a number of years, including conducting web surveys. Recently growing demand and changing web research designs stimulated a company effort to examine our web survey processes and technology. One key goal is to evaluate and, where appropriate, implement web surveys using Blaise IS.

Westat's web surveys have been developed mainly using ASP, Microsoft's scripting environment for generating and processing HTML pages. ASP provides a powerful set of capabilities including easy integration with SQL Server and other relational databases. However, along with its power and flexibility, ASP is a procedurally-oriented, relatively low-level development language. This means custom programming and significant maintenance and support challenges.

Westat's ASP development team has built authoring systems to enhance our capabilities--generating much of the ASP code and reusing standard code modules. Still, it is clear that the ASP approach is problematic for some important types of surveys, for example multi-modal and more complex questionnaire designs.

Of course, all these things fit Blaise IS to a tee, in theory. Blaise is intrinsically multi-modal; it handles complexity naturally; its rules engine provide unmatched routing control; and our Blaise experience in CAPI, CATI, and CASI surveys could be applied to web efforts. But is Blaise IS ready to deliver, ready for prime-time with the architecture, scalability, usability, and extensibility required? That's what our effort was designed to address.

Our major activities were:

- User interface testing and experimentation

- Implementing XSL stylesheet extensions
- Exploring instrument development methods
- Developing a training course
- Designing and planning for server-side management and support systems

We will cover the first three of these here. Most of the work was done using IS 2.0 build 161 of 21 November 2003. As we discovered some of the limitation of IS 2.0, we awaited impatiently the release of the Blaise 4.7 beta so we could test some of the large and small enhancements. Since the beta release on July 30th we have been working intensively on version 2.1 and will report on that also.

Web interview user interface

In evaluating the user interface (UI) capabilities of IS, the core issue is that the user may often have very limited familiarity with computers, web browsers, and how to interact with them, whereas Windows Blaise UI is designed for and used mainly by trained, experienced interviewers. We focused first on elements of relatively simple web surveys of individuals, using the NSSE2003 example. We found IS 2.0 handled well scalar items in dropdown lists, enumerated, string, numeric and open text. Also it implemented nicely grouped enumerated series of questions.

The technique implementing the enumerated grouping was a critical improvement to IS. It revealed how, using naming conventions for field response types in the standard Blaise datamodel, it is possible to instruct the XSL stylesheet to change the way one or more items are rendered in the HTML page sent to the browser. We discuss this in a later section.

We found problems with the usability of complex scalar items such as date and time fields. In Windows Blaise edit masks help the user understand what components of the field to enter. Masks are not allowed in the HTML text entry control. Again extensions to the XSL stylesheet are the solution.

Other UI problems in IS 2.0 included:
- Conditional items displayed on the next page, after a trip to the server
- Limited functionality for multiple language interviews

Both of these are fixed in Blaise 4.7.

Establishment survey prototyping raised other issues, including
- Limited flexibility for controlling the table display
- Difficulty with form-like entry of information with many fields on a page and more than one field on a line desireable.

XSL stylesheet extensions

The architecture of Blaise IS provides great benefits in terms of making the Blaise system's superior features--high-level, structured language using blocks and arrays to implement complex instruments; unmatched rules engine; capable proprietary database; and other elements--available for web interviewing. To do this IS controls the process much more strictly than other web system which typically allow the use of custom JavaScript and other modules to tailor and customize the web application.

The result until recently was that using Blaise IS meant accepting a more limited and dull user interface compared to what one could expect from an ASP or other native web development approach. For some web applications, such as CATI or data editing, with trained staff as users, the tradeoff seems easy to accept. But for many customers the most important part of web survey work is the ability to conduct self-administered surveys—having study subjects do the interview themselves.

For such untrained, voluntary users with widely varying web experience a limited and simple user interface is not likely to be desireable. Expectations of many clients on the look and feel of web applications are for polished, flexible pages.

Fortunately, the IS architecture uses XSLT (Extensible Stylesheet Language for Transformations), usually referred to as XSL. XSL provides a way in IS to extend and refine key functions and interface elements in web surveys. This extensibility potential is different from and in many ways superior to that available in traditional Blaise (DLLs or the BCP).

During a Blaise web survey each time the IS server is ready to communicate with the user's browser, IS uses the API to generate an XML (Extensible Markup Language) document with the appropriate data and metadata on the fields, texts, layout, and other facets of the current situation. The XML document's rich content provides such information as the field's type which allows the stylesheet to process in various ways.

The document is processed by the MSXML parser using the interview XSL stylesheet, biHTMLWebPage.xsl, or a substitute. This process transforms the XML information into the HTML page that is sent to the user.

API > XML > MSXML + XSL > HTML

Figure 1: Blaise IS server steps

As we prototyped surveys in IS 2.0, we identified a number of user interface elements that we needed but could not render with the system as-is. Our engineers studied the system and concluded that we could generate many of the requested elements by adding functions in the interview XSL stylesheet, following the model developed by the Blaise team for displaying a series of enumerated items as rows in a table with the question text in the stub and the enumerated choices as the columns.

Here's an example. In the datamodel TYPE section a TGroupExtent is defined (Figure 2) and applied to the four fields.

```
TYPE
 TGroupExtent = (large "Large extent", Moderate "Moderate extent", Small "Small extent",
   notatall "Not at all")
FIELDS
   BUILD_PREV "Build on previous R&D projects at your company?": TGroupExtent
   ENHANCE_PREV "Enhance the value of previous R&D projects at your company?": TGroupExtent
   BUILD_CURR "Build on other current R&D projects at your company?":TGroupExtent
   ENHANCE_CURR "Enhance the value of other current R&D projects at your company?":TGroupExtent
```

Figure 2: Grouping code

The interview stylesheet has been programmed to check for the answer types of fields, looking for a series of enumerated fields with the same type name and the prefix TGroup. When a set is found the stylesheet instead of displaying the series as individual items in sequence on the page, renders them in a table (Figure 3).

To what extent does your ATP project:

	Large extent	Moderate extent	Small extent	Not at all
Build on previous R&D projects at your company?	○	○	○	○
Enhance the value of previous R&D projects at your company?	○	○	○	○
Build on other current R&D projects at your company?	○	○	○	○
Enhance the value of other current R&D projects at your company?	○	○	○	○

Figure 3: Grouped enumerated display

This model made clear that a wide range of web-standard user interface elements could be added to Blaise IS using only the standard Blaise datamodel syntax and extensions in the XSL interview stylesheet.

We then began to explore the details of implementing this capability as we prototyped instruments and found web interview interface needs that could not be met as fully as we wanted using only the standard Blaise modelib.

Scales

Using the modelib one can display Likert-type scales horizontally but not with the labels displayed above the points rather than to one side. We requested this of the Blaise IS developers which they quickly provided. In the datamodel one uses a TYPE with a prefix of TScale (Figure 4) and a corresponding function in the interview stylesheet to produce the display (Figure 5).

```
TYPE
 TScaleAmbitious = (MuchLess "Much less ambitious", Less " ", SomeLess " ",equal "Equally ambitious",
   SomeMore " ", More " ", MuchMore "Much more ambitious")
FIELDS
   PROJ_AMBITIOUS "Relative to other R&D initiatives in your industry, how ambitious would you
      say are the overall goals identified for this project?":TScaleAmbitious
```

Figure 4: Likert scale extension code

Relative to other R&D initiatives in your industry, how ambitious would you say are the overall goals identified for this project?

Much less ambitious | Equally ambitious | Much more ambitious

Figure 5: Scale display in IS

Quantity-Unit

Quantity-unit questions are important to many surveys. We need to know some quantitative information ("How much are you paid?") and the respondent may think of the concept in terms of so much per hour, per week, per two weeks etc. Using standard Blaise with an interviewer, the two elements are different fields and the interviewer may be able to complete both based on the response received to the first part. Or a follow-on question about the unit may be required.

In self-administration this compound concept can be challenging. So we wanted the visual representation to link the two fields more clearly than a two-step dialog displayed vertically on the page. We specified that two successive fields of types prefixed with TQuanUnit would be displayed side-by-side. Our XSL specialist added this to the stylesheet. The datamodel code and display are below.

```
TYPE
  TQuanUnit_Unit = (thousand, million, billion, noRev "None (No revenues last year)")
  TQuanUnit_Quan = 0..999999
AUXFIELDS
       introrev "What were total company revenues for your parent company last year?":string[1], empty
FIELDS
      RevAmount "Reported amount:"/"Reported amount:": TQuanUnit_Quan
      RevUnit   "Amount reported in:"/"Amount reported in:": TQuanUnit_Unit
RULES
      introrev
      revamount
      revunit
```

Figure 6: Quantity-unit code

What were total company revenues for your parent company last year?

Reported amount:

Amount reported in:
- thousand
- million
- billion
- None (No revenues last year)

Figure 7: Quantity-unit display

Other user interface extensions added using the stylesheet include:

- Enabling two dual input lines to able to display assets and liabilities in two columns.

Please provide the following information from your company's annual financial report for last fiscal year.
We want a balance sheet format with Assets in a column to the left of Liabilities

Balance Sheet

Total Assets		Total Liabilities	
Current assets		Current liabilities	
Long-term assets		Long-term liabilities	
Owners' equity			

- Altering line spacing to show more elements on a page.

This information refers to the organization directly responsible for performir
verify the accuracy of the information listed below.

Company name	
Division or Unit Name	
Street address line 1	
Street address line 2	
Street address line 3	
City	
State	
Zip	

- Adjust the column widths of grouped enumerated sets so that the widths the same.
- Improved control of the width of text
- Increasing the size of the radio button and check box controls to improve ease of use
- Changing the menu and parallel tab look.

Applying custom stylesheet extensions

These extensions are added to the default IS 2.0 web interview stylesheet, biHTMLWebInterview.xsl, following XSL programming standards to prevent our references from conflicting or interacting with other functions in the stylesheet. This is done by defining the Westat namespace (Figure 8) so that all variables related to the extensions are identified by the namespace

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:westat="http//www.westat.com" version="1.0">
```

Figure 8: Westat namespace defined

To use the Westat extensions, developers start with the extended interview stylesheet. Working in the IS workshop and IS stylesheet editor the standard instrument customization is done—changing header and footer appearance, fonts, images, and the like—and saving a instrument-specific version of the extended stylesheet.

If the developer wants to change the default settings of the Westat extensions, the IS stylesheet editor can't be used. Instead a text editor or similar tool is needed to access the XSL code and enable or disable specific functions (Figure 10) or change parameters for the functions (Figure 10).

```
<!-- Westat start -->
<!-- OpenSource XSLT flow-control variables - all contributors -->
<xsl:variable name="westat:EnableQuantityUnit">true</xsl:variable>
<xsl:variable name="westat:EnableLineSpacing">true</xsl:variable>
<xsl:variable name="westat:EnableInputLineAlign">true</xsl:variable>
<xsl:variable name="westat:EnableEnumColWidth">true</xsl:variable>
<xsl:variable name="westat:EnableDualInputLine">true</xsl:variable>
<xsl:variable name="westat:EnableTabletStyle">true</xsl:variable>
<xsl:variable name="westat:EnableSpecialEffects">no</xsl:variable>
<xsl:variable name="westat:EnableMenu">true</xsl:variable>
```

Figure 9: Enabling/disabling extension functions

```
<!-- OpenSource Westat -->
<xsl:variable name="westat:DualInputLinePrefix">TDualInputLine</xsl:variable>
<xsl:variable name="westat:QUPrefix">TQuanUnit</xsl:variable>
<xsl:variable name="westat:LSCellSpacing">0</xsl:variable>
<xsl:variable name="westat:LSCellPadding">0</xsl:variable>
<xsl:variable name="westat:FieldLabelWidth">40%</xsl:variable>
<xsl:variable name="westat:FieldNameWidth">40%</xsl:variable>
<xsl:variable name="westat:InputLineWidth">60%</xsl:variable>
<xsl:variable name="westat:TabletRadioButtonWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletCheckBoxWidthHeight">width:26px; height:26px</xsl:variable>
<xsl:variable name="westat:TabletButtonWidth">150px</xsl:variable>
<xsl:variable name="westat:TabletButtonHeight">40px</xsl:variable>
<xsl:variable name="westat:HighlightColor">cyan</xsl:variable>
<xsl:variable name="westat:ParallelTabBackgroundColor">yellowgreen</xsl:variable>
<xsl:variable name="westat:ParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:ParallelTabFontFamily">arial</xsl:variable>
<xsl:variable name="westat:CurrParallelTabFontWeight">bold</xsl:variable>
<xsl:variable name="westat:CurrParallelTabColor">rgb(120,120,0)</xsl:variable>
<xsl:variable name="westat:CurrParallelTabBackgroundColor">gold</xsl:variable>
<xsl:variable name="westat:MenuButtonBackgroundColor">darkolivegreen</xsl:variable>
<xsl:variable name="westat:MenuButtonColor">yellow</xsl:variable>
<!-- Westat end -->
```

Figure 10: Extension functions parameters

Technical and organizational challenges of XSL extension

XSL stylesheets in Blaise IS open a significant avenue for extending the already powerful and broad capabilities of the out-of-the-box product. Some key features of this approach include:

- XML and XSL are industry-standard Internet technology being taught and learned widely, and not something restricted to a small niche. Courses, books, and other materials are readily available. So investing, building and applying XSL to web applications in Blaise is feasible at least for a larger organization.
- The XSL language is structured to preclude or minimize the chances of side effects in which one function or extension disrupts standard functions or other extended functions. So, following established development standards, extensions can be added with little risk of corrupting the standard Blaise XSL functions or other third-party extensions.
- Because of the richness of the XML document Blaise generates and passes, the XSL process has access to a large set of elements for its transformations and generation of the page to be sent to the browser. So the range of things that are accessible through the stylesheet is quite broad and substantial.

There's always a catch, of course. In this case it's that the power and potential of XSL for tailoring Blaise IS interviewing pages requires someone with serious technical skills in XML/XSL, and that person will have to become knowledgeable about the Blaise object architecture and its representation in XML.

The web interview stylesheet biHTMLWebPage.xsl for IS 2.0 has 5,335 lines and that in the 4.7 beta has 7,163 lines. Obviously, working in such a complex program requires skill and strong understanding of the details of the XSL and Blaise languages.

Does this mean that for all practical purposes extending IS web interviewing is only realistic for large organizations? This seems likely in most situations. At the same time, based on our work so far it seems that only a limited number of web survey interface elements are needed to make the system able to meet a very high percentage of web interview needs.

While the technical details of XSL programming with Blaise is far beyond the understanding of this writer, the Westat experience so far is that adding functions in the stylesheet is not as difficult and time-consuming a process for the skilled, prepared XSL developer as might be feared. Therefore we might be able to expect that the Blaise team will be willing to evolve its web interview stylesheet to add interface capabilities that users and the team agree are of general use.

Blaise 4.7 includes a powerful example of what can be done with XSL. The specify half of the other-specify is immediately activated and displayed when the "Other" choice is selected.

What is your current classification in college?

- Freshman/first year
- Sophomore
- Junior
- Senior
- Other, please specify:

Figure 11: Other-specify implementation in Blaise 4.7

No doubt the Blaise team is as excited as the IS users at the opportunity to continue to expand the usability of IS with such XSL innovations.

Wish-list of web interview capabilities extension via XSL

Here is a list of user interface extensions of general use that this writer would like to have. They are based on the work of Mick Couper, a leader in research on web interviewing and user interface issues.

Text input areas with labels linked to the text input box.
A currency symbol before or after the field.
$ _____

A '%' symbol before or after the field.
_____%

Specification: using a prefix of TSymLabel, fields of this prefixed type will get a text label from the field text string "[L|R|A|B][symbol]", e.g.

/"L$" {Place a "$" symbol to the left of the text entry field}
/"R%" {Place a "%" symbol to the right of the text entry field}
/"B(___) ___-_____" {Place the format string "" below the text entry field}

```
TYPE
TSymLabelPay: 1000..100000
 TSymLabelPct: 10..100

FIELDS
Salary "What do you get paid?"/"L$": TSymLabelPay
 Taxes "What percent of your pay goes to taxes?" / "R%": TSymLabelPct
```

Figure 12: Example code for labeled text entry controls

Dates in three fields with separator and labels
__ / __ / ____
mm dd yyyy

Specification: using the prefix TFmtDate, three fields of this prefixed type will be displayed in the order asked with the field text shown below as a label. The default separator symbol would be "/" and could be changed in the stylesheet.

```
TYPE
 TFMTDATEmm:1..12
```

```
 TFMTDATEdd:1..31
 TFMTDATEyy:integer[4]

FIELDS
 DOBmm / “mm”: TFMTDATEmm
 DOBdd / “dd”: TFMTDATEdd
 DOByy / “yyyy”: TFMTDATEyy
```

Figure 13: Example code for dates

Instrument development methods

The IS web survey instrument development process is a valuable feature of the system. In the Control Centre one codes, prepares, and test datamodels in the same well-integrated Blaise environment. The IS Internet Workshop and Server Manager can be started from the Control Centre tools menu.

Testing a web datamodel is done in two stages. First, the prepared datamodel’s static pages can be previewed in the Internet Workshop, enabling rapid checking of the general look and feel of the pages. Testing the active web survey requires more steps—generating the package of specification and files needed for the web server, using Server Manager adding the package to the IIS server, and finally running the survey.

We have found that rapid testing of the structure and rules of a web survey can be enhanced significantly using prepare directives. In the example (Figure 14), when we create the conditional define “webmode” in the Control Centre (Project > Options > Conditional defines > webmode), and prepare it, the NSSE2003 modelib file is used and the LAYOUT section is processed. If we run the datamodel in the Control Centre, its web layout (Figure 15) is not usable for testing.

```
DATAMODEL PrepareDirectiveExample
{$IFDef WebMode}
   {$modelib NSSE2003}
   {$message using modelib NSSE2003}
{$else}
   {$modelib depmode}
   {$message using default modelib}
{$endif}
TYPE
   TGroupOtherClass = (Freshman,Sophomore, Junior, Senior, Other)
   TGroupOtherSchoolType = SET OF (Votec, CommJrColl,Otr4Yr, None, Other)
   TEval = (Excellent,Good,Fair,Poor)

   FIELDS
      CurrClass (1) "@BWhat is your current classification in college?@B" : TGroupOtherClass
      TypeSchl (2) "@BSince high school, which schools have you attended ...B" : TGroupOtherSchoolType
      EvalExp (6) "@BHow would you evaluate your educational experience at this institution?@B": TEval

   RULES
      CurrClass TypeSchl EvalExp

{$IFDef WebMode}
LAYOUT
 FROM CurrClass to EvalExp FIELDPANE LabelList
{$endif}
ENDMODEL
```

Figure 14: Code using prepare directives

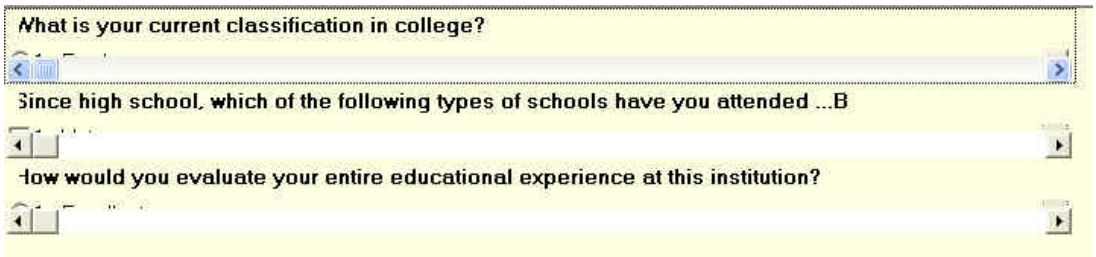


Figure 15: Testing datamodel in webmode

When we change the conditional define to "depmode" (Control Centre (Project > Options > Conditional defines > depmode), and prepare the datamodel, the depmode modelib file is used and the LAYOUT section is skipped, producing a non-web look. When we run the datamodel in the Control Centre, its Windows "dep" layout (Figure 16) is usable for testing.

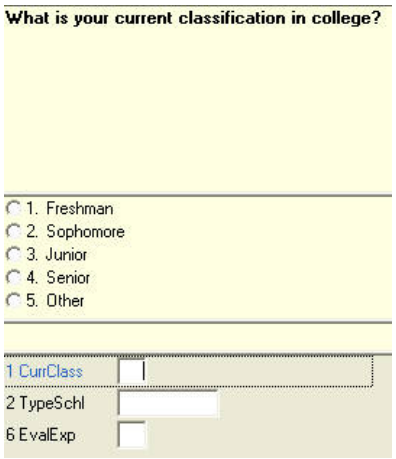


Figure 16: Testing datamodel in depmode

We found prepare directives very useful for rapid development and testing of web surveys. It seems clear that these techniques would also be valuable for multi-mode surveys, allowing the same Blaise source to look and function quite differently in CATI and web.

Conclusion

With the key enhancements in Blaise 4.7 now available, we believe IS ready for serious implementations of web surveys. The XSL capability provides a complex and powerful technology for fashioning new or specialized interface elements. The architecture, user interface, and development process all appear suitable to support a wide range of household and establishment surveys.

Developing the On-line Milk Production Survey using Blaise IS

Author: John O' Connor, Central Statistics Office, Ireland

1. Introduction

The Central Statistics Office of Ireland decided as part of its strategic plan to initiate a project to exploit the potential of data capture over the Internet using Blaise IS.
As this was a first for both the business area and the Blaise development team in terms of using Blaise IS, it was decided that we would proceed with a relatively short form and a sub-sample of the usual respondents. We decided to limit the number of respondents due to the constraints associated with the version of Blaise IS we were using. Even with the issues associated with the version of Blaise IS we felt the experience gained in terms of the development process and feedback from respondents in completing the form would be of great benefit to us in future similar projects and would be worth the initial effort.
The survey selected was the Milk Production survey which was scheduled for live release using Blaise IS in November 2003. Previously this had been a paper-based survey which meant that we would have to develop a new Blaise instrument.

2. Approach to overall management of survey

We decided to develop a Welcome page, which we would certify as a secure site and create hypertext links within this page to the on-line Blaise instrument. This allowed us to add new hypertext links for other surveys.
This Welcome page contains some background information as well as advice and technical assistance on completing the Blaise instrument(s).
We opted for the interview-based approach over the form-based approach in this case primarily because of the level of routing involved.
We also decided to use the same Blaise instrument for a number of reference periods of the survey thus avoiding development overheads associated with each new reference period. All potential respondents were informed by post of the new data collection method, the URL to log on to and their own unique PIN.

3. Development process

We needed to investigate Blaise IS in terms of translating a standard Blaise instrument into an application that somebody who is not familiar with the concept of routing, error handling, etc would be comfortable with. Up to now all Blaise interviews had been carried out by trained CSO interviewers. This new initiative would involve people who would not be familiar with the concept of automatic routing i.e. questions appearing and disappearing depending on the responses given or standard Blaise errors.
Using the sample Fears instruments and the various Blaise IS options we created several different instruments. We also began combining changes in the modelib and Dep menu Manager with the samples provided.
Initially we created nine separate prototypes in total, which we then presented to the business area. We started with the defaults, changing colour, text screen layout adding the CSO logo and contact information.
Once we had agreed on a general approach we began to develop a draft version of the live instrument.
As the live instrument involved a good deal of routing, we needed to ensure that the instrument was as user friendly as possible.
Certain key areas were identified which we felt would determine if the users would find the instrument easy to use.

These areas were –

3.1. Layout

- Very early on we abandoned the traditional Blaise screen (2 parts) and opted for a paper-based look. We felt that the idea of having two parts to the screen would not suit people who would not have seen this style before.
- As each question response involves the server validating the answer, the instrument could prove time consuming to complete, even for a short return.We therefore needed to create a instrument that was capable of being completed quickly in the event of a "nil" return or where the amount of information being returned involved only one section of the instrument.
- We were conscious that the types of hardware and software used by respondents would vary significantly. Because of this we needed to ensure that question text, font size, colour and insertion of page breaks were designed so that the instrument would be readable across all types of set-ups.

3.2. Navigation

- Navigating backwards proved to be problematic due to automatic page breaks which required a different approach in order to bypass.
- We also needed to decide whether to use scroll bars (effective within a page) or not.
- We felt the inclusion of a progress bar on the top of the instrument would be helpful for the respondent however we needed to ensure that it was constantly visible and that it did not scroll up.
- The instrument was eventually designed as one long page with the option of using either the mouse and scroll bar or the next and previous buttons to navigate forwards or backwards.
- We decided not to include any menu options.

3.3. Validation

- We needed to ensure that any validation and associated messages were intuitive.
- We decided against using hard errors as we felt that the option presented could lead to confusion (goto, suppress).
- The general consensus was to keep the instrument as simple as possible with the minimum number of options available but still ensuring an adequate level of validation and routing.
- The approach we took to ensuring that the respondents' details were correct in terms of their ID and reference period was to use a pre filled database that we included in the package. We created a partially completed database that contained the PIN and reference periods for each respondent for a period covering 3 years.This information represented the key for each record and comprised of the respondent's unique ID and the reference period they were responding to. If the record does not exist on the database the respondent is not allowed to carry on. We selected the option not to allow the creation of new records on the database.
- If the respondent attempts to complete a previously completed return, they are flagged that a return has already been made for the period concerned and asks them to confirm that they wish to continue. In the event a respondent selects a reference period in the future, a message is displayed informing the respondent of the error. This check utilises the current system date.

Every time a record is submitted for a reference period, a counter is incremented to record the number of times a return is completed by a respondent for that reference period. This identifies the return as an amendment to an already completed return in the business area.
We can interrogate the database to find out how many returns have been made for a particular reference period and if required, identify the amendments that have been submitted for previously submitted returns.

A summary question was included at the end of the instrument that reminds the user of the responses they are about to submit and provides a final opportunity to make any amendment if necessary.

4. Using Blaise IS

The Workshop, Package Builder, Server Manager are quite easy to use and are intuitive.
We found that by experimenting with various layouts, it was easy to become familiar with the options available.

4.1. Unforeseen issues

Initially, we used the same virtual directory structure for different instruments. This in itself did not cause a problem. We think a problem occurred when we attempted to stop and remove one of the instruments from the survey manager. As it shared the same virtual directory as other instruments it caused the Server Manager to fall over. At one stage we needed to uninstall Blaise IS and IIS and re-install them both before the problem was rectified.
As the version of Blaise IS being used did not facilitate concurrent access, we needed to ensure that no more than one respondent was connected at any one time We did this by developing a VBScript program which first checked to see if a .~lk file existed before allowing a user to connect to the interactive form. If the file existed the respondent was advised to try logging on later. Although a regressive step in terms of completing forms on line, we felt that it would not be a major issue as the number of respondents was small and the number of instances where somebody would be locked out would not be that frequent.
We did not initially envisage having to amend the receipt and error pages. Although the changes were small, they still required some understanding of HTML.

4.2. Amendments to default settings

A number of changes were required in the modelib to create the page layout we were looking for, a continuous page with a constant header and footer. These included the following amendments to the default grid of the layout interviewing section.
Default Grid
View height : 26
Cell width : 76 Cell height : 12
Page width : 1 Page height : 38
Default Field Pane
Field text set to visible
Answer list set to visible
Input line set to visible
A continuous page was considered the only viable option as page breaks made navigation, particularly backwards very difficult.
A major disadvantage of the Blaise development environment was that the changes to the modelib settings had different effects on the normal instrument and how it looked in the BlasieIS preview. This proved time consuming as the instrument had to be compiled after changes made to the modelib and then the instrument had to be brought into the BlaiseIS Workshop before the layout amendments could be checked using the preview option.
We used a variety of fonts, colours and sizes for maximum impact. The default font was changed to Arial, size 9. We used a bold font for the question proper and the default font for range explanation. We used a brown italic font for user instructions for example, how to move to the next question and a blue underline font for section headings.
We inserted the CSO logo and contact information logo on the top of the HTML interactive page/form and we included a progress bar. A permanent footer was also included with two navigation buttons that allowed the user move to the “Previous” or “Next” question. This involved some HTML knowledge.

We also made some changes to the receipt page to ensure the close button is always visible by moving it to the top of the page. Some text changes were made to the Blaise IS error page to make it more "diplomatic".
In order to minimise the amount of time a respondent could be locked out we amended the timeout option to 7 minutes with a warning after 5 minutes.

5. Security

Security was addressed by creating a certified secure site on a server outside the CSO firewall. Although the site is secure we still selected the SSL option when generating the Blaise IS instrument.
Two further measures were adopted. The first was to remove any identifying questions. The second was to remove the data from the database and transfer it inside the firewall as quickly as possible once the interview was complete.
This was done by using the on load event of the receipt page which only happens when a return has been submitted. A VB exe was written for this purpose. This guaranteed that the Blaise database never contained data for longer than a few seconds post completion of a return.

6. Testing

Testing of hardware and software was a significant issue. We needed to test as varied set-ups as possible to ensure the instrument layout was compatible with most types of hardware and software. As the CSO has a standard approach to software and hardware the normal testing associated with application development would not suffice in terms of assuring the suitability of the application across different platforms.
This issue was resolved by giving colleagues their own PINs, getting them to submit returns from their home PCs and record when and what they submitted.
Feedback was received on response times, user interface and ease of use.

7. Sequence of events for submitting a return

- Log on to secure site Eforms.cso.ie using name and password. The Welcome page provides some technical information
- Click on the hypertext link to the survey.
- The respondent proceeds to complete the form.
- When the form is submitted successfully i.e. without any hard errors and the record is written to the Blaise Database, a receipt page is generated by the Blaise IS system that is displayed to the respondent. The on load event of this receipt page initiates a VB application which carries out the following tasks :
  - Runs a manipula that extracts the record from the Blaise database into a fixed length text file and subsequently deletes the contents of the record except for some selected variables.
  - Generates an SMTP message which emails the text file as an attachment to an account inside the firewall.
  - Deletes the text file.

8. Conclusion

In hindsight the work involved in developing a suitable instrument layout and the work required in addressing security issues, involved far greater effort than originally envisaged.
Although, having completed the first survey, any subsequent surveys should prove much easier to implement.
The multi-user access issue needs to be resolved before Blaise IS can be used widely as a data capture tool.

The response time is a potential problem as it is dependent to a large extent on telephony hardware available to the respondent. A major concern in the testing was the amount of time involved in accessing the server to validate the answer and prompt the next question. As all the testers at one point were testing from home without the benefit of ISDN or broadband the response times were very slow. As this was a survey aimed at businesses that all have ISDN lines the problem is not as relevant to them. It did however highlight the fact that there are definite limitations as to what this mode of interviewing can be used for.

Programming Guidelines for Good Data Documentation

Sue Ellen Hansen, University of Michigan

Abstract

Prior International Blaise Users Conference (IBUC) meetings have had sessions and papers on screen design standards (e.g., Couper 2000, Gatward 2003, Hansen and Dinkelmann 2003, Kuusela 2003, and Wensing 2003). Past meetings have also had papers on programming approaches and standards (e.g., Soper and Dyer 2001 and Altvater, Stanford & Ziesing 2001). Generally, however, programming and interface standards have been defined separately from attempts to design data structures and to produce data documentation. Recent efforts at the University of Michigan and elsewhere suggest that it is time to revisit Blaise programming standards, with a view toward providing good (complete and readable) data documentation.

The University of Michigan (Michigan) has created a tool for producing XML-based codebooks and questionnaires (Sparks and Youhong 2004), which may be printed or viewed as web pages. In the course of the development, the system was tested on several complex instruments. The results varied in the "usability" of the documentation produced.

This paper describes the problems encountered, identifies specific programming styles or approaches that hindered adequate documentation, and proposes programming guidelines for future instruments that would ensure well-documented Blaise instruments and datasets.

1.0 Introduction

Couper (2000) suggests that there are three aspects to computer assisted interview (CAI) design that need to be balanced. These may be viewed perhaps as three legs of a development stool: (1) programming, (2) user interface, and (3) data output and documentation. An emphasis on one aspect can lead to neglect of at least one of the other aspects, and thus create an imbalance. The requirements of one aspect also may be in conflict with another. For example, it may take twice as long to program a Blaise instrument for good user interface than for poor user interface. Finally, those who are most knowledgeable about one aspect of design may be ill informed about other aspects, and may be unable to achieve a good balance when addressing programming, interface design, and data output and documentation concerns.

Many organizations have developed programming and screen design guidelines, but few have addressed the impact of programming and design decisions on data output and documentation. Michigan developed a preliminary set of CAI instrument specification guidelines, which to date have been used for programming a handful of Blaise instruments, and have resulted in reduced programming costs. The recent development of an automated Blaise Documentation System (see Sparks and Youhong 2004) has forced us to revisit these specifications, as well as programming and screen design guidelines, and to modify them so that instrument specifications lead to better documented survey datasets. The goal is to have three sets of guidelines for development of CAI instruments, all of which reflect a balance among programming, screen design, and data output and documentation:

- CAI Instrument Specification Guidelines
- CAI Screen Design Guidelines
- CAI Programming Guidelines

The remainder of this paper (1) describes the components of basic data documentation; (2) discusses aspects of instrument specification, screen design, and programming that need to be considered to generate good data documentation; and (3) and concludes with a summary of preliminary specification and programming guidelines.

2.0 Basic Data Documentation

Figures 1 and 2 show Blaise variables as documented by the Blaise Documentation System, the first as part of questionnaire documentation, the latter for a codebook with frequencies[1].

Figure 1. Basic Questionnaire Documentation

CNO_1

The next questions are about children. [Have you ever given birth to a child / Have you ever had a child where you were the birth father]?

- 1 YES
- 5 NO
- 8 DON'T KNOW
- 9 REFUSED

Universe

BLN_HHL.HU9>0

BLSCREENING.SC19=C01

The key elements for questionnaire documentation are:

- Question identifier
- Question
- Response options
- Universe

For codebook documentation, the additional elements are:

- Frequencies and Percents
- Variable characteristics (in this case for SAS)
    - Dataset position
    - Blaise variable type
    - SAS variable type
    - Number of decimals
    - Minimum value
    - Maximum value
    - Missing data codes
    - Whether an empty value is allowed (Blaise attribute EMPTY)

[1] These examples represent documentation generated from programming that deviates in some respects from Michigan guidelines.

Figure 2. Basic Codebook Documentation

TB4

Was there ever a period in your life lasting at least two months when you smoked at least once per week?

- 1 YES
- 5 NO
- 8 DON'T KNOW
- 9 REFUSED

Universe

TB2=C01

BLTOBACCO.TB4

Value Label	Frequency	Percent	Cumulative Frequency	Cumulative Percent
. MISSING	970			
D DON'T KNOW	1			
R REFUSED	1			
1 YES	2835	69.62%	2835	69.62%
5 NO	1237	30.38%	4072	100.00%

- **Position:** 2866
- **Blaise Type:** Enumerated
- **SAS Type:** Numeric
- **Decimals:** 0
- **Minimum:** 1
- **Maximum:** 5
- **Missing Data Codes:** ., .D, .R
- **Empty:** N

3.0 Instrument Specification

Pierzchala and Farrant (2000) make a strong case for providing non-programmers with enough of an understanding of Blaise to provide complete specifications for instruments. What is missed in the specifications will be left up to the programmer to decide on how to implement, with a probable impact (not necessarily positive) on the Blaise interface and data output and documentation.

In the CAI instrument specification guidelines under development at Michigan, designers are asked to provide detailed specifications about each question or variable (Blaise field) they expect to have in the final dataset. Each variable described in the specification for an instrument should have following information:

- Field Name,
- Question text / Interviewer checkpoint text,
- Field description,
- Skip [Go to] instructions, and
- Answer Categories/Data Type
  or
- Valid numeric range

Other components of a field would be specified as necessary:

- Field Tag
- Logic for Fills
- Interviewer instruction text, including text of probes
- Entry masks
- Soft Consistency Check (SIGNAL)
- Hard Consistency Check (CHECK)
- Field level DK, RF, EMPTY attribute

Figure 3 gives an example of basic specifications for a variable, "Working Now." In this example, the field name is "WorkNow," the descriptor "Working Now," and the tag "F6."

Figure 3. Basic Specifications for the Variable "Working Now"

WorkNow / Working Now
F6

We would like to know about what you do -- are you working now, temporarily laid off, unemployed, retired, permanently disabled, a homemaker, a student, or what?

- ENTER all that apply
- ENTER [Space] or [-] to separate responses

Working	1	Working	
LaidOff	2	Temporarily laid off	
Unemployed	3	Unemployed	
Retired	4	Retired	
Disabled	5	Permanently disabled	
Homemaker	6	Homemaker	
Student	7	Student	
Other	8	Other -- specify	Go to F6_Other
DK	*98*		
Ref	*99*		

The aspect of this specification relevant to data documentation is variable identification. Michigan recommends mixed case and meaningful field names and descriptors, and that all three identifiers including the tag are specified. This allows the use of any or all identifiers in the documentation, and for more flexibility in screen design (use of the tag in screen design is discussed in the next section)[2].

3.1 Preload variables, checkpoints, and constructed variables

Specifications should contain information on all variables expected to be in the final survey dataset. These include data that are to be preloaded into the Blaise data record at the beginning of the interview, explicit interviewer checkpoints, implicit internal checkpoints, and any constructed variables that analysts want in the final dataset (including randomly generated numbers). If care is taken to put such variable information in the original instrument specifications, programmers will know to declare such variables as fields and not auxiliary fields. This ensures that they are exported with the survey data and that they are appropriately documented.

Figure 4 provides an example of an explicit interviewer checkpoint specification. In this case question text is in blue indicating it is a question to be answered by the interviewer, with nothing read to the respondent. Figure 5 shows the associated Blaise screen.

Figure 4. Specification for Interviewer Checkpoint "Mode Checkpoint"

IwerChk1 / Mode Checkpoint
Check_1

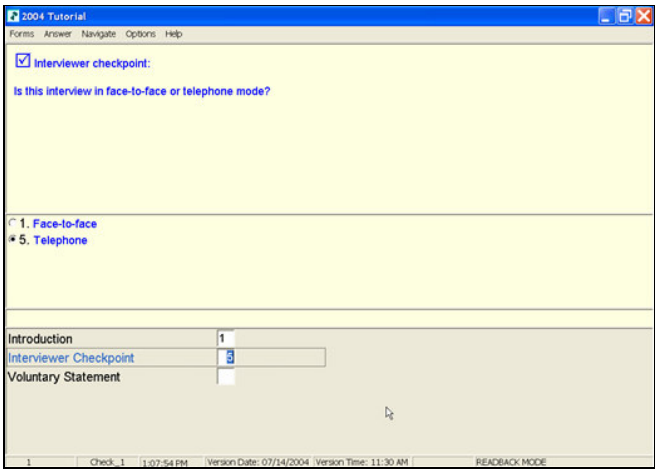
Interviewer checkpoint:

Is this interview in face-to-face or telephone mode?

1	Face-to-face
5	Telephone

[2] As indicated, Michigan is revising all CAI instrument development guidelines. Thus, examples do not necessarily reflect final guidelines. However, basic naming conventions and the use of the three variable identifiers will remain.

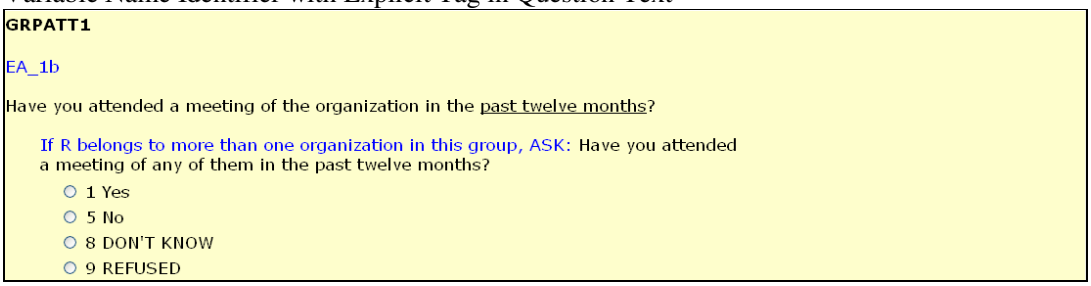
Figure 5. Blaise Screen for Interviewer Checkpoint "Mode Checkpoint"



4.0 Screen Design

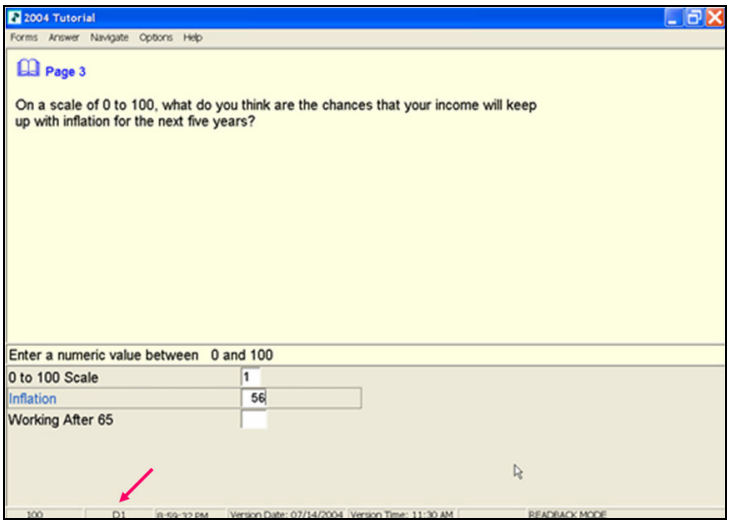
The primary programming issue related to screen design that has affected data documentation has been whether or not to include a question identifier in the form pane or question text area of the Blaise screen. While Blaise allows display of the tag in the form pane, programmers have no control over its format (e.g., color) and placement in relation to the question text. Thus, programmers are often asked to include a question identifier in the form pane. However, for data documentation systems such as Michigan's that automatically places an identifier above the question text, the result is duplicate identifiers in the codebook, as shown in Figure 6.

Figure 6. Codebook Example:
Variable Name Identifier with Explicit Tag in Question Text



While a variable descriptor or name is the preferred identifier in the form pane or input area of a Blaise screen (Figure 5), the tag is the preferred screen identifier. This is because it is short and easy to refer to when troubleshooting problems and when training interviewers to use an instrument. In our example it is much easier to refer to screen "EA_1b" than to screen "GRPATT1." Michigan's solution has been to suggest that the tag not be put in the form pane or question area, but instead be placed in the status bar at the bottom of the screen, as shown in Figure 7.

Figure 7. Example of Screen with Tag in Status Bar Used for Identification

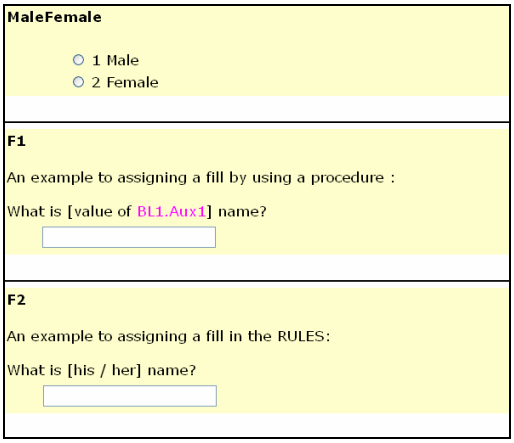


In a more general sense, screen design has a tremendous impact on the readability of any documentation created from Blaise instruments. Screen design standards affect both the instrument interface and the data documentation, since both take question text and response options from the Blaise datamodel. If question text (including all elements displayed in the info pane) and response options are poorly formatted for usability and interface design, then they will be equally poorly formatted for data documentation.

5.0 Programming

Sparks and Youhong (2004) discuss two aspects of Blaise programming that affected data documentation using Michigan's Blaise Documentation System, and they are both related to display of fill text. The first is that if an instrument relies on en external database lookup for creation of fills, they cannot be displayed in the data documentation. Figure 8 shows how a fill would display if created through a call to a procedure capturing the data from an external database (F1), and how it would appear if created directly in the instrument RULES section (F2).

Figure 8. Example of Problematic Fill Display When Use Procedure to Create Fill



The preferred display is as it appears for F2. One option for achieving this would be to avoid the use of fill procedure calls in Blaise instruments. The other would be to add to the Blaise Documentation System the option to capture fills from an external database. This may be done in a future version.

The second problem related to fills arises when a fill variable name is used multiple times in the same block, something which had been done in several Michigan instruments. All possible values associated with the variable will display in the data documentation, as shown in Figure 9 (F2). Figure 10 shows the mock code that generated the documentation.

Figure 9. Example of Problematic Fill Display When Reuse Fill Variable Name in Block

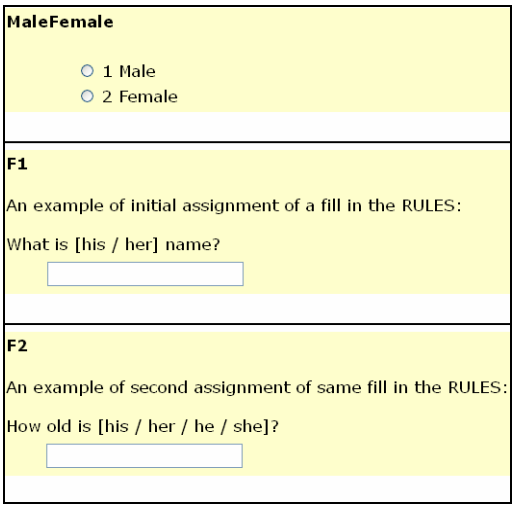


Figure 10. Blaise Code for Generating Reused Fill Example

```
DATAMODEL    TESTFill
  AUXFIELDS
    Aux1 : STRING
  BLOCK B1
     FIELDS
       MaleFemale :(Male, Female)
       F1 "An example of initial assignment of a fill in the RULES:
           @/@/What is ^Aux1 name?" : STRING [1]
       F2 "An example of second assignment of same fill in the RULES:
           @/@/How old is ^Aux1?" : STRING [1]
     RULES
       MaleFemale
      IF MaleFemale = Male THEN
          Aux1 := 'his'
      ELSE
          Aux1 := 'her'
      ENDIF
      F1
      IF MaleFemale = Male THEN
          Aux1 := 'he'
      ELSE
          Aux1 := 'she'
      ENDIF
      F2
   ENDBLOCK
   FIELDS
    BL1 : B1
  RULES
    BL1
ENDMODEL
```

6.0 Conclusions and Recommendations

Good data documentation must be easy to read and must be complete. The issues raised in this paper are primarily focused on those two aspects of data documentation. Good screen design can lead to readable data documentation, as can good specification and programming standards. Based on what Michigan has learned on the Blaise Documentation System development initiative, we would make the following recommendations:

- Ask clients and project managers to fully specify instruments, most importantly by providing meaningful field names and descriptors, as well as tags that can be used as short references to fields
- Do not put anything in an info pane that you would not want to appear in data documentation (such as an identifier above the question)
- Have fully elaborated screen design guidelines that are reflected in detailed instrument specification guidelines
- Do not use procedure calls to create fills from external data (unless you have a documentation system that can work with the external data
- Do not reuse fill variable names within the same block

The experience has also convinced us that all of Michigan's guidelines (specification, screen design, and programming) need to be updated, expanded, and cross-referenced, so that regardless of the backgrounds of the users these documents they will lead to efficient programming, usable instruments, and good documentation.

Contact: sehansen@umich.edu

7.0 References

Altvater, David, Ventrese Stanford, and Curtis Ziesing. 2001. Programming Techniques for Complex Surveys in Blaise. Paper presented at the 7th International Blaise Users Conference, Washington, September.

Couper, Mick P. 2000. Development and evaluation of screen design standards for Blaise for Windows. Paper presented at the 6th International Blaise Users Conference, Kinsale, May.

Gatward, Rebecca. 2003. Developing and updating screen layout and design standards. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.

Hansen, Sue Ellen, and Karll Dinkelmann. 2003. Screen design guidelines for Blaise instruments. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.

Pierzchala, Mark, and Graham Farrant. 2000. Helping non-programmers to specify a Blaise instrument. Paper presented at the 6th International Blaise Users Conference, Kinsale, May.

Soper, Ellen, and Ed Dyer. 2001. Challenges of Instrument Development. Paper presented at the 7th International Blaise Users Conference, Washington, September.

Sparks, Peter, and Youhong Liu. 2004. Blaise Documentation System. Paper presented at the 9th International Blaise Users Conference, Ottawa, September.

Wensing, Fred, Jane Barresi, and David Finlay. 2003. Developing an optional screen layout for CAI. Paper presented at the 8th International Blaise Users Conference, Copenhagen, May.

Blaise Documentation System

Peter Sparks and Youhong Liu, University of Michigan

1. Introduction

The University of Michigan (UM) has created a tool (BlaiseDoc) for producing XML-based codebooks and questionnaires, which may be printed or viewed as web pages. In the course of the development, the system was tested on several complex instruments. This paper describes the design of the system and the methods used in producing the automated documents. There are different levels of information stored in the XML document, including "go to" logic, fills, external lookup references, multiple languages, and help references. The XML can then be output using different stylesheets for different target documents. Document examples are provided.

This paper also describes problems encountered during development and future enhancements under consideration.

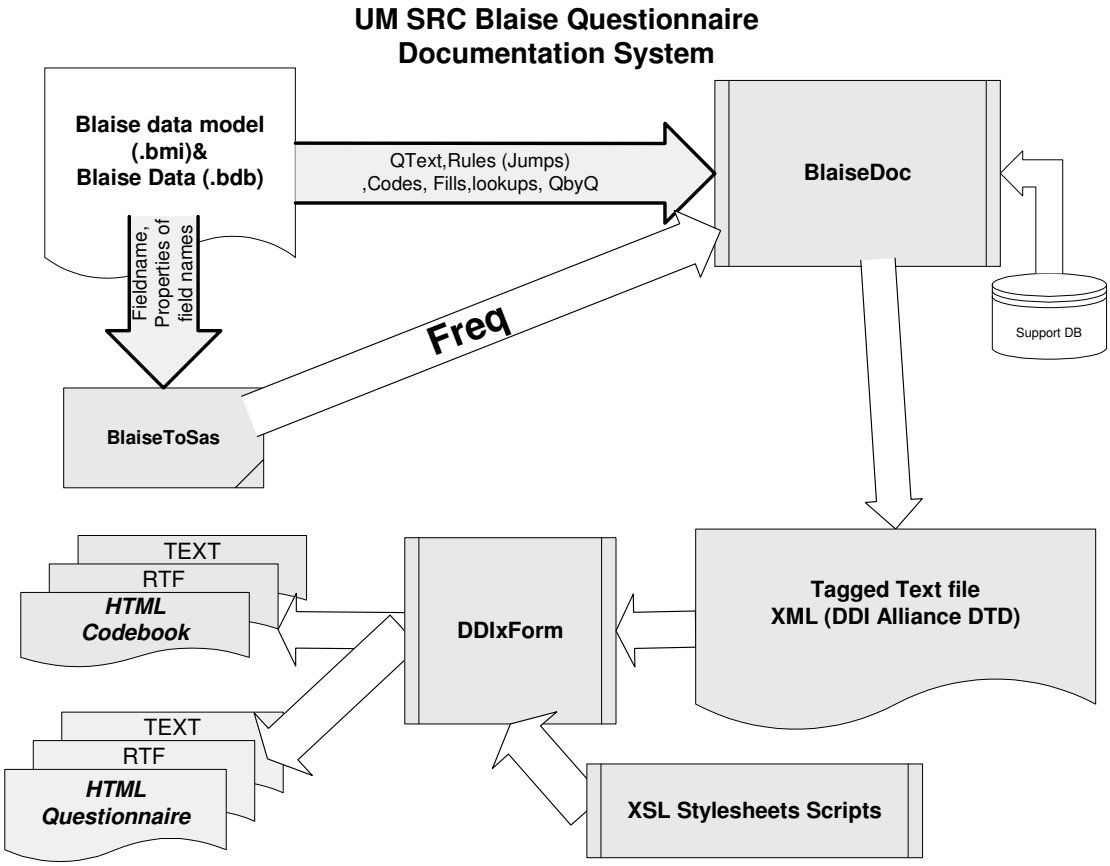
BlaiseDoc is a powerful tool to help create questionnaires, codebooks, and other documentation based upon the Blaise datamodel for a study. It works by analyzing the datamodel and its associated files, exporting that information to an XML file, and then displaying only what's needed via XSL (stylesheets) for the user.

2. Overall flowchart of system

BlaiseDoc has been designed to work as part of a larger system, parts of which are still being implemented. The current program reads a Blaise datamodel (and its associated external lookup files) and generates one XML file per language selected. It then transforms the XML data via a selected stylesheet to generate the output. This can be tailored to individual user needs by designing unique stylesheets that meet those needs.

The system has been designed to also work with SAS© data to read frequencies that will then be merged with the XML output. The frequencies had been done with Cameleon scripts, Blaise Manipula scripts, and a fair amount of hand processing. A new tool, BlaiseToSas, has been created to automate the export of Blaise data into a format more compatible SRC standards.

Other options include merging interviewer online help documents and respondent booklet information at the appropriate points in the generated document from a support database.



3. Methods used in producing automated documents

Creating a final document from a datamodel takes several steps and a great deal of processing. Visual Basic with the Blaise Component Pack was used for the main program. Additional modules were written (also in VB) so that a support database and SAS© could be included in the output. VB.net was used to implement the transformation of the DDI XML file into the different output files.

3.1 Processing the datamodel

BlaiseDoc begins by processing the entire datamodel per each language chosen. The output is dependent upon each language because there may be more or fewer questions, different fills, different frequencies and so forth for each. The program processes the datamodel by stepping through every rule through the DictionaryAsStatements method. It was found to be much faster than using the RulesNavigator for the same information and less memory intensive.

Each appropriate statement is added to a global structure in memory that annotates the datamodel. The different types of statements stored include

Questions:

ASK if presented at some point they are stored as a normal question

assigned if the question is not ASKed but is assigned a value they are shown as constructed.

SHOW if not ASKed or assigned they are presented as a preload question

KEEP if not ASKed or assigned they are presented as a preload question

Consistency checks:

CHECK is presented as a hard consistency check

SIGNAL is presented as a consistency check

Conditions:

IF if a destination for a goto is present within the IF logic, and the IF logic itself cannot be expressed simply, then an internal checkpoint is created. "Simply" means an enumerated condition that can be resolved, such as A54 = Code1.

The execution of BlaiseDoc is essentially linear.

Action	**Description**
Collect user inputs to the program	Destination directory Datamodel to use External files Languages to process Stylesheet to use Support database (if used) SAS© data (if used) Blocks to process
Set up the directory structure	Needed for the output
Process external databases	Read in and write the data from the external databases to a separate XML file.
Step through datamodel in RULES order.	Add questions, consistency checks, internal checkpoints to a global structure in memory.
Examine each enumerated question and internal checkpoint	Resolve the destination goto for each value, DK, and RF.
Look for fill references in text (questions, codes, consistency checks)	Resolve each fill use
Create the XML file(s)	Write the information in the global structure and datamodel to an XML file per language selected. Place help and RB references contained in the support database into the question text. Merge frequency data from SAS© (if used).
Apply stylesheet	Current stylesheets produce HTML, RTF, and TXT output.

3.2 Notes on Resolving Goto references

The determination the destination of an enumerated or DK/RF value from a question is difficult. It requires a scan proceeding from the next statement after the question/internal checkpoint until an appropriate "goto destination" is found. The destinations can either be another checkpoint, another question that is asked, or the end of the Blaise datamodel.

Each value/DK/RF for the question/internal checkpoint needs to be examined since it's possible to have different destinations for each. Each goto destination needs to be examined to see if it is an appropriate destination according to the logic and language. For example,

```
RULES
(1)     A1
(2)     IF ACTIVELANGUAGE = SPA THEN
(3)             IF A1 = Yes THEN
(4)                     A2S
(5)             ENDIF
(6)     ENDIF
(7)     A3
```

The questions on lines (1), (4), and (7) are added to the global structure as questions. The conditions on (2) and (3) are not added as internal checkpoints because both can be resolved simply. Line (4) is the first "goto destination" from A1, but it is not valid within the context of default language. Hence all goto's from A1 are to A3, line (7), for the default language.

If the language was SPA, then A2S is the first valid goto destination but only for A1 = Yes. All other values of A1 are to A3. The output from the above program is shown below.

3.2.1 English Sample Output

Testing

A1

Do you like ice cream?

- 1 Yes
- 2 No
- 8 DON'T KNOW
- 9 REFUSED

View this question in...

A3

Thanks, that's the only question I have.

- 1 Continue

View this question in...

3.2.2 Spanish Sample Output

The Spanish that is presented below was translated via AltaVista (http://babelfish.altavista.com/babelfish/tr) and is only intended for demonstration purposes.

Testing

A1

¿Usted tienen gusto del helado?

- 1 Sí
- 2 No **GO TO A3**
- 8 DON'T KNOW **GO TO A3**
- 9 REFUSED **GO TO A3**

View this question in...

A2S

¿Qué clase de helado?

View this question in...

A3

Las gracias, ésas son las únicas preguntas que tengo.

- 1 Continue

View this question in...

3.3 Notes on Resolving fill references

Fills are resolved by a complex algorithm that works from the rules in the same block as the question. If assignments to the fill variable are not found in the context, Blaise doc backs up a level and searches the rules of each prior block until the fill is resolved. It will do forward searches to subblocks, and matches internal block parameters to parent parameters from the calling block. It will be unable to resolve a fill if the fill is assigned within a procedure or is a result of a external search. In this case the fill variable name is displayed.

3.4 Overview of XML content

XML (e**X**tended **M**arkup **L**anguage) was chosen as an intermediate output of the BlaiseDoc program. It gives a great deal of flexibility when exporting the information from processing into different formats based upon user preference. The initial XML output for BlaiseDoc was custom made for the program. This was changed to take advantage of a standard being supported by the DDI Alliance whose members include a number of organizations across North America and Europe (http://webapp.icpsr.umich.edu/cocoon/DDI-COMMITTEE/expert.xml).

3.5 DDI

The Data Documentation Initiative (DDI) is an effort to provide a standard for the documentation of datasets in the social and behavioral sciences internationally. The standard has an XML base that enables efficient use of the datasets.

The structure of the DDI is stored in a Document Type Definition (DTD) which provides a information about the elements, attributes, comment, notes, and entities that are used in the DDI.

More information about the DDI can be found at http://www.icpsr.umich.edu/DDI/index.html.

You can also download a copy of the DTD from
http://www.icpsr.umich.edu/DDI/users/dtd/index.html#version2.0.

Information on the DDI Alliance and the DTD were found on ICPSR's website. ICPSR is the Inter-University Consortium for Political and Social Research at the University of Michigan.

3.6 Stylesheets

The final step of the process involves running stylesheets against the exported XML data. This process is straightforward and makes use of VB.Net to provide the transformation of the XML data into RTF and HTML outputs. This transformation is performed by DDIxForm, a VB.net program that is called from BlaiseDoc (a VB6 program). There are plans to move the program to just one platform and so DDIxForm will be incorporated into the BlaiseDoc program.

4. Installing BlaiseDoc

BlaiseDoc requires a series of installations in order to work. They include

BlaiseDoc
Blaise BCP 2.0
DDIxForm
MS_XML_4.0
SAS© (if working with SAS© data)

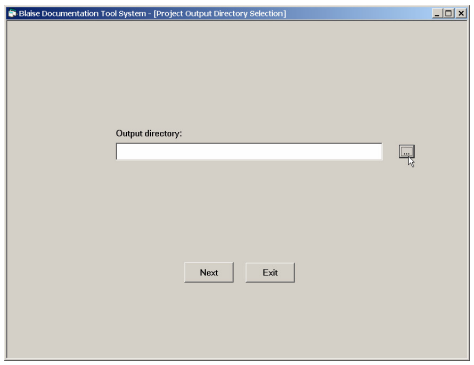
It is highly recommended that the current program is installed on Windows XP (or later) machines only. As part of the packaging process certain system DLLs are incorporated as part of the setup. When the install routine is run on an older Windows operating system it may attempt to update key system files with those from XP. These are incompatible with the other system files on the older system.

If working with SAS© an additional ODBC connection needs to be established to the SAS© library. The program will look for this connection in order to read frequency data from a SAS© dataset.

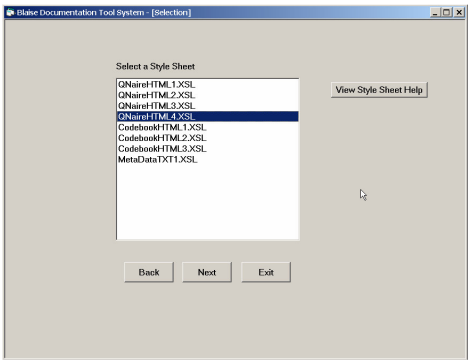
5. Running the program

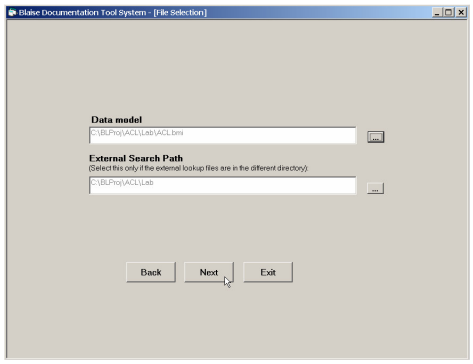
5.1 Questionnaire method

Once the program is installed, run BlaiseDoc via the normal methods. It will display a splash screen, followed by the first screen of the project wizard.

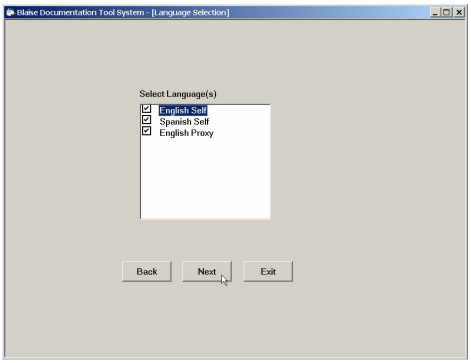


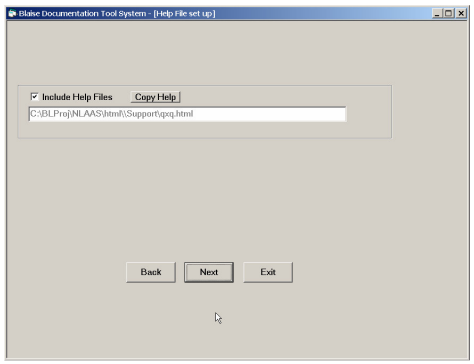
The Output directory is the location where the generated files (XML and HTML/RTF/TXT) will be stored. The datamodel and its external Blaise databases are next chosen.





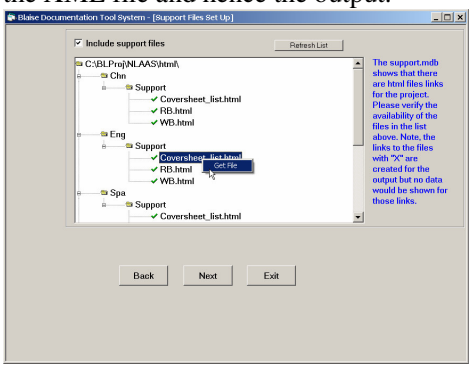
Next, select the languages to process. These are taken directly from the datamodel's defined languages. If a support database has been defined and the stylesheet that was selected indicates it, you will be asked to supply a properly marked help file that will annotate the output.

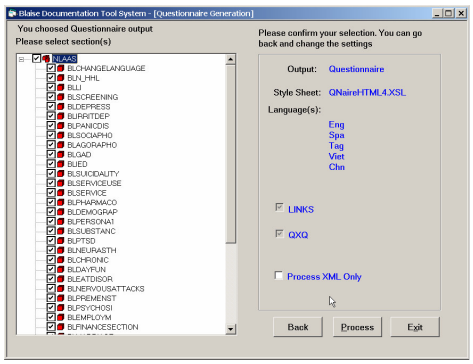




If there are respondent booklet options stored in the support database then BlaiseDoc will present a map of the project and the support files it is expecting. Right-click and choose "Get File" to copy the appropriate html file to your project (output) directory.

Finally, the program will ask you to select the blocks of data to process. The entire datamodel will be scanned so that fills and gotos are resolved, but only those blocks chosen will be part of the XML file and hence the output.

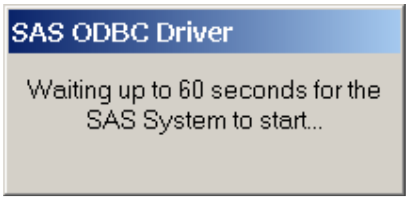




If the "Process XML Only" option is chosen it will run the stylesheet transformation against the current XML files. Transformation of just the chosen blocks is a future feature.

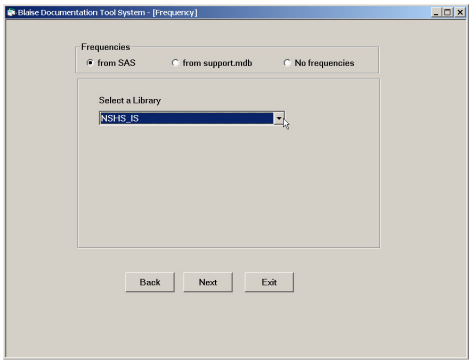
5.2 Codebook method

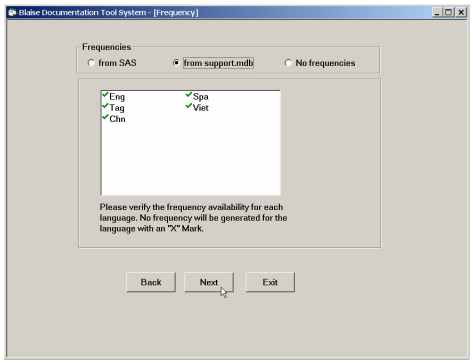
The process for creating a codebook is the same as a questionnaire except that there are additional options for the user. If SAS© has been installed on the system and the codebook stylesheet is one that uses frequencies then SAS© will start up.



The user then has the option of choosing frequencies from SAS©, from the support database (to be discontinued), and no frequencies. The support database frequencies are stored in Access and simply report which language frequencies will be generated based upon the earlier language selection.

The SAS© frequencies are taken from the selected ODBC library.





Note: if more than one language is available then a combined "all" language is assumed. BlaiseDoc expects frequencies for each language, including an "all." If there are no frequencies for a language or a particular field, then no frequencies for that question(s) will be shown.

6. Example output

The following examples are from the National Latino and Asian American Study, "all" languages.

6.1 Codebook (html)

Note the fill "this experience / these experiences" in brackets, the help reference "QxQ" and the "Respondent's Booklet" reference. The QxQ and RB references work the same across codebooks and questionnaires. The user can click on hot link to open another html document with the information about the question.

DS3

☞ (RB, PG 53)

What do you think was the main reason for [this experience / these experiences]? Would you say?

- ☐ 1 YOUR ANCESTRY OR NATIONAL ORIGIN OR ETHNICITY
- ☐ 2 YOUR GENDER OR SEX
- ☐ 3 YOUR RACE
- ☐ 4 YOUR AGE
- ☐ 5 YOUR HEIGHT
- ☐ 6 YOUR SKIN COLOR
- ☐ 7 YOUR SEXUAL ORIENTATION
- ☐ 8 YOUR WEIGHT
- ☐ 9 YOUR INCOME OR EDUCATIONAL LEVEL
- ☐ 10 OTHER

98 DON'T KNOW

99 REFUSED

View this question in...

External Links
QxQ | Respondent's Booklet

Universe
BLSCREENING.SC19=C01
DS2=C02

BLDS.DS3

Value	Label	Frequency	Percent	Cumulative Frequency	Cumulative Percent
.	MISSING	1437			
D	DON'T KNOW	149			
R	REFUSED	13			
1	YOUR ANCESTRY OR NAT	836	25.60%	836	25.60%
2	YOUR GENDER OR SEX	167	5.11%	1003	30.72%
3	YOUR RACE	711	21.78%	1714	52.50%
4	YOUR AGE	218	6.68%	1932	59.17%
5	YOUR HEIGHT	37	1.13%	1969	60.31%
6	YOUR SKIN COLOR	136	4.17%	2105	64.47%
7	YOUR SEXUAL ORIENTAT	16	0.49%	2121	64.96%
8	YOUR WEIGHT	66	2.02%	2187	66.98%
9	YOUR INCOME OR EDUCA	294	9.00%	2481	75.99%
10	OTHER	784	24.01%	3265	100.00%

- **Position:** 4581
- **Blaise Type:** Enumerated
- **SAS© Type:** Numeric
- **Decimals:** 0
- **Minimum:** 1
- **Maximum:** 10
- **Missing Data Codes:** ., .D, .R
- **Empty:** N

6.1.1 Codebook (html) – QxQ

The following is output from clicking on the QxQ (Question-by-Question objective or online help) link.

DS3

The Respondent is asked to identify the main reason for which s/he believes s/he had experiences related to discrimination. Record all the reasons mentioned. If the Respondent gives another reason that was not on the list, be sure to record it verbatim.

6.1.2 Codebook (html) - RB

The following is output from clicking on the RB (Respondent Booklet) link.

WHAT DO YOU THINK WAS THE MAIN REASON FOR THIS/THESE EXPERIENCE(S)? WOULD YOU SAY . . . ?

- Your ancestry or national origin or ethnicity
- Your gender or sex
- Your race
- Your age
- Your height
- Your skin color
- Your sexual orientation
- Your weight
- Your income or educational level
- Other (Specify) _____

6.2 Questionnaire (html)

This sample question demonstrates the added DK/RF attributes on a range question. Note the numerical range programmed in the instrument is actually 0 through 97, but the allowable responses are 0 through 10, 97, DK, RF. Anything that is marked in magenta is typically hot linked to the appropriate item. For example, clicking on the GO TO _IC_9 reference will take the user to _IC_9 internal checkpoint in the current document.

G5

What is the longest period of months or years in a row you ever had when you were [worried or anxious / nervous or anxious / anxious or worried] <u>most days</u>?

IF VOL 'WHOLE LIFE' OR 'AS LONG AS I CAN REMEMBER,' CODE: 995 YEARS

INTERVIEWER: FIRST ENTER THE NUMBER

0 - 995	Actual Range
998	DON'T KNOW
999	REFUSED

View this question in...

Internal Checkpoint G5_IC_8

((G5 <> DONTKNOW) AND (G5 <> REFUSAL)) AND (G5 <> 995)

1	EXPR IS FALSE **GO TO _IC_9**
2	EXPR IS TRUE

View this question in...

G5a

ENTER THE UNIT OF TIME

☐	1 DAYS	**GO TO _IC_9**
☐	2 WEEKS	**GO TO _IC_9**
☐	3 MONTHS	**GO TO _IC_9**
☐	4 YEARS	**GO TO _IC_9**

View this question in...

G38d

☞ (RB, PG 9)

(Using a 0 to 10 scale on page 9 of your booklet, where 0 means no interference and 10 means very severe interference, think about the month or longer in the past 12 months when your [worry or anxiety / nervousness or anxiety / anxiety or worry] was most severe. What number describes how much your [worry or anxiety / nervousness or anxiety / anxiety or worry] interfered with each of the following activities during that month or longer?)

Your social life?

(IF NEC: How much did your [worry or anxiety / nervousness or anxiety / anxiety or worry] interfere with your social life during that time?)
(IF NEC: You can use any number between 0 and 10 to answer.)

IF 'DOES NOT APPLY',CODE 97

0 - 97	Actual Range	
98	DON'T KNOW	**GO TO G44**
99	REFUSED	**GO TO G44**

View this question in...

External Links
Respondent's Booklet

Consistency Checkpoint

Valid condition:	((((G38d >= 0) AND (G38d < 11)) OR (G38d = 97)) OR (G38d = DONTKNOW)) OR (G38d = REFUSAL)
Error returned to the user:	Error in G38d. Valid answers are 0 thru 10 and 97
Involved fields:	BLGAD.G38d

Constructed Variable G39

*INTERVIEWER CHECKPOINT (SEE *G38a - *G38d)*

1 ALL FOUR RESPONSES TO *G38a - *G38d SERIES EQUAL '0'
2 ALL OTHERS
8 DON'T KNOW
9 REFUSED

View this question in...

6.3 Questionnaire (rtf)

The following sample questions have been taken from a first draft RTF output using a training tutorial. Note that many features present in the HTML format are no longer available, such as drop down lists or hotlinks to other targets.

BAdvanced.MOD3_11
In the example question listed below, enter a 7 for "OTHER - SPECIFY" and type "I will be replacing my car with a motorcycle" in the specify window.

"If you were to replace your car, would it be with another car, some type of van, pickup, or utility vehicle, or what?"

1	**CAR**	**(GOTO BAdvanced.MOD3_13)**
2	**VAN**	**(GOTO BAdvanced.MOD3_13)**
3	**PICKUP**	**(GOTO BAdvanced.MOD3_13)**
4	**UTILITY VEHICLE**	**(GOTO BAdvanced.MOD3_13)**
7	**OTHER - (SPECIFY)**	
8	**DON'T KNOW**	**(GOTO BAdvanced.MOD3_13)**
9	**REFUSED**	**(GOTO BAdvanced.MOD3_13)**

Question Type: Enumeration

BAdvanced.MOD3_12
Please specify OTHER response.

Remember, to exit the response box, press [TAB] then [Enter], or press [ALT] and the "S" key together. To move to the next question afterwards, press [Enter].

DON'T KNOW Allowed
REFUSAL Allowed

Question Type: Memo

7. Technical problems encountered

BlaiseDoc was developed over a short time frame. Programming started in February 2004 and the first complete release was done July 2004. Not all development work went smoothly and there are certain surprises that appeared.

7.1 Slow navigation - RulesNavigator

A requirement of producing questionnaire documentation is to write out the questions in exactly the same order as presented to the interviewer. This bit of navigation was first done using the RulesNavigator within the BCP. This method, however, had several shortcomings that contributed to the slowness.

7.1.1 One RulesNavigator object per instance

Because of how BCP works with the datamodel, it usually is practical to only work with one copy of it in memory. Therefore all reference to the datamodel is done using pointers to the object, and any navigation via the RulesNavigator (perusing the rules) affects the content of all

pointers. This would be similar to having a shared document; it gets updated and everyone sees the changes. This is usually not desirable when attempting to "hold" a place pointer, perform a look ahead, and then resume from the same spot.

Therefore it was necessary to have two copies of the same datamodel loaded into memory to accomplish this method.

7.1.2 Large overhead for loading the datamodel

As one can surmise, a large datamodel takes a chunk of memory and some time to load it. For every datamodel beyond the first additional system resources are required.

7.1.3 Inability to move to a field and navigate at random

The RulesNavigator method did not provide a method to move to a particular statement in the rules, or to pick up navigation from other methods (such as the Statements collection). It is purposely made to execute through all the rules of the datamodel from the first statement. In order to do a look ahead and return back to the same spot then required noting the statement the RulesNavigator started from, moving to a new destination, then start at the beginning of the rules and coming forwards again to resume.

7.1.4 Heavy recursion

The particular method used to navigate with the RulesNavigator made use of recursion. Unfortunately very complex instruments did force BlaiseDoc to run out of stack space (set by VB around one megabyte).

7.2 Memory use

BlaiseDoc is very string manipulation intensive and hence is very memory intensive. It relies upon aVisual Basics method to collect used strings and reallocate memory. However, Visual Basic hangs onto the memory used by a process until the process is released. This has the effect of slowly eating away the memory (although the process may currently be using a relatively small amount) until virtual memory comes into play (using disk space for memory). Once that happens processing speed dramatically slows down.

7.3 Not all information available - Procedures

BlaiseDoc has been limited by what is available via BCP (Blaise Component Pack). For example, any procedures written in the Blaise program are not accessible via BCC. For example, fills created by procedure calls cannot be fully resolved. Therefore, complete documentation is currently available for instruments without such calls.

7.4 Reusing variables for fills

As noted earlier, a common programming habit has been to reuse temporary variables for fills. This tends to lead to overly abundant text options when displaying fills on later questions. The problem still exists in the current release.

8. Programming Solutions & Suggestions

By far, the most impressive improvement in performance came from changing the method of rules navigation in the datamodel. The Statements collection provides nearly the same functionality as the RulesNavigator but does requires the programmer to manage the navigation through the instrument. The Statements collection offered the following advantages:

8.1 Simple loop navigation within minimum recursion

Instead of evaluating conditions to determine whether or not to go down an execution path, the Statements collection lists every statement within a block. The program simply loops through this list, and if it encounters a block it recurses to that block. This avoids the heavy recursion method using the RulesNavigator.

8.2 Can navigate freely

A particular statement can be reached quickly by using StatementIdent, an internal identifier to each statement in the Blaise datamodel. Using this method and the Statement navigation methods BlaiseDoc can freely and quickly examine statements to resolve gotos.

8.3 One datamodel only needed

Because quick navigation is now available, only one datamodel ever needs to be loaded. This reduces overhead greatly.

8.4 Low memory overhead

The routines that had once used pointers to the datamodel just pass a StatementIdent. With less recursion, one datamodel, and smaller parameters the memory use has been greatly reduced. However, the same problems for string manipulation note before are still present.

Given the limitations in the BlaiseDoc environment, the following suggestions are provided in order to help produce a more meaningful output.

8.5 Avoid procedures

Since BlaiseDoc uses the BCP, and BCP cannot read within procedures, place assignment of fills within auxblocks or within the code. This is obviously a less-than-optimal solution and is only suggested so that BlaiseDoc can fully quantify fills and goto statements.

8.6 Do not reuse fill variables

This should be taken only as a suggestion, but fills are not being resolved contextually within the rules section. So if fills are assigned multiple times in the same block the later questions will display too many options for that question.

9. Future enhancements

BlaiseDoc is still in its early stages and there is a list of enhancements waiting to be implemented.

9.1 Full meta information to be stored

If the goal is to reproduce the Blaise source code from the XML file then additional information needs to be stored. Currently only relevant information to producing documentation has been stored in the XML file. Certain categories of datamodel information have been left out, such as the layout, many modelib settings, and the datamodel properties.

9.2 Procedural information missing

BlaiseDoc has been limited by what is available via BCP (Blaise Component Pack). The current release of the BCP does not provide a method to retrieve the procedural information, and hence this limits the usefulness of some items (evaluating gotos and resolving fills).

9.3 Reading .HLP file directly

BlaiseDoc currently uses the original RTF file that has been annotated for use by the Microsoft HelpMake utility in order to merge the help file into the XML document. A future improvement will read the .hlp file directly since the source file is often not found with the datamodel.

9.4 Fills substitution library

In order to produce a readable document some fills need to be changed by hand. For example, ^xFill[1] may be correct, but if it is assigned within a procedure then BlaiseDoc cannot resolve it any further. Hence an external database that translates xFill[1] into "easy/fast" would be much more readable. This same library could be used for fill expressions such as "A[1] + ' and ' + SUBSTR(xLocation, 1, 5) + ' or that.'"

9.5 More context sensitive fill resolution

Fills that are reused between questions appear to have more values in later questions than in earlier ones. The reason behind this is they did indeed have all the values listed, but only a subset of those values are valid in the context of the question. The routine to determine fills needs to be more intelligent and determine only those values for a fill for the currently specified question.

That is, suppose before A1 the fill xFill was assigned either "one" or "two" based upon some count. Question A1 displays the fill, and BlaiseDoc will show the fill as "(one/two)." Then just after A1 the fill is reused and is assigned either "three" or "four" and used in A2. Question A2 will now show "(one/two/three/four)" instead of just "(three/four)."

9.6 Full evaluation of expressions

Because of the complexity involved in determining an expression for all its values, BlaiseDoc produces a number of "internal checkpoints." The output would be much tighter (especially more gotos via enumerated questions) if BlaiseDoc interpreted the following base expressions:

<
>
<=
>=
NOT
function calls
IN operator

9.7 Clear indication of arrays/question blocks

The current output does not show the start/end of arrays of blocks or the logical range for questions within a block. The latter may be of interest to programmers and data analysts.

9.8 Reprocessing Stylesheets

The "Process XML Only" option on the last screen of the BlaiseDoc wizard runs the current stylesheet transformation against the current XML files. An option for the future is to intelligently reprocess those blocks chosen so that processing of very large or complex datamodel can be done in steps.

10. Conclusion

BlaiseDoc is still early in its lifecycle but has proven to be a valuable tool for producing documentation from the Blaise system. We expect it will continue to improve and expand to its full potential.

If you have questions or comments, please contact Peter Sparks (zebulon@umich.edu) or Youhong Liu (yliu@umich.edu).

An evaluation of Delta, a documentation tool

Rebecca Gatward, Office for National Statistics, UK.

1. Introduction

Since the 1980s CAI has become the main method of survey data collection. CAI software allowed researchers to meet the demand need for more complex survey questionnaires. The demand for a tool to document CAI questionnaires followed. CAI users throughout the survey process require a 'human readable' version of electronic questionnaires, for example, survey commissioners, researchers, methodologists, interviewers, data analysts and archivists.

The need for 'human readable' versions of Computer Assisted Interview (CAI) questionnaires is now well accepted. This issue described as the 'Global documentation problem' has been well discussed and debated at workshops, conferences, seminars and in papers. This paper will not revisit these arguments as these have been documented elsewhere. Bethlehem (2001), *'Survey Automation: report and workshop proceedings'* (2003)

Delta is the first documentation tool to be incorporated within the Blaise software. This paper will provide an early assessment of the Delta tool and will provide some recommendations for further development. The evaluation will be based on the set of function requirements that were the outcome of the User Requirement Survey which formed part of the TADEQ project. Delta developed from the TADEQ tool.

CAI has many advantages over paper questionnaires except to date, when it comes to documentation. During the early stages of CAI software development, developers of the CAI software packages quite rightly did not rank documentation tools as high priority. They concentrated their efforts on producing systems that would securely collect accurate data. Once this was achieved developers then turned their attention to meeting the demands of their growing numbers of users by improving the functionality of the packages. Producing a 'human readable' version of a CAI questionnaire is one function that has, until recently, not been tackled by the developers of CAI software. This forced survey organisations to tackle the problem themselves by developing their own 'in-house' documentation tools. Examples include the tool developed by the Office for National Statistics (ONS), which could automatically produce documentation from the Blaise Metadata files, and the US Census Bureau funded IDOC (Instrument Document) for CASES.

What occurred was a situation where organisations were working independently to solve the documentation problem themselves. The TADEQ project aimed to coordinate these efforts and provide a joint approach to finding a solution to this problem.

1.1 The TADEQ project

The TADEQ project (Tool for the Analysis and Documentation of Electronic Questionnaires) is a Fourth Framework Project of the EU. The project pooled the efforts of National Statistical Institutions from four European countries (Netherlands, United Kingdom, Portugal and Finland) and the Max Planck Institute in Germany[1]. In summary the aim of this project was to find out what Computer Assisted Interviewing users required from a questionnaire documentation tool (via a User Requirement Survey) and to develop a first prototype tool to meet these needs.
The outcome of the project was the TADEQ prototype which is capable of documenting and analysing the contents and structure of electronic questionnaires. TADEQ is designed to be a neutral tool that can be used to process any electronic questionnaire capable of being exported

[1] During the course of the project people involved at the Max Planck Institute moved to the Institute of Computer Graphics at the University of Vienna.

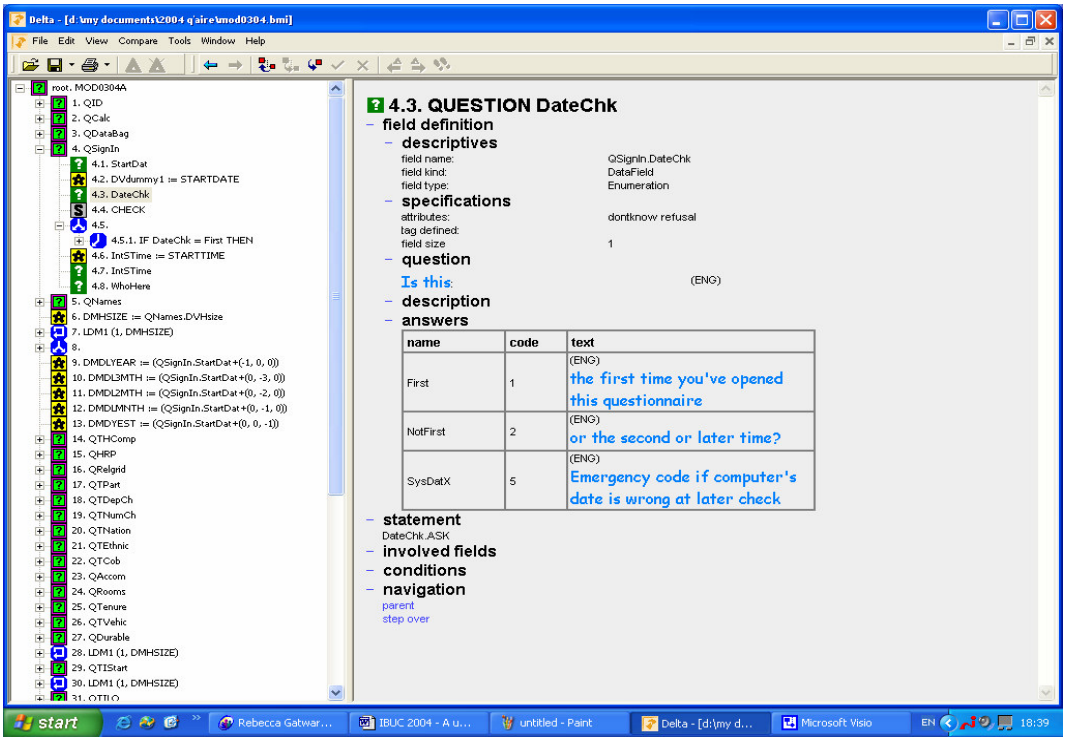
to XML. More details about the TADEQ prototype can be found in the 'Using the TADEQ prototype' (Bethlehem, Hundepool, Van de Wetering, 2000). There are overviews in earlier IBUC papers including (Bethlehem, J.G. and Hundepool, A.J. 2001).

1.2 Introduction to Delta

The first version of Delta was released as part of the Blaise Component Pack in June 2003.
Delta is an acronym of Documentation tool, using ELements of TAdeq. As this name suggests, the Blaise developers at Statistics Netherlands have used the TADEQ prototype as a basis from which to build the Delta documentation tool. Their goal (as stated in the documentation that accompanied the Delta release) was to 'offer improved functionality of TADEQ, built upon Blaise components'. Other documentation tools have been developed by organisations using Blaise. However, Delta is the first such tool to form part of the Blaise software.

Blaise developers at Statistics Netherlands have taken some of the functionality of TADEQ and incorporated it into Blaise. As stated in the Delta on-line help, the two main elements of TADEQ that are not yet in Delta are the graph module which displays a questionnaire as a flow chart and the analysis options. A tool to compare two versions of a questionnaire is included in Delta.

The screen layout in Delta is almost exactly the same as TADEQ, with the tree view (displayed on the left of the screen) providing a means to navigate around the questionnaire. Detailed information about the questionnaire content appears on the right hand side of the screen. The format of the information and level of detail is controlled by stylesheets (written in XML). An example screen using the default stylesheet is provided below.



2. Evaluation of Delta

2.1. Criteria for evaluating Delta

One of the aims of the TADEQ project was to establish a set of user requirements for a documentation tool. A survey was carried out amongst users of CAI software to collect data about what they required from a documentation tool. The information collected formed a set of function requirements for TADEQ.

A summary of the TADEQ user requirements are provide below, a detailed commentary of these requirements can be found in Kelly and Kuusela (2000).

The TADEQ project found that users required a documentation tool to:

- automatically generate accurate documentation that requires little or no manual editing,
- produce comprehensive documentation which provides all the information a user requires,
- meet the needs of different users by providing a flexible tool which allows the information displayed or printed to be changed,
- generate documentation in paper and electronic form,
- produce detailed information about questions and the logical structure of the questionnaire.

As the survey process has not changed radically since the TADEQ user requirement survey was carried out in 1998, we can assume that the user requirements for a documentation tool still remain the same. The user requirements, listed above, will therefore be the criteria by which we will assess Delta. The next few sections will take each one in turn assessing to what extent Delta meets these requirements.

2.1.1. Accurate documentation

It is essential that a documentation tool automatically produces output that requires little or no manual editing to make it fit for purpose. Any manual intervention risks introducing inaccuracies, even if the person carrying out the amendments knows the questionnaire very well. Any manual editing of documentation makes it difficult to guarantee the documentation still replicates exactly what is happening in the CAI questionnaire. Manually crafting documentation to meet user needs can also take a long time, resulting in some of the efficiencies gained by using CAI rather than PAPI are then being lost on the documentation process.

Delta partly meets this criterion. The information displayed in Delta documentation does accurately reflect the information gathered in the Blaise questionnaire. This is because Delta extracts meta-information, data definition and rules that are held in the Meta information file (.bmi) along with screen layout information stored in the DEP mode files (.bdm) and translates it into Delta output.

Generating documentation using Delta is automatic and very straightforward, the user simply selects the questionnaire .bmi file and Delta returns the documentation on screen.

Delta also allows the user to manually edit what appears in the documentation. There are two main reasons why the user may wish to do this. The first is to tailor what is displayed on screen to meet user needs, the second to edit the documentation before it is printed. Manual editing of Delta documentation will be dealt with in more detail when paper and electronic versions are considered later in the paper.

2.1.2. Comprehensiveness

The second user requirement to be considered is all the information required by a user should be available in the documentation (even if the user chooses not to display it on screen).

Different users require different types of information at varying levels of detail. For example, one user may only need to view variable names and question text while another user may require detailed routing for each variable.

A table of all the information that the TADEQ user requirements survey found necessary is included as an appendix to this paper (Appendix A); the table provides some detail about whether or how Delta provides each item of information.

In summary, Delta displays most of the information users require, although some items are not shown in the most accessible format for the user.

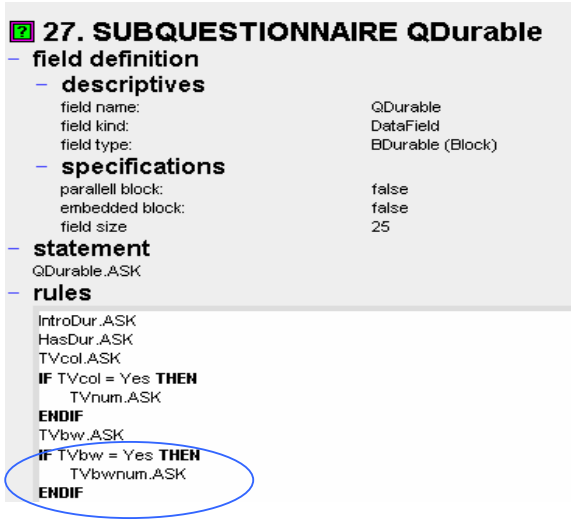
The following items are not currently provided by Delta but are required.

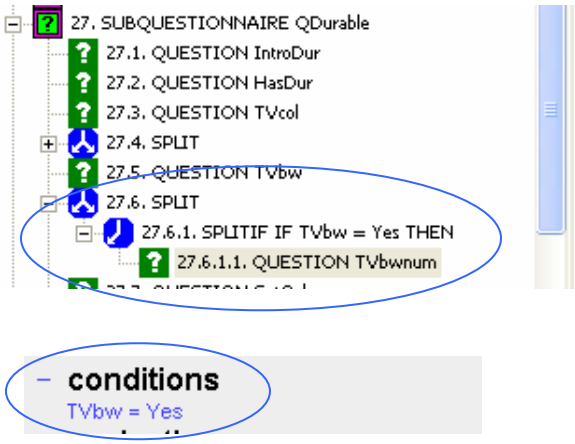
- A 'natural language' description of the questionnaire filename.
- Variable attributes.
- References to external files
- Information about the type of coding frame or type of search (e.g. Trigram)
- Details of external programs or DLLs referred to in the questionnaire.
- Text substitutions (fills) are not labelled or translated.
- 'SET OF' details to identify multi-coded variables.
- Natural language explanation of routing logic.
- Variable index (or option to list all variables).
- Information imported from external programs.
- Contents of an external coding file.
- LAYOUT commands displayed at block or field level.

I will take some of the more important requirements in more detail, starting with the way routing is dealt with by the Delta tool.

Routing

In Delta routing is displayed at three different points, at block (or sub questionnaire) level, in the tree view and at the Field level. In the example below the routing to the variable called 'TVbwnum' is highlighted at these three points.





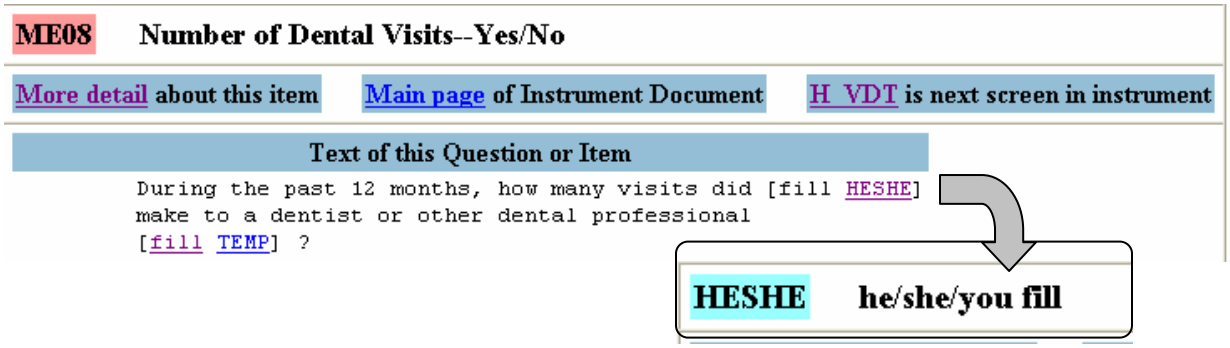
The routing is displayed in the same format as it is written in Blaise code, i.e. as IF and ELSE statements. This format may be useful for some users but not others. For example, a survey commissioner may ask for documentation to include the full routing to each variable whereas an interviewer may find routing in the 'Go to' format (as it appears in paper questionnaires) easier to follow.

Users need more flexibility around changing the format and complexity of the routing displayed. They need the option to display routing at varying levels of complexity, for example, routing between blocks (sub questionnaires) or to each field. The routing also currently uses value names: these are not always meaningful and make deciphering routing a lengthy process. It would be helpful to have the option to choose whether the value labels or value names are shown in the routing.

Questionnaire developers attempt to help users interpret routing by annotating the rules sections with natural language descriptions of the routing logic. Unfortunately this information is not currently included in the Delta output (see Appendix A).

Text substitutions

A second information requirements to be looked at in more detail is question text, specifically text substitutions. Although the format of the question text in the documentation tool is the same as it appears in the Data Entry Program, text substitutions appear as they do in the Blaise syntax. Text fills are not translated or labelled. Other documentation tools however have managed to do this. The IDOC (Instrument Document) tool (developed on behalf of the US Census Bureau) deals with fills by labelling them in the question text e.g. [fill 'variable name'] this is highlighted with links to the 'variable name' where the fill was originally defined.



This solution is not ideal because the user still has to go elsewhere to find the contents of the text fill. It is essential for Delta to provide the user with some indication of what text substitution means. Other ways of dealing with text fills are described in Kelly and Kuusela (2000).

Imported information
The TADEQ requirement survey also found that users would like to be able to import other information about a survey into the documentation tool. For example, details about the survey methodology (sample size, fieldwork dates) or history of changes to a questionnaire. It is more convenient to keep all this metadata in one document and would make it a document that is fit to be sent to data users. This is not currently possible in Delta. A solution could be to have the option of including an 'information' block within the Blaise questionnaire where any additional information about a survey could be stored and extracted by Delta.

Other information that would be helpful is a summary of the settings applied to the questionnaire during its preparation or the interview. For example, the name of the Modelib file and details from within the Modelib such as whether the audit trail facility was activated.

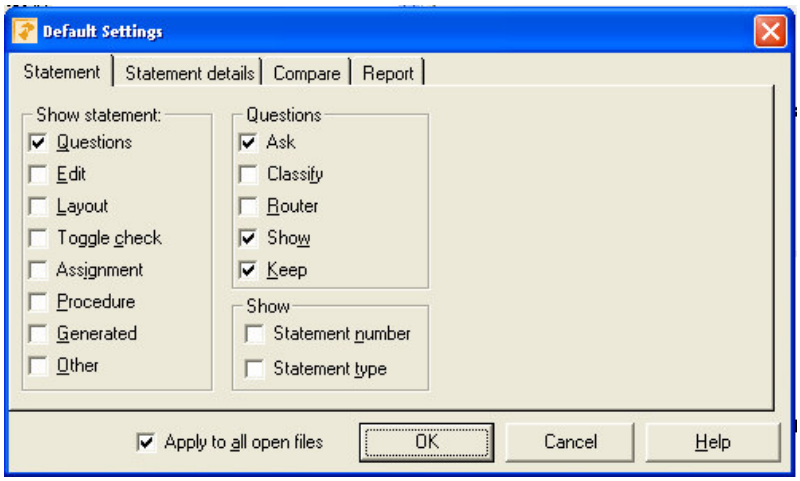
Currently any text which is specified at an Auxfield (typically an introduction to a new section of questions) is not displayed in Delta. Although this is not strictly question text, it is still important that this detail is available to users.

2.1.3. Flexible

As mentioned already, people who are involved at different points in the survey process all require different information from the questionnaire documentation. A documentation tool needs to be capable of producing the documentation, both in electronically and on paper, to meet these differing needs.

The tool needs to allow the user to be flexible in terms of language i.e. having the option to display natural language descriptions for variables and the variable names and also allow the user to choose the level of detail shown.

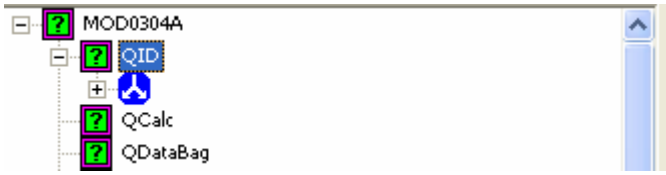
Delta does provide this flexibility. It is possible to specify which details are shown on screen or printed in the paper version. The user can choose what information is shown in the structure tree by changing the 'default settings'. The screen shot below illustrates the options available.



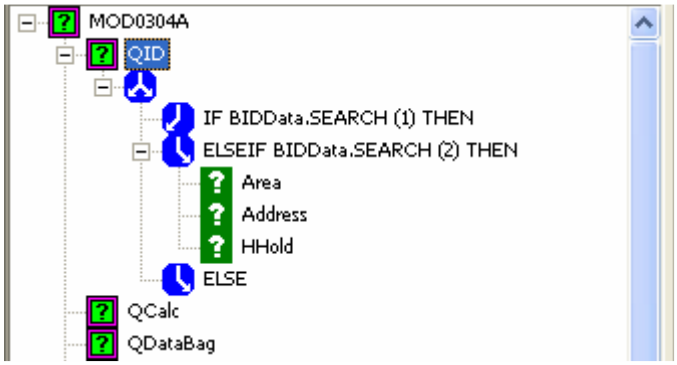
An additional user requirement was the ability to show the global structure of the questionnaire and to also have the means to focus on the detail of parts of the questionnaire. The following example illustrates that this is possible in Delta.

1.

2.



3.



A further user requirement is that it should be possible to delete some sections of questions from the documentation (for e.g. questions that survey commissioners do not need to see like administrative details). Currently this would not be possible in Delta, unless the sections were

simply removed from a paper version or an alternative questionnaire was prepared which excluded the relevant questions.

2.1.4. Paper and electronic format

A primary requirement for TADEQ was to produce well laid out, good documentation in both an electronic and paper format. The documentation should be capable of producing documentation in either format which meets all user needs. Some
components of a questionnaire are important to one type of user but less so to another.

- Interviewers are mostly interested in question text, on screen instructions, routing and on screen help instructions.
- Survey commissioners need to see the question wording, routing, checks and computations.
- Data users require question routing and computations.

Paper version

Even in the age of the 'paperless office', paper versions of CAI questionnaires are still required by users of CAI software. People working throughout the survey process require paper versions of questionnaires for a variety of purposes. Some use the paper version as an easy reference document, such as field staff whilst dealing with interviewer queries or researchers whilst analysing data. It is sometimes easier and quicker to reach for a paper copy than to open up an electronic version. Many people also find it quicker to read and navigate a paper copy than an electronic version.

Currently the paper documentation offered by Delta is simply a print out of the electronic version. This format is not ideal for a paper version. It is possible to tailor what is printed to meet user's needs. Information (statement details) appear on the right hand side of the screen, and stylesheets are used to control how this is displayed. Currently however only one default stylesheet is provided with Delta so it is down to the user to amend the stylesheet. Amending the stylesheet, which is written in XML, is not a straightforward task especially for a 'non-programmer'. At least it is possible to save the stylesheets which means that stylesheets only have to developed once to meet different user requirements or to a 'housestyle'. This task could be simplified for the non programmer if a stylesheet editor was provided with Delta (like the one provided with Blaise IS).

Printing a paper copy of the Delta documentation is also problematic. It takes a long time to print (the on-screen estimate was 60 minutes for a surveys of average length and complexity) and frequently crashes the PC before the whole document has printed.

The Delta tool allows the user to add some formatting to a paper version, such as page numbers, headers and footers. This makes it easier for the user to find their way around what can be a very long document. Other options are print preview, the option to print just the tree view, the statement details or all details and also to print more than one page per sheet. Delta does not give the user the option to add an index or contents page or to print more than one variable on one each sheet. Having an option available to produce an index based on variables names is particularly important.

Exporting the output to a package that offers more formatting options is possible but would require saving the document as a HTML file (the only option available in Delta) then resaving

this version as another type of file. This process is time consuming and additional time would then have to be spent reformatting the text.

The advantage of saving the Delta output as a HTML file is that the functionality of the document remains within the document, for example the ability to expand the branches. However saving the document is a very long process, the onscreen estimate for gathering and saving all details in a questionnaire of average length and complexity was 53 minutes.

The addition of an index of variables would be helpful. This could be used in both the paper and electronic versions. A possible format is a list all variable names alongside a brief description of the variable, with links that allow the user to jump to any variable within the documentation.

Many websites now offer the option of printing a 'printer friendly' option that is formatted specifically to be printed rather than viewed on screen. Delta would benefit from a similar feature.

Electronic format

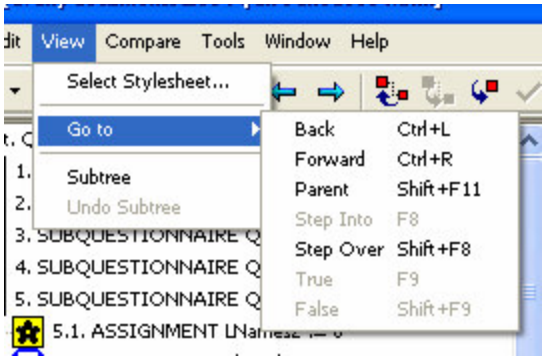
Some users who took part in the TADEQ requirement survey did use electronic versions of documentation (although all still used a paper version as well). The advantages of electronic documentation are interactive use, easier navigation around the questionnaire, search and help facilities available on-line and option to use standard windows facilities, for example, being able to copy text into a different package.

Interactive use

As described under the 'flexible' criterion above, Delta allows the user to show and hide information as required.

Navigation

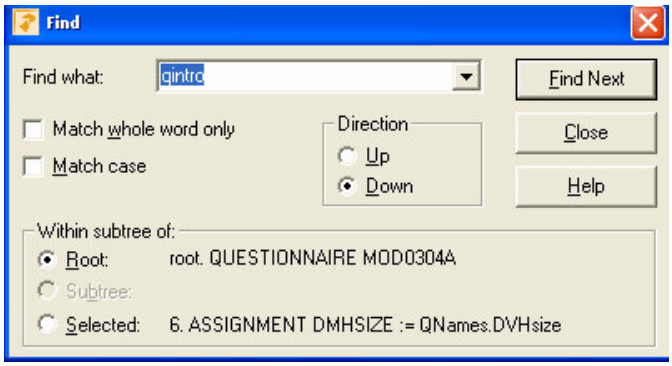
Navigation is via the tree view on the left hand side of the screen. This system of navigation works well with small questionnaires but it is easy to get lost in the detail when looking at a large complex survey. Different symbols are used to identify the various statements (such as a white question mark with a dark green background representing a question), this is helpful. The user can also find their way around the documentation using the 'Go to' options (shown below) or the hyperlinks within the document.



Navigation could be improved by adding links to variable names, this would allow the user to jump to variables mentioned in the condition or rules sections rather than using

the search facility. A further improvement would be the addition of an 'expand all' and 'collapse all' option to the 'view' menu.

Search and help facilities
The search (or 'Find') option in Delta allows the user to search for any text within the documentation, either within the whole document (from the 'Root') or within selected sections. Each occurrence of that text string is highlighted on screen.



The Help facility provides more detail about all aspects of the Delta tool.

Standard window utilities
It is only possible to save the statement details as an HTML file and the tree as a JPEG. It is not possible to export to other packages. Exporting to WORD would be useful.

2.1.5. Ability to produce detailed information on the logical structure of the questionnaire

The Delta tool only incorporates some of the functionality of the TADEQ prototype. The functions that are currently missing from Delta are the analysis and graphical tools; it is these elements that would meet this final user requirement.

Graphical tool

Many of the users who took part in the TADEQ user survey stated that a flow diagram would help them and other users gain an overall view of the flow of a questionnaire. For example, a flow diagram would be useful for interviewers to gain a picture of how respondents with differing characteristics would be routed through a questionnaire. The flow diagram is also an ideal format to display 'go to' routing.

Users should also be able to use the flow chart to drill down to sections of the questionnaire they are particularly interested in, for example, from the overall structure of questionnaire down to the routing within a block and then the logic behind the derivation of a computed variable. These functions are available in TADEQ.

Jabine (1985) advocates flow charts as a useful tool that can aid the questionnaire development process, both in the early and later stages of design and also for training interviewers.

Analysis tool

The TADEQ user requirement survey did collect some information from users about what they needed from an analysis tool. These are detailed in Kelly and Kuusela (2000).

The following analysis functions were available in the TADEQ prototype.

- Calculation of the number of paths through the questionnaire.
- average length of interview, calculated using estimated time taken to answer each question.
- Frequency of the different elements of a questionnaire for e.g. number of checks, open questions etc.

Compare tool

Alongside the documentation element a comparison function has been included in Delta. The compare tool compares two versions of a questionnaire finding and documenting the differences between them. This is a time saving tool, although useful enhancements would be the addition of an option to select sections of the questionnaire to compare rather than whole questionnaire and an option to select changes that can be ignored, such as additional toggles around text and spaces.

3. Summary and recommendations for future development

The first version of Delta partially meets some of the TADEQ user requirements. In its current form Delta:

- automatically generates documentation very quickly,
- provides some of the information users need,
- is flexible – in that the user can choose what and how information is displayed,
- provides documentation in an adequate electronic format,
- provides a tool to compare two questionnaires.

Aspects that require particular attention for development are:

- use of natural language, e.g. for labels, for all Blaise elements, including computations,
- displaying routing in a more meaningful way,
- providing a 'printer friendly' format option,
- providing a graphical or analysis tool that meets user requirements.

In addition to the recommendations detailed under each evaluation criterion, the following recommendations that were not explicitly found to be user requirements by the TADEQ project but would be very useful enhancements to the Delta tool.

Use Delta to analyse audit trail data

Delta could be developed to be a tool to analyse and summarise audit trail data. The analysis component of the TADEQ prototype calculates an average interview length based on estimated length of time taken to ask each question. Now that actual timings can be calculated using audit trail data, Delta could provide accurate timings rather than

estimates. The tool could produce actual interview length and average block timings. These timings could be provided for the whole sample or by respondent type. Users could be given the option of selecting the type of respondent, for example, females aged 60 years and over, Delta would list the questions which could be asked of that sub sample and the average interview length.

Alongside the work required from the Blaise developers, Blaise users must also play their part in improving documentation. Delta can only extract what is in the Blaise questionnaire. Questionnaire developers need to be encouraged to add descriptions and annotations in natural language; this is in line with the general push for more accurate and thorough metadata throughout the world. Blaise may need to be enhanced to permit labels in all elements.

4. References

Bethlehem, J.G., Hundepool, A.J. and Van de Wetering (2000) A. *Using the TADEQ prototype. TADEQ Working paper 6*, Statistics Netherlands, Voorburg, The Netherlands

Bethlehem, J.G. and Hundepool, A.J.(2001) *TADEQ: A Tool for the Analysis and Documentation of Electronic Questionnaires* Pre-proceedings of the ETK-NTTS 2001 Conference, Volume 1, pp. 133-143

Bethlehem, J.G. (2001) *Documentation of electronic questionnaires*, Statistics Netherlands, Voorburg, The Netherlands 2001

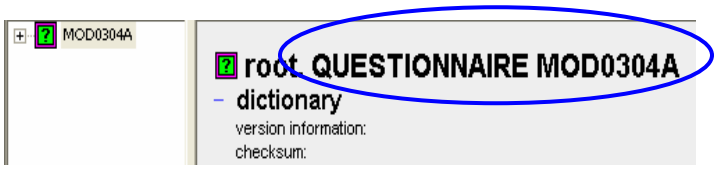
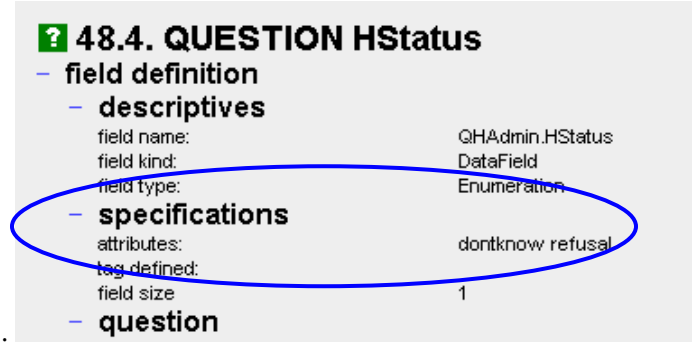
Bethlehem, J.G. and Hundepool, A.J. (2001) *Automated Questionnaire Documentation. Research Paper 0124*, Statistics Netherlands, Methods and Informatics Department. Also presented at the 7th International Blaise Users Meeting, Washington D.C., 11 September 2001

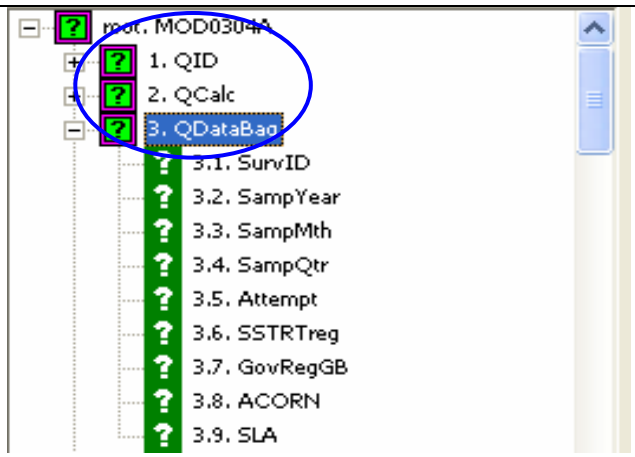
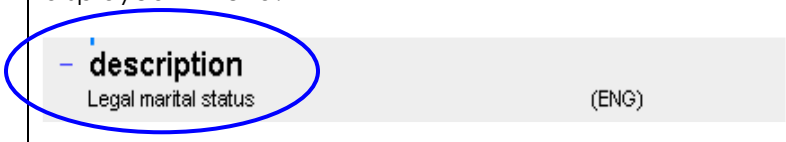
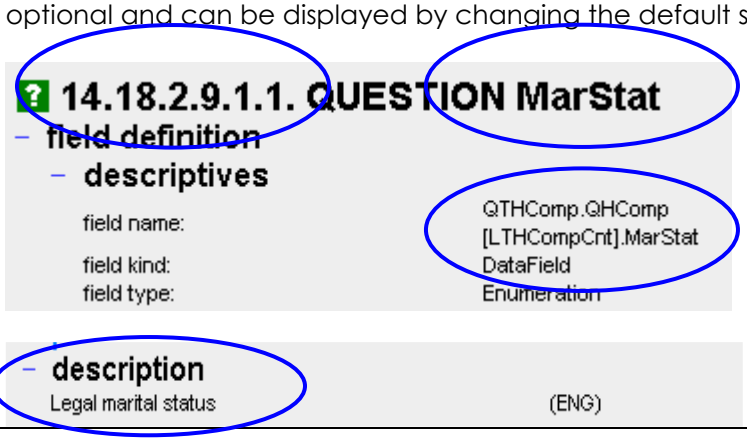
Jabine, T. P. (1985) *Flowcharts: A tool for developing and understanding survey questionnaires*, Journal of Official Statistics 1, pp. 189-207

Kelly, M. and Kuusela, V. (2000) *User requirements TADEQ Working paper 1*, Statistics Netherlands, Voorburg, The Netherlands

'Survey Automation: report and workshop proceedings' Committee on National Statistics (CNSTAT) 2003

Appendix A - Information users require from a documentation tool

Information required	What is available in Delta	
Questionnaire file name and description	Filename is provided at the highest level of the tree. File description is not shown even when this information is included in the Blaise syntax. For example: • the Blaise syntax `DATAMODEL MOD0304A "BDSS code for HOUSEHOLD surveys"` • how this appears in Delta output 	✓ ✗
Global and variable attributes	Global attributes are not displayed or available in the documentation produced by Delta. Variable attributes are displayed, for example: 	✗ ✓
External files	References to external files are not displayed in the Delta output. Information about the type of coding frame or search is also not included.	✗ ✗
External programs/DLLs	Details of any external programs or DLLs referred to in the Blaise questionnaire are not displayed in Delta.	✗
Questionnaire sections	Questionnaire sections are listed in the tree view and can be used to navigate around the questionnaire. An example:	✓

	 Delta allows the user the option to display individual sections or the whole questionnaire on the screen.	✓
Structure diagram	The structure diagram (tree) in Delta can be expanded or collapsed to display an overview or to focus on sections of questionnaire in more detail. This diagram can also be printed if required.	✓
Natural language descriptions	The variable description (a short meaningful description of the variable) is one of the statement details that are displayed on the left hand side of the screen in Delta. As long as that information is added in the Blaise questionnaire by the questionnaire developer. e.g. The description for a variable named MarStat is specified in the Blaise code as 'legal marital status' . This description is displayed in Delta. 	✓
Information about variables • **Full name** • **Short name** • **Variable numbers** • **Variable description**	All the information about variables listed as a 'user requirements' are displayed in Delta documentation. The variable numbers are optional and can be displayed by changing the default settings. 	✓

Question text	Question text is displayed as it is written in the Blaise syntax. However, text substitutions (fills) are not labelled or translated. For e.g. 46.93.1.7.1.3.1.1.1.1. QUESTION Spinc – field definition + descriptives QRenters specifications – question ^LIncTxt3 have any separate income of ^LIncTxt5 own?	✓ ✗
Answer types	Information about answer types are displayed on screen. Some examples: • **Enumerated field** Although enumerated types are identified, although there is nothing displayed to indicate whether the variable is a multi-code. In the example below it is only possible to identify this because 'Code all that apply' instruction is included in the question text and displayed in infopane. 42.1.2.1.1.3.1.1.1.1. QUESTION Ben1Q – field definition – descriptives field name: QTBen.QBen[LTBenCnt].Ben1Q field kind: DataField field type: Enumeration The answer categories are also shown along with the value names and labels. • **String** String variables are labelled 'Integer' rather than string. field type: integer specifications attributes: empty dontknow refusal tag defined: field size 3 – question Age for whole sample, from Birth and AgeIf (ENG) – description Age (ENG) – answers number : [0..120]	✓

	• **Datetype** field kind: DataField field type: Date – **specifications** attributes: dontknow refusal tag defined: field size 8 – **question** ^LColour What is your date of birth? For day not given... enter 15 for day For month not given... enter 6 for month (ENG) – **description** Date of Birth (ENG) – **answers** Date	

Routing	In Delta routing is displayed at three levels. The entire RULES section for an include file or block is displayed at the block level, they appear in the same format as they do in the Blaise syntax, for example:
	22.1.2. SUBQUESTIONNAIRE QCob
	– **field definition**
	– **descriptives**
	field name: QTCob.QCob[LTCobCnt]
	field kind: DataField
	field type: Block
	– **specifications**
	parallell block: false
	embedded block: false
	field size 49
	– **statement**
	QCob[LTCobCnt].ASK
	– **rules**
	Name.SHOW
	Lcob2 := 0
	lcob4 := 0
	lcob6 := 0
	FOR Lcob1:=1 **TO** DMHSIZE **DO**
	IF AxCob2[Lcob1] = RESPONSE **THEN**
	Lcob2 := Lcob2 + 1
	LcobA[Lcob2] := AxCob2[Lcob1]
	ENDIF
	ENDDO
	FOR Lcob3:=1 **TO** AxCob1 **DO**
	lcob4 := 0
	IF Lcob3 = 1 **THEN**
	LCobB[1] := LcobA[1]
	lcob6 := 1
		✓

Routing to variables within the block are
Displayed at the variable level under the heading conditions.

```
– conditions
  1 <= LTQualsCounter <= DMHSIZE
  LDMINDINTERVIEW[LTQualsCounter] = 'now'
  QTHComp.QHComp[LTQualsCounter].DVage < 70
  QTIStart.QIStart[LTQualsCounter].PersProx = InPerson
```

It is possible to specify if the rules or conditions section are displayed.

Information which has been added by the questionnaire developer that may help others understand the routing logic is not displayed in Delta.

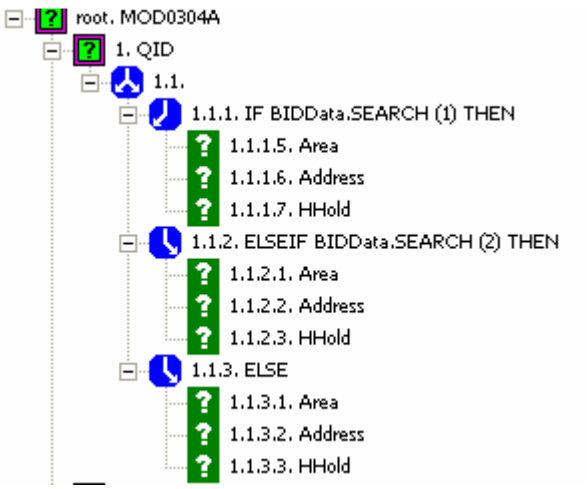
In the example below the developer has annotated the rules section.
This information does not appear in the rule section in Delta.

✘

```
RULES { BNation }

 Nation
 IF (Other IN Nation)"Nation=Other" THEN
   NatSpec
 ENDIF
```

```
– rules
  Nation.ASK
  IF Other IN Nation THEN
     NatSpec.ASK
  ENDIF
```

Variable list	No option just to list all the questionnaire variables on screen. It is possible to change the default setting to display just the questions in the tree view. For example:  	✘
Allow other information to be imported into the tool	It is not possible to import additional information into the tool.	✘

Edit checks	Checks and signals are displayed along with routing at the block level.
	They are not identified as a separate item in the tree view so it is not possible to specify whether they should be displayed.
	
	43. SUBQUESTIONNAIRE QMortgag
	– **field definition**
	– **descriptives**
	field name: QMortgag
	field kind: DataField
	field type: BMortgag (Block)
	– **specifications**
	parallell block: false
	embedded block: false
	field size 19
	– **statement**
	QMortgag.ASK
	– **rules**
	MorgYr.ASK
	IF MorgYr = RESPONSE **THEN**
	CHECK
	MorgYr <= QSignIn.StartDat.YEAR
	ENG The year started paying mortgage cannot be in the future
	SIGNAL
	IF QTHComp.QHComp[QHRP.DVHRPNum].Birth = RESPONSE **THEN**
	MorgYr >= QTHComp.QHComp[QHRP.DVHRPNum].Birth.YEAR
	ENG The year started paying mortgage cannot be before the HRP was born
	ENDIF
	SIGNAL
	IF (QHRP.HRPPart < 11) AND (QTHComp.QHComp[QHRP.HRPPart].Birth = RESPONSE) **THEN**
	MorgYr >= QTHComp.QHComp[QHRP.HRPPart].Birth.YEAR
	ENG It is unusual for the year started paying mortgage to be before the HRP's partner was born
	ENDIF
	ENDIF
		✓
		✗

	34.1.2.1.1.1.1.11.1.1. CHECK SEmpStY > ((QSignI... **condition** SEmpStY > ((QSignIn.StartDat.YEAR - QTHComp.QHComp[LTEmpLen1].DVage) + 16) – **condition text** Are you sure? Started self-employed business when less than 16 seems a bit young. – **involved fields** + QTEmpLen.QEmpLen.SEmpStY – QSignIn.StartDat – **descriptives** field name: QSignIn field kind: DataFie field type: Date – **specifications** attributes: dontkn tag defined: field size 8 – **question** Enter date interview with this household was started – **description** – **answers** Date + QTHComp.QHComp.DVage – **conditions** 1 <= LTEmpLen1 <= DMHSIZE LDMINDINTERVIEW[LTEmpLen1] = 'now' QTILO.QILO[LTEmpLen1].DVILO4a = InEmpXuf SEmpStY = RESPONSE **34.1.2.1.1.1.1.5. TOGGLE CHECK CHECK** – **statement** CHECK – **involved fields** – **conditions** 1 <= LTEmpLen1 <= DMHSIZE LDMINDINTERVIEW[LTEmpLen1] = 'now' QTILO.QILO[LTEmpLen1].DVILO4a = InEmpXuf – **navigation** parent step over	
Computations	Like signals and checks, computations are displayed on screen in the rules section at block level. Details about the computed variable are displayed separately. It is not possible to choose whether to display the computations.	✓ ✗

5. SUBQUESTIONNAIRE QNames

- field definition
  - descriptives

field name:	QNames
field kind:	DataField
field type:	BNames (Block)

  - specifications

parallell block:	false
embedded block:	false
field size	122

- statement

QNames.ASK

- rules

```
LNames2 := 0
FOR LNames1:=1 TO 10 DO
    QBNames[LNames1].ASK
    IF QBNames[LNames1] <> EMPTY THEN
        LNames2 := LNames2 + 1
    ENDIF
ENDDO
DVHsize := LNames2
DVHsize.KEEP
```

- involved fields
- conditions
- navigation

parent
step into
step over

5.4. QUESTION DVHsize

- field definition
  - descriptives

field name:	QNames.DVHsize
field kind:	DataField
field type:	integer

  - specifications

attributes:	empty dontknow refusal
tag defined:	
field size	2

  - question
  - description
  - answers

number : [1..10]

- statement

DVHsize.KEEP

- involved fields
- conditions
- navigation

parent
step over

How questions appear on screen	Delta does display the format of the question text, instructions and answer categories as they appear on screen. So for example if question instructions are displayed on screen in different colour to the question text this can be seen on screen. However, screen colours and layout are not displayed on screen. **27.11. QUESTION Drier** – **field definition** – **descriptives** field name: QDurable.Drier field kind: DataField field type: YN (Enumeration) – **specifications** attributes: dontknow refusal tag defined: field size 1 – **question** Does your household have any of the following items in your (part of the) accommodation... ...Tumble drier? If combined washing machine and tumble drier, code 1 for both – **description** Tumble drier – **answers** name | code | text Yes | 1 | No | 2 |	✓ ✗

Recording Interview Sound Bites Through Blaise Instruments

M. Rita Thissen and Gilbert Rodriguez, RTI International

1. Introduction

In any data collection effort, survey managers need to be concerned about the conduct of the interviewers with respect to following protocol and the possibility of falsification.[1] In the past, field operations have been at a disadvantage compared to telephone surveys due to the inability of field supervisors to perform silent monitoring during interviews. Now that mobile computing devices (MCD's) such as notebook or handheld computers offer digital sound-recording capabilities, it becomes practical to record sound bites from field interviews for later review, providing an audio sample of the actual interchange between field interviewer and respondent.

Because the recording captures the verbal exchange between interviewer and respondent, it provides an auditory snapshot of the situation at the time of the interview. At any time afterwards, a monitor or supervisor can review the recording and rate the interviewer's performance, effectiveness of the item, and likelihood of authenticity of the interview. The review can take place shortly after completion of the interview and transfer of case data to the central site, providing a rapid mechanism for verification and feedback. Concerns about falsification of response data, instrument usability or interviewers' lack of adherence to protocol can be addressed effectively by this unobtrusive approach.[2]

The technique, known as computer-assisted recorded interviewing (CARI), involves hardware, software and work processes:

- Field computers or MCD's are equipped with internal or external microphones.
- Survey specifications note items for recording.
- Blaise programs include audio recording commands.
- If randomized recording is desired, that logic is added to the questionnaire.
- The data transfer system transports the audio files along with the case data, or some alternative method is devised to capture and manage the sound files.
- Monitors are trained to review the recordings and assign ratings.
- Tracking and report programs enable central staff to manage the CARI review process and workload for monitors.
- Potentially falsified cases are referred for more intense investigation through call-back or field re-interviewing.

At the heart of the process is the questionnaire. The survey instrument controls the sound recording, with procedure calls to generate the audio files.

In this paper, we discuss the basic issues involved in implementing CARI through a Blaise questionnaire, including decisions about CARI uses, item selection and hardware. Audio recording is a powerful new tool for capturing information that cannot be obtained through other methods, valuable for quality assurance, field staff management and questionnaire evaluation.

2. Benefits of Using CARI to Verify Interview Authenticity

Although there are several advantages offered by instituting CARI, possibly the most compelling reason is for confirming the authenticity of data. CARI can act as a deterrent to curbstoning and

as a tool for detecting questionable interviews. Interviewers who are aware that monitors may listen to parts of each interview are less likely to falsify data, because the audio file acts as a "witness" to their actions. In this way, the simple presence of CARI can reduce cheating.

Audio files are generated in the field, but sent to a centralized team of monitors for objective review. When monitors review the CARI recordings, they categorize the interview as likely to be authentic, questionable, or probably falsified. Questionable files can be referred to a second monitor for review, or the cases can be routed to a traditional verification process such as re-interview or callback.

Speech patterns heard in audio files provide information about the veracity of the interview, as indicated by the timing and phrasing of questions and responses. In a normal interchange, people pause between words, phrases or sentences, as they consider their answers or express their views. [3, 4] CARI monitors quickly become adept at distinguishing between recordings of normal interviewing circumstances and suspicious ones.

If the interviewer acts alone when falsifying data, there may be no voices at all in the recording or only that of the interviewer, without the expected pausing, inflection or clarity of voice to have been part of a two-way exchange. If the interviewer enlists an accomplice to pose as the respondent, the accomplice may display inappropriate attitudes or emotions, may make unexpected remarks, may respond without pausing to understand the question or may pause at unnatural places while inventing an answer. Figure 1 lists some possible indicators of questionable authenticity, which may be noted through CARI review. Monitors can assign a rating for the perceived authenticity of the interview, based on hearing all available sound bites from that case, and noting any technical problems with the recording quality.

Figure 1. CARI monitors listen to the recordings, expecting to hear two distinct voices – the interviewer and the respondent.

CARI Indicators of possible falsification

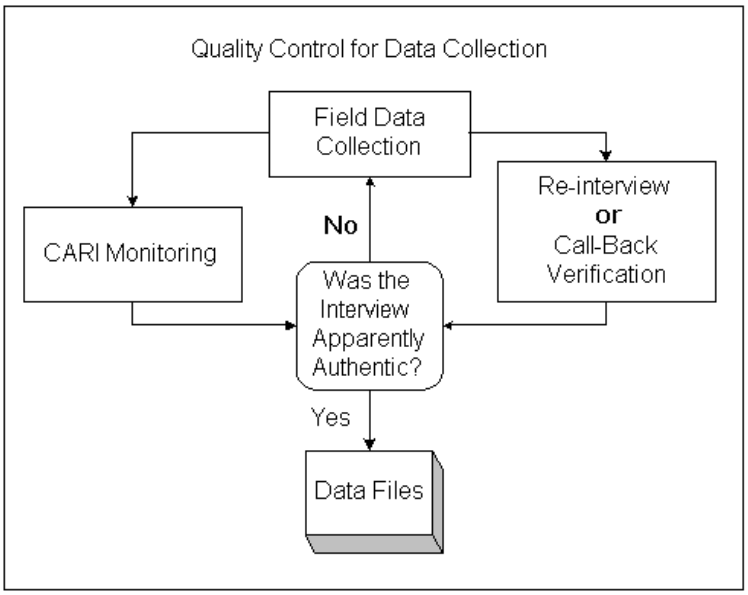
- No voices can be heard, although room noises are audible.
- The interviewer can be heard, but appears to be speaking to himself or herself.
- The respondent answers too quickly or laughs in inappropriate places.
- The respondent makes comments suggesting the interview is being falsified.
- The same respondent's voice is heard in recordings of multiple interviews.

By using CARI, a survey organization may be able to reduce its re-contact and re-interview effort substantially, though not entirely. CARI monitoring can be used in place of most telephone verification calls or field re-interview, if implemented as a means of deterring falsification. However, it remains important to have a second means for following up, since some respondents may refuse to allow audio recording. Also, an alternative method may also be needed for gathering evidence when falsification is indicated by the CARI files, in which case it is desirable to send the case back to the field, as indicated in Figure 2. The number of cases which need to be refielded or re-contacted for these reasons is small.

In addition, it is good practice to route a percentage of all cases to traditional verification alongside CARI monitoring. The benefits of this are twofold: first, it tells the interviewing staff that they cannot avoid monitoring, even if they discourage their respondents from allowing CARI, and second, it allows comparison of the results of the two approaches to confirm the validity of the monitoring processes. The diagram in figure 2 shows the two systems working in

parallel, but the cases flowing through the CARI process would substantially outnumber the cases flowing through traditional verification.

Figure 2. CARI monitoring can replace much of the traditional effort of verifying interview authenticity.



By conducting the majority of authenticity monitoring through CARI, the effort is reduced compared with traditional methods because of the centralized nature of the monitoring process and the greatly reduced need for re-contact with respondents. Thus, adding CARI processes reduces the burden on interviewing staff and respondents alike.

3. CARI for Enhancing Interviewer Performance

Field staff do not always conduct interviews in an optimal manner, and it is difficult to obtain reliable information about their performance. Personal observation provides a wealth of information, but the presence of an observer may bias the evaluation by affecting the behavior of the interviewer or respondent. CARI offers a unique opportunity to listen to the interview exactly as it took place, without observation effects. From the sound bites, a monitor may be able to provide feedback, either as praise or constructive criticism, about the way in which the interview was conducted. If used in the first few weeks in the field, feedback can be an important tool for reinforcing lessons learned during training. Figure 3 indicates the some types of behavior that are amenable to review through CARI monitoring.

Figure 3. Review of CARI files can provide insight into field staff performance.

Indicators of improper questionnaire administration
- Paraphrasing
- Improper probing
- Suggesting responses
- Poor enunciation
- Inappropriate commentary

Behaviors to reinforce
- Precise adherence to protocol
- Adept handling of difficult situations
- Consistency, honesty and professional behavior

4. CARI for Evaluating Questionnaire Items

CARI can also be used to evaluate the usability of questionnaire items. The audio recording of an interviewer's presentation of an item and the subject's response provides a clear indication of whether the item succeeds in several ways:

- readability – based on the interviewer's fluency in presenting the item
- clarity of content – based on the respondent's ease of understanding

Items which evoke negative reactions or require frequent explanations are detrimental to the response rate and increase the level of burden on both interviewer and subject. Using CARI, especially during field testing of an instrument, allows the survey specialist to evaluate the success of the questionnaire items in eliciting the desired information.

5. Selecting Questionnaire Items to Record

How much of a questionnaire to record depends on the purpose for which the audio files will be used. If the goal is verifying authenticity, a few brief sound bites will suffice.[5] If the purpose is monitoring staff performance, providing feedback on techniques or evaluating the usability of specific items or sections, longer recordings may be needed. For some purposes, the entire interview may need to be recorded, perhaps entire sections will be needed, or it may be desirable to record many individual items.

When selecting items to record, the mechanism of transmitting the files from the field to the monitoring site also must be considered. Audio files tend to be large, and if the interviewer uses a low bandwidth channel (such as 56K baud dial-up modem), it is important to limit the number of kilobytes to transmit. For long recordings, an external removable storage medium with high capacity may be necessary, such as a Zip disk, compact disk or memory stick. These removable data stores can be shipped or carried to a central site. Figure 4 lists some of the uses of CARI, the expected audio file quantity, and appropriate transmission methods.

Figure 4. Item selection for CARI must take the transmission method into account, if dial-up lines are used.

Purpose of Recording	Number of Items, Length of Recording	Transmission by Dial-Up	Transmission by Broadband or Removable Storage
Authenticity check	Few, short	Yes	Yes
Interviewer conduct	Few, short	Yes	Yes
Interviewer feedback	Many, long	No	Yes
Selected item usability	Few, short	Yes	Yes
Section usability	Few or many, long	No	Yes
Whole interview	Many, long	No	Yes

After the quantity of recording has been determined, there remains a choice of which items to record. For item usability, the choice is clear -- only those items that need review should be recorded, along with a set of comparable items to act as a control sample. At the other extreme, if the entire interview is to be recorded, or specific sections, CARI will be activated at the start of the interview or section and continued to the end (or to breakoff), at which point the instrument deactivates the recording process.

For monitoring authenticity and/or interviewer conduct, however, there are more options to consider, especially when trying to limit the overall length of recording for transmission by dial-up line. One might choose three items from the beginning, middle and end of the questionnaire. Alternatively, it might be best to choose a few items of high importance or high likelihood of curbstoning. One can cover both bases by selecting an important item from the beginning (such as asking the respondent if they have read, understood and agreed to the informed consent), one from the middle (such as a gateway question for an important section), and one from the end (such as asking the respondent if they have received the specified incentive). With this combination, one can check authenticity of voices and confirm important responses spanning the interview session.

Sometimes it may be useful to have additional items recorded for a fraction of the cases instead of all of them. This can be accomplished by generating a random number before the item, and then checking to see if it falls within a certain range. If the item is selected in this manner, CARI is activated, but if not, the item is not recorded.

6. Selecting Cases to Record

Although it may be too time-consuming to review CARI recordings from all interviews, there are advantages to collecting audio files whenever the respondent agrees. First, for CARI to act as an effective deterrent, the interviewer must not know which cases will be reviewed; each interview must seem to be as likely to receive verification as any other one. Second, if CARI monitoring raises suspicions about an interviewer's honesty or professional behavior, recordings from additional interviews (or from all interviews) may need review. Finally, if all interviews are recorded, selection of which ones to monitor can be made by the monitoring staff or by a post-data-collection sampling method, providing additional flexibility to the survey staff.

7. Technical System Overview

The entire process of CARI includes sound recording on the laptop or MCD, file transmission and subsequent playback by CARI monitors. Blaise statements within the instrument invoke a dynamic link library (dll), which in turn activates or deactivates software to run the computer's sound recorder. The laptop case management system or a custom application packages the audio files for transfer to a central monitoring location, as shown in Figure 5. At the central site, monitoring staff play the audio files and rate the interview's authenticity, the interviewer's performance and/or the questionnaire's usability, as shown in Figure 6. The monitor's ratings determine subsequent acceptance of the data or remedial action.

Figure 5. Audio files are recorded during the interview, by implementing calls from Blaise to a custom dll which activates the laptop's sound recorder.

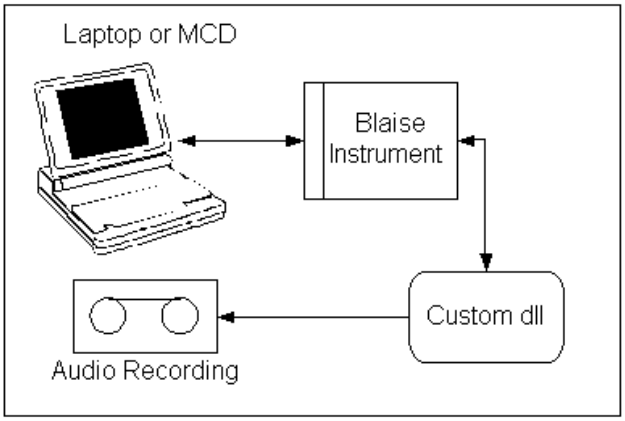
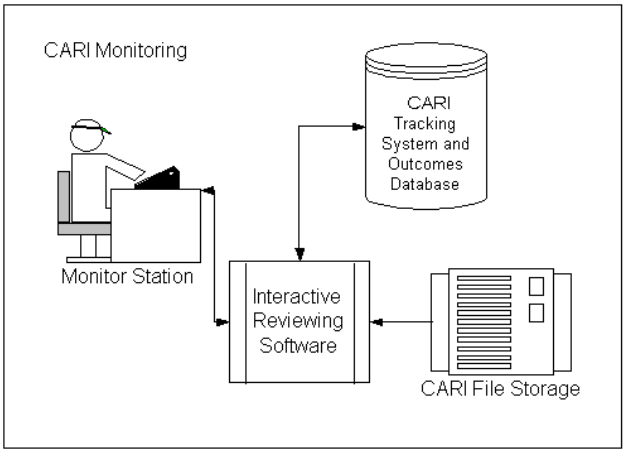


Figure 6. After the audio files are transmitted to a monitoring site, CARI monitors listen to the sound bites and evaluate the authenticity of the case and/or the interviewer's performance.



8. Programming CARI Calls Through Blaise

There are several steps to take when setting up a new survey for CARI. The specifications detail which items or sections will be recorded, and instrumentation must be programmed with appropriate logic and control structures to capture the designated portions of the interview. Several audio files may be generated for each respondent, and so the files should be named in a systematic manner to identify the case and item or section. Adding a timestamp to the filename can prevent confusion, if the respondent backs up and then returns to the same item a second time. In the example below, the letters "A" and "B" are appended, as a simple example of how to distinguish among recordings for a specific case.

Coding a Blaise instrument to perform CARI is much like coding any instrument with a call to any external program, except that the CARI instrument should include a revocable consent flag, and certain items will be marked for audio recording. Figures 7 - 12, when taken together, provide a complete example of a CARI-enabled instrument. For presentation here, this demo instrument is divided into several figures to call attention to specific features in the code.

Most studies will include a CARI consent screen as part of the instrument, just as telephone interviewing generally has a statement that the call may be monitored. In the example below, the consent screen is programmed as a parallel block, so that it can be presented as the first item and also accessed at any time during the interview, through a hot key or menu item.

Figure 7. By making a parallel block for the CARI consent flag, the respondent can consent to recording or withdraw consent as often as required.

```
DATAMODEL CARIExample

{ CONSENT is a routed parallel block; it is asked at the outset but can also be jumped to via
menu command or hot key }
PARALLEL CONSENT

PRIMARY QID

TYPE
STR6 = STRING[6]
```

Figure 8. Calling the StartStopRecording procedure activates or deactivates the sound recorder, through a call to external software.

```
PROCEDURE StartStopRecording
 {Starts and stops audio recording
 IMPORT PARAMETERS
  X -- either 1 (start) or 0 (stop)
  Y -- if starting a recording, then duration of recording in secs; set equal to 0 if no time limit.
  QID -- string used by recorder.dll to name the resulting audio file, part of which is the case ID
 EXPORT PARAMETER
  ERRORFLAG -- indicates if a problem occurred }
 PARAMETERS
 IMPORT
   X, Y: Integer
   QID: STR6
```

```
 EXPORT
  ErrorFlag : Integer
  ALIEN('Recorder.DLL', 1)    { Delphi DLL which calls an application that starts/stops
recording of a wav file }
ENDPROCEDURE
```

Two pieces of software are necessary in addition to the Blaise code, to enable CARI questionnaire administration. First, an application is needed which will call the laptop or MCD's sound recording system, activating it for the duration specified in a parameter, or turning it on or off explicitly. Second, a custom dll is required to call the sound recording application. Figure 8 above shows how the instrument calls the dll, which in turn calls the sound recording software.

The following lines of code present the consent question. Note that any time the consent is withdrawn or refused, an explicit call to the StartStopRecording procedure cancels any recording which had been started. This precaution is necessary, since the respondent may initially consent, but later decide to refuse. If that refusal takes place within a recorded section, it is important to stop recording at that time, rather than waiting for the next preprogrammed end to CARI.

Figure 9. Informed consent is an important part of the CARI process.

```
{ CONSENT is a routed parallel block; it is asked at the outset but can also be jumped to via
menu command or hot key }
{ bCONSENT contains one question, a consent question about recording }
BLOCK bCONSENT
   FIELDS
   Q1 "@/Do you consent to being recorded?" : (yes "Yes", no "No")
   RULES
   Q1
   { If consent is not given or is revoked, be sure to stop any recording in progress }
   IF Q1 = NO THEN
       CONSENTFLAG := no
       StartStopRecording(0,0,'DUMMY',ErrorField)
   ELSEIF Q1 = YES THEN
       CONSENTFLAG := yes
   ENDIF
ENDBLOCK
```

The demo instrument shown here contains two calls to CARI. The purpose of the demo is to exhibit the capability of the CARI timer to record from one point in the rules to another or optionally to stop after a specified time interval, as well as to show how to embed CARI calls within the instrument. Because of those goals, the text of the items tells the user when CARI is being activated and terminated. In a normal interview, neither the interviewer nor the respondent would be aware of which segments of the interview were being recorded.

Figure 10. This section of the demo presents the Blaise field definitions and additional information to be stored.

```
FIELDS

  QID "Case ID": STRING[5]
  CONSENTFLAG : (yes "Yes", no "No")

  StartRecord "Start Time" : STRING[30]
  EndRecord "Stop Time" : STRING[30]
  StartRecord2 "Start Time" : STRING[30]
  EndRecord2 "Stop Time" : STRING[30]

  INTRO "@/Case ID : @B^QID@B
  @/@/Welcome to the Blaise CARI demo instrument!  In a moment you will be asked to read a
passage aloud. Press [ENTER] to continue." : string[1], empty
  ITEM1 "@/@/Type 1 and then press [ENTER] when you are ready to begin." : 1..1
  ITEM2  "@/Please read the following aloud:
  @/@/@B For the purposes of this demo, you are aware that you are being recorded, so that you
can see that the timer feature of the software is functioning.  This recording has been given
unlimited time, and it will continue recording for as long as the item remains on screen.  In a
normal interview, the respondent would not be told about the occurrence of audio recording, and
therefore would not be aware of which items were recorded and which were not.@B
  @/@/When you are finished, type 1 and then press [ENTER].  This recording will terminate
when you press the [ENTER] key."   : 1..1
  ITEM3   "@/Great!  In a moment you will be asked to read another passage aloud.@/@/Type 1
and then press [ENTER] when you are ready to begin recording." : 1..1
  ITEM4    "@/Please read the following aloud:
  @/@/@B This item will be recorded for 30 seconds, regardless of how long the item remains
on screen.  As stated before, if this were an actual questionnaire, neither the interviewer nor the
respondent would not know whether recording was taking place.@B
  @/@/When you are finished, type 1 and then press [ENTER].  Otherwise, the recording will
terminate after 30 seconds."   : 1..1
  THANKS "@/Thank you for your cooperation.  @/@/Type 1 and then press [ENTER]  to exit."
: 1..1
  ErrorField "Error flag": 0..4

  QIDA "CASEID + A" : STRING[6]
  QIDB "CASEID + B" : STRING[6]
  CONSENT : bCONSENT
```

Item one simply begins the interview. Item Two is the first CARI recording, and it is set up to record as long as the item remains active on screen. In this demo, recording is stopped when the user advances past Item Two, but the audio recording could have continued for several additional items or even for the entire interview. Item Four hosts a second call to StartStopRecording, with a time parameter of 30 seconds. Because of this, the audio recording will cease at 30 seconds after it starts, regardless of whether the respondent has finished. An explicit call to terminate the recording could be set at the start of Item Five, if the need was to have CARI for a maximum of 30 seconds, but only include the single item for which it was called (Item Four). Instrument fields can make note of the length of the sound bite by storing the start and stop times, to track the actual duration of each sound bite.

Figure 11. This section of the Blaise rules presents the first item, which is allowed unlimited recording time.

```
RULES

{ keep values so that they'll be there when resuming a breakoff }
QID.KEEP

StartRecord.Keep
EndRecord.Keep
StartRecord2.Keep
EndRecord2.Keep

QIDA.KEEP
QIDB.KEEP
QIDA := QID +'A'
QIDB := QID + 'B'

INTRO

CONSENT

ITEM1      { start recording after this field has been answered }

  if CONSENTFLAG = YES THEN
     if STARTRECORD = EMPTY and ITEM1 <> EMPTY then
        STARTRECORD := Str(Hour(systime)) + Str(Minute(systime)) + Str(Second(systime))
        {Activate CARI with no end time specified and then de-activate CARI at the start of the
next
         item, to terminate recording that item}
        StartStopRecording(1,0,QIDA,ErrorField)
   endif
  endif

ITEM2     { Stop recording after this field has been answered }

if CONSENTFLAG = YES THEN
   if ENDRECORD = EMPTY and ITEM2 <> EMPTY then
      ENDRECORD := Str(Hour(systime)) + Str(Minute(systime)) + Str(Second(systime))
      StartStopRecording(0,0,QIDA,ErrorField)
   endif
endif
```

Figure 12. This section of the Blaise rules illustrates a timed call to CARI software and closing out the questionnaire.

```
ITEM3  { start recording after this field has been answered }

if CONSENTFLAG = YES THEN
   if STARTRECORD2 = EMPTY and ITEM3 <> EMPTY then
      STARTRECORD2 := Str(Hour(systime)) + Str(Minute(systime)) + Str(Second(systime))
   {Activate CARI for 30 seconds }
      StartStopRecording(1,30,QIDB,ErrorField)
   endif
endif

ITEM4

THANKS
{ stop recording after this field has been answered, since we are about to exit the instrument }
if CONSENTFLAG = YES THEN
   if ENDRECORD2 = EMPTY and THANKS <> EMPTY then
      ENDRECORD2 := Str(Hour(systime)) + Str(Minute(systime)) + Str(Second(systime))
      StartStopRecording(0,0,QIDB,ErrorField)
   endif
endif

ENDMODEL
```

9. Potential Issues

Technical barriers or difficulties may arise with CARI, as with any other form of technology. In addition to technical issues, there may be field management or human factors issues. Some of the common technical and staffing problems are listed in Figure 13, with an indication of how to address them.

There may be confidentiality, privacy and legal concerns as well. In most places, CARI recordings are treated the same as telephone monitoring. Specific localities may have restrictions on the allowable length of time for retaining the audio files, association of identifying information with the files, requirement for encryption during transmission, or other concerns related to privacy. For a survey to adhere to the strictest restraints, the following guidelines should be enforced:

- All respondents should explicitly consent to the recording at the start of the questionnaire and should have an option of retracting that consent at any time.
- Files should not contain personal information, such as recording the respondent's name or sensitive information, and the file naming convention should use an assigned ID number or other method to affiliate the sound bites with the questionnaire, without revealing the identity of the respondent.
- Audio files should be encrypted before transmission and stored in a secure place at the central and reviewing sites.

These considerations must be weighed carefully in developing a CARI process.

Figure 13. Potential problems that may arise when setting up a CARI system.

Problem	**Type**	**Resolution**
Audio quality is poor on playback	Technical	Evaluate microphone and laptop volume control settings, and check laptop sound card. If external microphone, check plug. Check playback settings. Use headphones during monitoring. Select alternative playback software.
File storage space	Technical	Allow sufficient disk space for audio file storage, on the MCD and in the central site. Significant space may be needed, depending on the file format used, number of interviews and length of storage. Select a compression method or alternate format if files are too large to handle.
Transmission failure	Technical	Check file size and transmission bandwidth. Check case management or file control system.
Matching audio files to case data	Technical	Use a naming convention for CARI files that includes a case ID and sequence number or item name.
CARI is detectable during the interview	Technical	Select recording algorithm that does not strain system performance. Postpone compression and encryption if necessary.
High respondent refusal rate for CARI	Human	Train interviewers or reword consent question. Monitor refusal rates by interviewer, and use an alternate form of verification for those with high rates (re-interview or call-back).
Interviewer dissatisfaction	Human	Provide a benefit such as positive feedback from CARI. Emphasize CARI as a tool for rewarding honesty, as opposed to a tool for catching cheaters.
Revocation of respondent consent	Technical and Human	Provide a hot key that changes the consent variable from yes to no. Check the flag at each potential call to CARI.
Location and selection of monitoring staff	Technical and Human	Select file storage and playback interface for ready access by monitoring staff. Consider possible bias among monitors: field supervisors may not be fully objective.
Privacy concerns	Technical and Human	See notes above.

10. Future Opportunities

Use of CARI will become commonplace as benefits are recognized by the survey research community. Once the initial introductory decisions and development effort have been completed, survey shops may notice a reduction in reinterview effort, reduced reliance on respondent phone numbers and lower respondent burdens due to receiving fewer call-backs. As technology advances, CARI may move from laptops to additional platforms, such as handheld computing devices or pen-based computers. The Blaise language offers good control of the process of recording sound, making CARI a viable tool for improving field performance and authenticity.

11. References

1. P.P. Biemer, and L. Lyberg,, *Introduction to Survey Quality*, New York: John Wiley & Sons, pp. 149-187, 2003

2. P.P. Biemer, Deborah Hergert, Jeremy Morton, and Gordon Willis, "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, 2000

3. S. Kowal, D.C. O'Connell & E.J. Sabin, "Development of temporal patterning and vocal hesitation in spontaneous narratives", *Journal of Psycholinguistic Research,* Vol. 4, p. 195-207, 1975

4. D.C. O'Connell. & S. Kowal, "Pausology". In *Computers in Language Research*, W. A. Sedelow, Jr. & S. Yeates Sedelow, Editors, pp. 221-301. Berlin-New York: Walter de Gruyter & Co, 1983

Acknowledgements

The authors would like to recognize the contributions of the following people at RTI:

Phil Cooley, R. Suresh and Al Bethke: System design and development team
Pauline Robinson: Application programmer
Paul P. Biemer: Research and development in CARI methodology
Staff of the Research Computing Division and the Survey Research Division

We also appreciate the support of organizations who have sponsored parts of our work with CARI techniques:

U.S. Bureau of the Census (http://www.census.gov)
U.S. Department of Health and Human Services, Administration for Children and Families (http://www.acf.dhhs.gov)
RTI International (http://www.rti.org)

Deploying Blaise to Tablet PCs for Mobile Use

David Hill, Senior Systems Analyst, Westat, USA

1. Introduction

In tablet PC computers, clipboard ergonomics and form factor may be returning to field interviewing in a smarter, more flexible device. There is no mistaking that a century of design and implementation development in the survey industry led to the preponderant use of hardcopy booklets on clipboards for field data collection. Using a pen to mark structured paper forms on the flat surface of the clipboard was preferred, an optimal combination of mobility, flexibility and effectiveness. Throughout the 20th century, the portable typewriter existed and could have been used for field interviewing but was not. From a usability perspective, portable typewriters strongly resemble laptop computers of today. Yet laptop computers are pervasive in current field computer assisted interviewing (CAI).

So far in the marriage of computer assistance and interviewing, keyboard devices have dominated due to their early development, consistent presence in available hardware platforms yielding software tuned to this as the input device and a history of user familiarity with keyboards. A long chain of evolving computing devices and uses relies on keyboards for input. First, keyboards predate computers on typewriters and this led to their adoption. Evidence lies in the continued presence of the QWERTY standard layout that originated on typewriters. For computers, the first major innovation from manually punched cards or operated punch devices was keyboard-based cardpunch machines. These machines strongly resembled typewriters (except they punched cards instead of impressed inked type on paper) so migrating the keyboard interface was obvious. When eliminating the cards evolved, computers took their input directly from keyboards, mostly in command form. Next multi-processing supporting many keyboard based terminals occurred. Then, CPUs became individualized in personal computers (PCs) while the "killer applications" of word processing and spreadsheets arose. Since both of these applications are text or digit based, keyboard input was emphasized in early PCs. Yet, as more diverse PC utilization, more applications, and advances in visual representations of data matured, the graphical user interface (GUI) won out. Fundamental to the GUI interface is interacting with the displayed graphics. To do so requires a pointing device. The mouse proved technologically practical and was embraced for this purpose. Thus, a keyboard and mouse became the standard PC interface. This determined that when PCs became small enough to be portable, a keyboard and pointing device was embraced as the main hardware interface. Portable computers quickly standardized on the clamshell laptop configuration fifteen or more years ago and have not since changed substantially, in this regard.

Laptops support general business portable computer usage well, no doubt determining the lack of continued configuration evolution. Plus, obvious interface alternatives, such as handwriting or voice recognition and non-manual pointing devices continue to be costly and suffer in reliability. However, while laptops support portable use, they do not support mobile use. That is, while laptops can be taken many places and set down--along with the user--to work, one cannot use them effectively while moving or standing. This is the essential difference between portable and mobile uses. Effectively using a keyboard-based laptop requires two hands. One cannot accomplish this while devoting one hand to hold the machine. A laptop must be set down on an appropriate working surface, since assorted attempted harnesses or stands for carrying laptop computers for mobile operation have proven unsatisfactory.

To date, the availability and cost points of laptop computers have determined that the field survey industry largely conducts CAI operations as portable implementations. Yet general field interviewing is not inherently a portable task. It is a mobile task. Or at least, if supported as a mobile task, it can also support portable implementations, while the converse is not true. This is the usability lesson we should inherit from the preference for the clipboard form factor of paper questionnaire administration.

Since many interviewing tasks involve moving, standing, or further set-up variations that do not accommodate laptop set-up, reliance on laptops for computer-assistance is a dead-end for mobility. Therefore, researchers designing survey projects tend to coerce the field environment to accommodate laptop use or abandon utilizing them. They must shoehorn their CAI use into portable (laptop) computer operations, without a practical mobile alternative. If portable set-up cannot be accomplished in the expected interview encounter environment, that choice eliminates the many advantages of CAI data collection. Field interviewing and operations software can fully support many mobile CAI tasks, yet a hardware platform to carry these applications has been absent.

Tablet PCs fill this prior mobile CAI device void and their flexibility, after overcoming other resistance points, may dominate most field applications. Just as the clipboard form factor dominated the survey world of paper form administration, the tablet PC is set to be the hardware platform of preference for field interviewers conducting non-trivial CAI.

The remainder of this paper contains the following sections. Section 2 discusses tablet PCs generally, focusing on distinctive software considerations and usage opportunities. Section 3 describes desirable elements of application interfaces for tablet PC operation. Section 4 discusses the utilization of Blaise on tablet PCs. This includes a review of strategies to deploy Blaise programs on tablet PCs and recommended modifications to Blaise software that would make it friendlier to tablet operation.

2. Tablet PCs: What are they? Why now?

Tablet PCs resemble hardcopy clipboards. They are slate style mobile computers. They use a flat display meant to be pointed at or written upon. The primary interface device is a stylus, essentially an electronic pen. As a pointing device, one points directly at the screen objects and touches the tip of the pen to the screen (taps) to select (the equivalent of mouse clicks). As a writing or drawing device, the tip is placed on the display surface and moved just like writing with an ink pen upon paper. Of course, electronic pens do not write tangible ink on the display surface but utilize virtual electronic inking that flows as a line on the graphics display. Attributes of electronic ink, such as color or thickness, are controlled by software setup.

While a number of previous generations of this form factor have been attempted (and generally called “pentop” computers), none of those attempts have achieved general commercial success other than spawning the next generation of devices. The difference in this device cycle is that the biggest organization in the PC industry is now embracing these devices. Microsoft has formed and proselytized a standard for this platform including hardware, performance and compatibility aspects that enable the machines to run the Windows XP operating system and achieve reliable, or at least predictable, operation of Windows programs.

Microsoft’s promulgation of a tablet standard and assurance that the Windows operating system functions on the devices made a new market for pen/slate computers. Numerous mainstream

hardware manufacturers (e.g. Compaq/HP, Toshiba, Gateway, plus others) now build devices[1] to this standard. This means slate style mobile PCs are no longer specialty hardware devices. Now they are commodity devices, another flavor of field-able PC computers suited to mobile tasks. A price premium[2] for tablet PCs exists over the more ubiquitous laptop models. This is a resistance point to general adoption.

For that cost difference, users buy practical mobile usability, in addition to portability. You can hold this slate form factor and use it while standing or moving. One hand supports the machine; the other operates it. You can easily set the pen down (or just drop it when the pen is tethered to the computer) and use the free hand for other purposes too. You can also set a tablet PC down on a surface, using it like a writing pad. Since the slate machine is flat, it does not have awkward balance and vantage point constraints as such laptop use. Also, when a tablet PC is set down for use, a now unencumbered second hand is free for some other action. Variously, one can prop a tablet PC display and use an auxiliary keyboard in a setup indistinguishable from a laptop computer (such dual use is the very design crux of the convertible form factor of some tablet PC models). In all, a tablet PC is more flexible than a laptop. It enables mobile use and supports a greater range of portable use too.

2.1 Software Considerations

Because tablet PCs operate a standard form of the Windows XP operating system, any application that runs on this operating system will run on tablet computers. This includes Blaise interviewing software. However, just running PC software is not the entire objective. The main reason to use a tablet PC is its mobile or more flexible ergonomic potential. Relying on keyboard input squanders the greatest strength and primary reason to use this platform. The pen interface is the obvious and practical mode of mobile and flexible input. This exerts presentation and interaction imperatives for applications intended for such mobile and pen/slate interface. These are alternatives to the historic keyboard input focus of traditional PC applications, including Blaise. Lack of such mobile tuning in software is another resistance point to general tablet PC adoption.

For pen focused operation, the crux software considerations are 1) the pen is most effective as a pointing device and the main desirable interface device for mobile or flexible machine use, and 2) what users point at is the direct rendering of the output on their screen. Neither of these is true

[1] The tablet market has split ostensibly into two, overlapping, machine configurations: pure slate and convertible form factors. Pure slate tablets look like electronic clipboards. They rely on pen input. You can plug auxiliary keyboards into them to achieve a laptop-like configuration. Convertible tablets are hybrid PCs that generally have a rotating hinge attaching a keyboard. In one hinge configuration they resemble laptops, although generally continue to rely on their pens as the built-in pointing device. Their rotating hinges allow the keyboard to flip around and lay flat against the screen portion of the machine. This yields a slate-like configuration (albeit with a thicker profile) where pen-only operation is the norm. The remainder of this discussion refers to slate-like use of the tablets, whether pure slate or convertible. This means pen interfacing, without the benefit, or variously the encumbrance, of a tangible keyboard.

[2] While a large price reduction in tablet PCs--over prior pentop PC models--occurred in this latest cycle, they remain more expensive than laptops. As relatively newly engineered devices, the full effects of mastering manufacturing specialization in tablet PCs, economies of scale and manufacturing cost-amortization among the producers are not yet fully expressed in unit prices. Plus, the presence of multiple producers making machines to the same standard creates market competition that is already exerting price reduction pressure and will continue to do so. This market effect did not exist previously for this class of machine, when each was a specialty device. Yet, the properties of tablet PCs determine that some price premium for such machines will continue since producing their pen-sensitive screens is simply more costly. In time, we expect this difference to settle under two hundred dollars, rather than the couple hundred dollars of the current market.

for keyboard operation and while a mouse is a pointing device, it does not directly point[3] at the target objects. In pen operation, the output surface for user interface is literally the input surface too. This triggers a number of immediate effects related to human skills. For example, since the pointing is direct, the accuracy of that pointing is limited by the ability of human users to accurately point at the target object. In practice, this proves less accurate than mouse pointing. Plus, there is no opportunity to adjust the sensitivity of motion, since there is no indirection in the portrayal of the movements. In a tuned pen interface, this suggests that target objects should be larger for pen pointing. This suggestion coincides with another factor on tablet PCs. Mobile users are typically less able to focus just on their computer and program interface. Mobile use is often required because the users are engaged in a further task(s) at the same time or are positioned in a less ideal, although more flexible, posture. This places more emphasis on clarity of layout and interaction. One element of that clarity can be more conspicuous objects, by being larger.

More detailed software considerations also appear in the tuning of application interfaces for tablet PCs. One area where keyboards excel is open text entry. Possibly the best means to address this in tablet operations is designing the instrument to avoid non-essential open text fields. This design objective is shared with web applications where processing and transmittal of open text data is less desirable in web browser/server architecture than immediately interpretable actions. Still, invariably in interview instruments, some need for open text entry arises and effective and well integrated alternatives are needed. A body of creative techniques is emerging as a richer set of pseudo-standard Windows objects that service open text entry without a tangible keyboard. Fully pen tuned applications should integrate these objects to advantage.

The screen of a tablet computer is its largest tangible hardware feature. The outside dimensions and shape of tablet PCs correspond to the screen display. Those dimensions now proportionally mirror conventional PC displays. That is, one dimension (typically horizontal) is longer than the second dimension (typically vertical). Unlike traditional PC monitors, tablet PCs can be rotated. In practice, holding tablet PCs by their narrower dimension is easier ergonomically. The Windows XP tablet compliant operating system supports this display rotation very well at a BIOS and OS level[4]. But this does not automatically solve application screen layout issues. For the application software, this hardware rotation converts the display screen from the traditional landscape orientation (horizontal wider than vertical) to a portrait orientation (the reverse). This is a familiar issue since printing output to a standard page of paper carries a corresponding dimension switch. Still, in tablet operation the issue is deeper because the rendering is not only an output format but must support input interaction too. Printed pages are only an output rendering, such that simple pagination of printed pages (both horizontally and more frequently vertically) often suffices for this dimension switch. The screen equivalent is scrolling and fully supportable in Windows programs, but generally not an optimal solution for sparse input

[3] Notwithstanding some early GUI advocates assertions so. I believe the assertion of mice as direct pointing devices was an effusive description to emphasize the utility of pointing at active objects introduced in GUI interfaces. Also, at the time, no other practical pointing device could make a more substantial claim to direct manipulation, so this elevated the status of mice. In reality, you roll a mouse on a secondary flat surface and that two dimensional movement is transformed to corresponding cursor movements on the screen. This is not direct pointing.

[4] Rotation of the display between landscape and portrait orientations was lacking or weakly supported in previous pentop OSs. If present, it was typically a video driver flip of the image performed as a last display manipulation. These drivers could be specific to video hardware and their precise dimensions. The OS had no comprehension of more than one screen display mode. Nor was there any generalized scheme for display dimensions across video devices. Therefore, software applications had no means to be sensitive to the mode of operation or a way to accommodate detailed dimension variation. Windows XP OS supports all of this.

interfaces. This dimension layout issue can be further aggravated by a desire for larger objects to facilitate pointing accuracy and clarity. It causes more expansion of the horizontal size needed for a full display of the application window. A sure sign of an un-tuned tablet PC application is when critical operation menu buttons disappear off the screen to the right, when presented in portrait screen orientation, and no means exists to scroll to the right to ever view or click on them. Even a number of Microsoft's purportedly tablet-friendly core applications suffer from this limitation. It is a pitfall of assuming that just because traditional Windows programs can run on tablet PCs, all usability aspects are satisfied. Or, it may determine that the program must always run in landscape mode, disregarding the ease of possibly holding the machine by another dimension.

2.2 Opportunities in Tablet PCs

While detailed points of software considerations above appear as a litany of negatives regarding pen/slate interfaces that must be overcome, one very persuasive positive should be remembered. Pointing is natural in human communication, perhaps innately so from pre-speech survival of the species. This is expressed in small children that naturally point before they can talk and at a developmental stage turn anything stick-like into a pointed toy. Later in development, one commonly must teach children not to point, as an avoidance of a socially unacceptable gesture, rather than to point. Typing on a keyboard or manipulating a mouse are learned skills that commonly are acquired later in age and require practice to sustain. Only expert typists ever make the leap to non-cognitive use of this skill. In contrast, often in fatigue, injury or other circumstance when verbal expression evades us, we retain the inclination to point as a communication skill. All of this is a practical clue of the natural human comfort with and effectiveness of pointing as a gesture. Capitalizing on this and applying it as the main action in any interface where greater flexibility is desired, is a self-evident advantage. For mobile PC computers, the challenge is overcoming years of bias toward keyboard operation built into applications due to the historic dominance of keyboard hardware.

A second opportunity delivered by tablet PCs is simply for CAI to "go where no one has gone before." Because tablet PCs confer viable mobile PC use, projects can realize CAI advantages in data collection in non-trivial scenarios previously unavailable (while trivial instrumentation migrates to palm or PDA devices). This can include such common interview scenarios as complex and rigorous doorstep screening of households in area probability samples or individuals circumstantially picked in encounter situations; interviewing respondents on the move or flexible situations where an obvious place to setup a laptop computer no longer determines session viability; supporting computer assisted data capture of complex observations where the user or observed event is mobile (such as moving around a hospital ward or a school wing); or encompassing field operation activities that are essentially mobile (like listing operations) combined with CAI on a single integrated machine platform. Also, because this cycle of tablet PCs operate a standard Windows XP operating system version, the full richness of the Windows environment becomes instantly accessible to these new mobile uses. No complications arise on standardization of machine resources, interface look-and-feel, core program operation and programming, external devices and ports to access them, or the basic inter-operability of instrumentation with other Windows computers (for example, a multi-modal operation where just some machines are tablets). The wealth of commercially available Windows-based programs or peripheral devices can be combined readily with tablet computers. Thus, such project functions as security, network connectivity (including wireless), data transmission, or case management applications that most organizations have solved already for the Windows environment do not require reimplementation for tablet PCs. Further, a new body of peripheral devices already integrated with the Windows OS is available for ready project integration, such as GPS units,

authentication tokens, scanners, printers, audio recording, digital cameras, external storage devices, etc.

Another opportunity in tablet PCs is the practicality of new data types. This has two major directions. First, through the use of assorted input devices (mostly external peripherals), new forms of data are viable and their integration is readily supported by the Windows OS (noted above). For example, a digital picture may be more effective than an observer description, or recording the sound of the interview may be more effective for validation and is certainly less expensive than in-person supervisory observation for technique coaching of interviewers. While not strictly a tablet advantage, many of these peripheral forms are more compelling because of the mobile character of tablet PCs. For example, the use of integrated and automated GPS recording is not compelling if the PC capturing the data does not move much or is only setup and operated inside buildings where the GPS signal cannot be recorded (as typical for laptops). Another great use of GPS technology in support of field staff is coordinated use with real-time mapping software to assist in location and route finding, or even location logging. This application of the technology can enhance staff effectiveness and reduce labor and travel costs. But, it only makes sense if the users have access to mobile, not just portable, computing. The second form of new data type utilizes the pen interface to draw, not just point. This enables user drawn diagrams, annotations added to other images, or just freeform notes (retained as an image, rather than coerced to recognized text). This data type is not practical via a keyboard or indirect pointing device (mouse) input interface. Specific extensions to Windows XP, in newer versions, support electronic ink as a new native data type to facilitate this form of data. Also, built-in objects for raw inking and application programs for rich note taking exist in the OS DLLs, available for application integration on tablet PCs.

The greatest tablet PC opportunity is the potential to implement user interfaces precisely developed and implemented to serve the task of interviewing (or other forms of CAI data capture). Going one step further, individual project implementations could be customized for greatest operational effectiveness. This can be done as a software specialization rather than a hardware modification. To date, almost all CAI implementations occur on hardware designed for general business use, not field data capture. The reason is the cost-effectiveness of buying the going PC models on the general market. But, PC manufactures inherently design their machines for their largest market sector, which remains general business users, or for portable machines, these same users on the go. No one designs PC hardware specifically for interviewing. Doing so is cost prohibitive. Leaving the survey industry to muddle through making the best of using hardware devices not designed for their specific use. For example, all laptop computers come with keyboards of *N* keys. But for the vast majority of enumerated interview questions, how many of these keys are actually used at any one time? Even being generous by including all navigation options this number is less than a dozen. Meanwhile, the basic keyboard interaction for an enumerated question is entering an alphanumeric coded response, the classic being 1=Yes and 2=No. So users must elicit the response, correlate it to a coded answer, locate that key(s) on the keyboard and press it to succeed in reliable data entry. While that numeric coding of responses may be second nature to researchers and programmers inured to abstracting data in coded form, there is nothing innate about it to interviewers. It actually takes a cognitive mental operation to transpose the response to the code and enter it (albeit one so often repeated that it may become ingrained). Why do we continue to rely on this as the main input interface: history, habit, and a lack of alternative via keyboard devices. Most interview software, including Blaise, already provides for pointing and clicking at the desired response object or label with a mouse as an alternative to key entry. This eliminates the cognitive encoding step. Still, the mouse is an indirect pointing device and is not well suited to portable set-up or portable built-in devices remain clumsy. This keyboard coding is so familiar that we rarely question it. Our best users are

so facile at it that changing the interface, thereby needing to convert the behavior of existing users, forms another resistance point. This is force of habit or resistance to change.

Tablet PCs, or more specifically their slate style input surfaces, offer a compelling alternative to the core key entry interaction described above. The critical differences are that the output and input surfaces are the same and what shows on the screen, that users then interact with, is entirely under software control. These differences are subtle, but important. They cause an evolution of the GUI principle of what-you-see-is-what-you-get (WYSIWYG) and indirect interaction with the graphics via a keyboard or mouse. On a tablet PC with a pen, now what-you-see-is-what-you-do and the interaction is direct. If the responses for a question are two enumerated choices and three navigation options, via software control we can put objects (rendered in any manner of our choosing) for those, and only those, interactions on a tablet display. The user selects one by pointing directly at and tapping the screen surface with the pen. This is a precise, direct interaction customized to a desired level of detail, under software control. Organizations can make full software development choices regarding the degree of desired customization. We do not have to convince a hardware manufacturer to build us a different keyboard or face static input mechanism arrangements of hardware ["Now let's see, where's that one key among the *N* pieces of plastic I must push to record this response?"] If we now desire to change it or scrap it, we do so by changing the software driving the output object display. If we want more copies, we load more software. If we want two variations simultaneously, we implement a programmed logic branch or make configuration options in one software driver. And for users, we can display any sort of label or image on our input objects and readily change them to suit each instance. ["If the answer is yes, tap the 'yes' button."] CAI processing is fully capable of recoding or deriving that on-screen button press into most any recorded data we desire. All of this enables us to make spectacularly capable and responsive interfaces for our users, with incredibly detailed variations when desired. More likely, we will rely on standardized software for most uses, as less development intensive and more user consistent, and therefore more cost-effective. Still, we have that choice and an untapped potential to do better for our users.

3. Desirable Elements of Tablet PC Interface

Before launching into details on Blaise implementation on tablet PCs, identifying the driving points of the desired interface is relevant. These can be reduced to a couple over-arching principles: visibility, clarity, size, easy interaction, screen arrangement, and open text alternatives. Since these may be principles of all user interfaces, rather than recapitulate the entire Windows GUI standards on these points, this discussion focuses on tablet PC variations with that standard or entirely novel angles. Each item appears individually below.

3.1 Visibility

Visibility of any computer screen and what appears on it are threshold issues for all hardware and applications. The hardware aspects become more critical in portable PCs since the environment of use is less controlled and the resources of the machine are more limited (e.g. size of display, battery supply to power luminescence, etc.). For a mobile PC, the environment gets even less controlled and often even hostile to visibility. Extremes like standing outdoors in bright sunlight or heavy shade can be commonplace for mobile household interviewers. Moreover, while what-you-see-is-what-you-do may be an interface usability advantage, it makes visibility issues more stark. One cannot do what one cannot see.

What can be done for tablet PC use to address visibility? First, get good hardware. Most standard tablet PC screens function decently in common indoor lighting. However, many do not

show well in outdoor light. A number of tablet PC manufacturers make outdoor visibility enhanced tablet displays. Of all the hardware upgrade options[5] this is likely the most valuable on tablet PCs for mobile use.

Some application behaviors in the rendering of screen output can contribute software pluses in visibility. The selection of colors, or lack thereof, can improve extreme lighting visibility. Tablet PC displays can render the full range of Windows supported colors, but not using them in mobile applications, or using them sparsely, can improve visibility. The reason is that the contrast of adjacent imaging is most significant in distinguishing it in very bright or dim lighting. The highest contrast achievable is returning to black and white display schemes. Removing color from the displayed imaging is literally returning to shading that is physically the greatest contrast. The next closest arrangement is juxtaposing exactly complimentary colors. However, this is not as good since the color itself can cause refraction of the desired high contrast for best visibility, can lead to very garish appearing screens, or more likely presents resolution problems for color blind users.

One more software visibility improvement is making objects bigger. Bigger images are inherently easier to see (especially for middle-aged and older users whose eyesight may be suffering from aging effects). Bigger objects are described in more detail below.

3.2 Clarity

Every interface design strives for clarity suited to its use. It must be clear what and where to accomplish some interaction. For tablet applications, especially in mobile use, this is particularly acute. The reason is that the utilization of mobile computing generally implies the user will be doing something else while working with the computer. This may be as mundane as just standing while working. It could be another distinct task. It likely means users cannot devote their full attention to what appears on the screen. They probably will glance at the screen to interact with it, but otherwise, their attention will be focused elsewhere[6]. Therefore, when users glance back at the screen, clarity of presentation will aid them in visually acquiring the point of interaction and the operations applicable to it.

This suggests that the sparsest practical screen presentation works best. Applications should only present those images that significantly contribute to the user performing the immediate task. Even the Windows GUI convention of graying-out disabled buttons is dubious here. First, in tough lighting, the distinction of graying-out can be lost. Second, it takes a moment of cognition to discern the graying-out and its meaning. The scheme of only showing what the user can do at any given time is more pragmatic. This reinforces the working principle of what-you-see-is-what-you-do. Conveniently, a sparse layout also saves screen space so the remaining, most important displayed objects can be enlarged to optimal extent. Also, response indirection or multiple places of representation should be avoided. Give users one place to interact and make it easy to visually locate. And big, so it is easy to access. All promotes clarity.

---

[5] Upgrade options for tablet PCs resemble other PC upgrades: faster CPUs, additional memory, assorted drives, software bundles, etc.

[6] This generally applies to all in-person interviewing. Establishing rapport with respondents, eliciting responses, probing and further interviewing techniques should be the focus of interviewer attention. More bluntly, interviewers should be interviewing; they are not dedicated computer operators. Computer-assistance should be a tool to conduct the questionnaire, not requiring the focus of the operator during live interviewing.

Other points of clarity are grouping and placement of items on the screen (covered below in screen arrangement) and making the necessary interaction obvious (covered below in easy interaction).

3.3 Size

On tablet PC screens, size matters. Regarding the hardware, tablet PCs exist due to a relationship with size and functionality of the human hand and an intrinsic, historically proven, general form factor. Tablet PC screens are relatively large, for handheld devices, which accomplishes better visibility, good manual interaction, and space for the composition of non-trivial applications. Current market conditions also appear to be coalescing around two size dimensions: larger for general users (approaching 8.5 x 11 inches); smaller for the most mobile, limited task oriented purposes (5 x 7 inches and smaller). Among those largest displays, dimensions are limited by the desirable size of the machine footprint. Two manufacturers have even enhanced their screen sizes slightly larger by expanding the screen to the very edge of the machine surface, but not expanding further.

In application software, large size objects enable a number of other benefits, already mentioned. Seeing large objects is easier. Clarity is promoted with fewer, larger objects. Because there are fewer, space becomes available for making them larger. Interacting with them is easier (more below). Further, the accuracy of pointing with the human hand does not correspond to the pixel accuracy of high density PC video. User lab testing indicates that the smallest area for reliable pen pointing/tapping is half a centimeter or larger[7]. In some Windows applications, this is approximately the size of some menu or scrolling buttons. Multiples of this size increases ease and accuracy of use by corresponding proportion.

3.4 Easy Interaction

Large size is one way to increase ease of interaction in tablet applications. A large hot spot area for handheld pens to land on, without undue attention to intricate accuracy, works better. Easy interaction also arises from visibility and clarity since the operative principle is what-you-see-is-what-you-do. Two further properties that apply here are the obvious functioning of object controls and suitability of a control for pen operation.

Alongside large objects that are clearly visible on the screen, easily recognizing what to do to make the object function is necessary to qualify as easy interaction. Naturally, a judgment may vary with the type of user, their level of training and experience with the application. What is obvious to one person may only be so because of their familiarity with the subject. Emergent Windows GUI and web browser standards define a decent set of objects for general users. With minimal training--if not known already from experience--most users can readily recognize click icons, radio buttons, check boxes, drop down arrows, scroll bars, spin controls, plus some others and hybrids. One caveat is that they may appear slightly unusual in large sizes (e.g. large radio buttons often look odd to background users when first encountered). The commonality of all these controls is their visual cues. Relatively consistent operation is also important. Not displaying a drop down arrow on a field controlled by a drop down box object can stymie a user. They get no cue to drop down the applicable selection list. They may have to tap around to discover this action or may conclude they have reached a terminal interface path without this discovery.

---

[7] Jarrett and Su, Building Tablet PC Applications. Microsoft Press, Redmond, WA, 2003. p. 31. See also Part I, chapter 2 for a general discussion of layout sizing of tablet PC applications.

Another easy interaction criteria for tablet PCs, is whether the control works well with a pen. Any object that requires multiple actions, like typing this sentence as text from a keyboard, is going to be less effective via pen input. Pens are good for pointing and simple selection. Avoiding intricacy and multiple trigger actions are good rules. For example, scrolling long lists, while possible with a pen, tends to 1) get too intricate for precise list positioning, 2) expanding the size of the function elements of a scroll bar can actually shrink the travel space of the scroll bar exacerbating the precision effect, and 3) scrolling and selecting takes multiple actions to control. In terms of simplicity, the set of controls of radio buttons, check boxes and drop down lists can accommodate much of plain interviewing interaction well. The reason for including drop down lists here relates to clarity. Moderate lists of radio buttons or check boxes can lead to visual clutter, especially if combined with a multi-field, form layout. A sparser layout, accepting that one tap pulls down the list of contextual responses while a second tap picks the desired one, can be more effective (even though it is literally two actions).

3.5 Screen Arrangement

The windows metaphor of the Windows OS is a screen arrangement protocol. GUI standards add another layer of conventions. However, due to the variation in tangible use of tablet PCs, some screen arrangement assumptions should be changed. The most significant screen arrangement item lies in response to users covering parts of the screen when reaching over it with their hand to interact with the surface. Unfortunately, the largest covering happens directly at the point of selection when the pen taps down on the surface. The hand is close to the selected item and closest to the screen surface, blocking more of it. The area of coverage comes from the lower corner of the screen, from where the user reaches their hand out, and for whichever hand the user is holding the pen. Since most of the population is right handed[8], this will be from the lower right. One advisable application feature is to make advancing to another screen a two tap action by placing a next button to move the user's hand away from covering the screen (i.e. lower right or left). While software is clearly capable of a tap-and-go-next action sequence, this is clearly not desirable regarding input recording accuracy. Getting the user's hand away from the screen, visually revealing their choice, leaves the opportunity to recognize and verify the correct item/response was recorded before moving to the next screen.

Grouping related items in presentation can aid the user in locating where the next interaction needs to occur. Crossing this with sparse presentation, this suggests presenting most single interview questions one at a time. Everything the interview requires to present the query and enter the response is one display unit, unless compounding with multiple fields that form one question unit. Some additional background operations are needed on most interview screens, such as just moving to the next item, backing-up, recording missing values, or calling more detailed help and should be readily accessible. Still, these are subsidiary to the question item of focus and should be treated visually so to promote clarity. This suggests that the main item of attention should be placed as the main item of the screen.
Assumptions about the prime space on a tablet screen are different for tablet PCs than keyboard and monitor based machines. In monitor/keyboard screens the prime area is center and then the closest area to the placement of the hands (that is downward). When holding a tablet this changes to the center and upwards, since the hand is held over the screen itself, in fact covers portions of it

[8] Organizations may want to make a choice whether to rearrange the screen for the roughly fifteen percent of the population that are left handed and therefore likely to reach across the tablet screen from the other direction. Note, most left handed people have pretty good coping mechanisms to deal with designs for the right handed majority.

toward the bottom, and most likely, users are looking upward to see any concurrent activity during mobile use (i.e. to make eye contact with respondents when interviewing). Placement assumptions change with this change in importance of screen areas. For example, placing general action menu bars across the top of the screen emphasizes them on a tablet display, since they occupy some prime screen area, not matching their functional use as subsidiary operations to the main action. Therefore, on tablet displays for mobile use, moving menu bars to the bottom of the display is advisable.

3.6 Handling Open Text and Alternatives

Many PC users take for granted that entering open text always exists. Until now, PCs have sported keyboards from their earliest commercial release and emphasized them. With tablet PCs you can either hinge the attached keyboard of convertible models into position or attach an external keyboard to a pure slate model (either by plugging or infrared port alignment) and conventional PC operation is achieved. However, with slate use of tablet PCs, eliminate the keyboard and how to handle open text? The machine does not limit keyboard use but the core purpose of utilizing a tablet (mobility or flexibility) makes it difficult.

The obvious open text solution when holding a pen device is to use it as a writing implement. Functionally, this is fully possible on tablet screens. In software enabled boxes, electronic inking works well, identical to tangible ink. But troubles arise when trying to recognize that freeform ink as text data. Handwriting recognition (HWR) is required. This is a topic, almost a full discipline, unto itself and beyond the scope of this paper. Suffice to say that applied computer scientists and experienced users commonly advise not relying solely upon text recognition. Some people can use it with great success. For most people the reliability is mixed. Some users can never get it to function. Therefore, the wisdom for tuned tablet applications is provide for HWR when relevant but leave another method for last recourse text entry.

Pop-up keyboards or keypads are the general alternative to open text entry on tablets without relying on HWR. These are pop-up windows with a series of button objects that commonly look like tangible keyboards and behave similarly. Tap a key on a pop-up keyboard and the corresponding letter or digit is entered in the active field. The Windows XP OS standard now includes a pop-up keyboard feature[9]. A secondary software market has also appeared providing alternative and specialized on-screen keyboards that mostly pop-up too. Many pop-up keyboard windows include multiple layout, function and presentation options. However, there are usability problems with pop-up keyboards. A simple one is that the pop-up keyboard takes some screen space and therefore covers something of the application window(s) beneath it. Making the pop-up keyboard drag-able solves this, remaining just an inconvenience when it must be moved out of the way. Another problem may be the activation of the pop-up keyboard. For easiest use, the keyboard should pop-up automatically when relevant or be a background utility that may be activated whenever needed to fill a text field. However, typically the application interface knows when the context may require open text entry, but a background utility is typically an operating system controlled process. Or, the program running on top must reach into or allow process invocation into a lower layer. This does not occur automatically. Applications must be programmed to do so. The Windows OS convention for this is placing a fixed icon on the desktop toolbar that activates the standard pop-up keyboard[10]. This is fine for general users

[9] This pop-up includes multiple input features, one of which is a keyboard. Other features include HWR boxes that then write the recognized text to the active field or voice recognition integration.

[10] Some vendors add a BIOS feature where the keyboard may be invoked through a small icon trailing certain field invocations (although presence or lack of this trailing icon can be problematic to precision

whom retain access to the toolbar. But, many specialty applications (Blaise included) seek to isolate the users from free access to the toolbar, to keep them focused on the special task or prevent inadvertent changes of window focus.

A more intrinsic problem is keyboards were never designed nor intended for this interaction. Organizing the many hot spots of a virtual keyboard by the metaphor of a tangible one is readily recognizable, both in look and function. A potential confusingly large set of distinct key actions are organized at once in a familiar form. Yet, that familiarity leads to another problem. All users hunt-and-peck on pop-up keyboards, even expert typists. While the appearance and function may be familiar, the action is not. The reason is that users interact with such a keyboard image via a pen. A pen is a singular pointing device, only having one selection point. On tangible keyboards, good typists use multiple parallel fingers, each with a distinct position and ability to press keys. The pop-up keyboard interaction is no different than using a tangible pen to type on a tangible keyboard. It is relatively slow and laborious. Keyboards simply are not ideal as a single point/select interface.

If an OS built-in solution for handling open text is not optimal, what are the alternatives? These involve two tactics (possibly in combination): design away the open text or finesse the application interface for tablet use. Both require some attention to tuning for this platform. Examples of designing away open text entry in interview applications include re-sequencing or perhaps extending question sets, into more precise querying or response categories, to eliminate open ended responses that necessitate open text entry. Another variation is decomposing responses into compound pieces that can become category or list selections. For example, a year entry field can be broken into a drop down with categories combining century and decade (the first three digits or a four digit year;) and then a second drop down of year within a decade (fourth digit of a four digit year). While an unorthodox division of a year date, this is effective by needing a small number of actions (2) against reasonably sized category lists (10 each to span 100 years). If drop down lists are situated on the actual entry fields, the interaction is cued in place and therefore easy to discern and operate.

While liberal and creative use of list selections or categorized responses, or careful crafting of questions and responses can do a lot in interview applications, invariably some required open text field still arises (such as names, addresses, other specify description, remarks, etc.). Pop-up keyboards or HWR—for those that can accomplish it—is tolerable in small doses. There is one more option. Taking the open text as freeform writing and retaining it as inked data, rather than converting it. This does not work for any entry that will be reused later in real-time, such as a name that is later displayed as question text fill. Without such additional reference, it does work well. For example, instead on insisting on real-time text recognition for an other specify field, take the response as ink and store it. The new Windows native "journal" data type (specifically for electronic inking, also supporting inking combined with other graphics, thus supporting annotations) is intended for this. In addition to the ink imaging, all the stroking and pace attributes of the electronic inking are saved in this native type so no degradation of later recognition occurs[11]. If the user can return to the inked text, at their convenience later, they can perform the recognition then. Or for an item such as other specify text, the inking can be passed

---

field sizing or placement) or by right-clicking (performed as a tap while depressing a button on the pen shaft with a stylus) a text field and selecting a pop-up keyboard item on a pop-up menu. These are techniques carried forward from prior pentop OSs, but are additions to the Windows XP standard.

[11] This was a weakness of earlier electronic ink/HWR implementations. The HWR had to be performed in real-time or the subsidiary ink attributes were lost because only the image of inking was stored permanently.

downstream in data flow to other human users for interpretation[12]. This method can be applied to other situations by analyzing when an open text entry really needs to be stored as text data. If not, retention as electronic ink in real-time is viable.

4. Blaise Use on Tablets

As mentioned, Blaise as a Windows program functions on tablet PCs. Still, while not abjectly unfriendly to tablet operations, Blaise is not tuned for it either. In its out-of-the-box configuration, the history of Blaise as a keyboard application is apparent[13]. For hardware and usability reasons regarding general field interviewing as a mobile use explained above, I suggest that ongoing Blaise development should be focusing on such software tuning factors for pen/slate operation. Doing so not only anticipates a probable eventuality of tablet-like field computer use, but by removing a resistance point for adoption, may speed that eventuality to fruition.

Westat currently has a number of field projects utilizing tablet PCs. The most developed are complex, custom implementations aimed at full utilization of the pen/slate interface. We also have implemented field data collection using Blaise instruments deployed on tablet computers. In doing both of these, we have gained insight on the potential of the pen interaction supporting interviewing and the pluses and minuses of more conventional Blaise software.

Blaise is clearly the leading COTS interviewing software. We are delighted that, as a Windows software product, it operates on tablet PCs out of the box. The programming, data preparation, instrument infrastructure, data storage aspects (all the non-presentation and interface) of Blaise function here, just as on other Windows PCs. However, having identified the output presentation and input interaction as key aspects of optimal tablet PC operations, these aspects work at a basic level for Blaise out of the box. Still, Blaise is a good example of application software that has inherited, for historic reasons, a user interface cultivated around keyboard input. As such, its operation with a pen interface shows some problems. Of these problems, existing presentation customizations, and therefore programmable variation, can address much of the output tuning. However, the input tuning is more dependent on supported system operations and therefore more limited.

4.1 Strategies for Blaise Implementation

The ideal for any software integrator is finding great applications perfectly suited to their particular operating environment, out-of-the-box or with very limited setting of prepared configuration options. However, expecting any one software package or system to deftly support all environments clearly is unrealistic, especially for relatively new platforms that may vary with the legacy ones for which the software was designed. If the ideal is unavailable, the alternative is finding software with configuration depth, modifiable or reconfigurable to satisfy the amount of precise variation desired. Blaise qualifies as this software for interview applications. We can even conceive of a number of strategies for how Blaise might be tuned for better pen/slate operation.

---

[12] In my experience, all other specify responses are passed to a backend data coding shop where a data clerk or subject expert reads them and back codes or specially codes the response or accumulates a list of extraordinary responses. If a human is interpreting the text and the data infrastructure is in place to pass the inking (as a blob field) to the point of interpretation, they can just as well read freeform inked writing as fully recognized text.

[13] If, for no other reason, there are some actions that only can be triggered by keyboard action (e.g. popping-up a table-based item search with a Backspace key press or mapping actions to function keys).

In picking strategies, we start with simpler ones and advance to the more complex (although a clear effort to reward ratio exists). The first strategy is the ideal: what is useful in Blaise ostensibly running in its default state, out-of-the-box. The next strategy is using the full depth of Blaise offered options, properties or programmable features to achieve a particular presentation. A new strategy is treating the instrumentation as a Blaise IS application (web instance). This may be an obvious approach in multi-modal instrument applications (where both CAPI and web-CASI administration of the same instrument is needed). Still, a local instance of a web server can be deployed on distributed PCs and the browser interacts with that mobile server. Regarding user interface details, this pushes implementation into the domain of web browser operation or what can be accomplished using web screen rendering (i.e. HTML, XML, XSL, Java Scripting). Another more intensive Blaise use on tablets strategy could arise from utilizing alien routing and replacing the I/O aspects of operation, at proportionally more customization effort. The last strategy takes this a step further. A new application can be programmed that only uses a Blaise backbone for instrumentation, via Open Blaise Architecture access, but sustains the whole session operation itself, including the user interaction. All of these amount to further strategies for Blaise implementation. We will try to put them in context.

4.1.1 Blaise, Out-of-the-Box (Default)

As noted, Blaise readily runs on Tablet PCs[14], directly out-of-the-box since it is a Windows program. However, as established in most of this paper, simple running on tablets in default configuration as a Windows program does not amount to easy use in pen/slate operation nor is advisable to do. With any project software implementation, much depends on the requirements, schedule, available resources and budget, and the culture of the implementing organization. For something fast, small in sample size or just trial, using out-of-the-box Blaise can be done.

Problems encountered here spread across usability criteria and generally can be labeled as a modest lack of pen operations tuning. The outset trouble may be visibility of the initial Blaise windows. While the blue, cream, grey and black writing color scheme is pleasing to the eye and a good branding of the software, it is not high in contrast. This means it washes out in bright lighting, (particularly the black on grey form pane of the screen) limiting visibility. The default sizes of the screen output verges on too small for effective visibility beyond a controlled PC use environment. For clarity default Blaise does well. The three pane organization of default Blaise screens conceptually separates operation areas, organizing output and input interaction. Consistency contributes to clarity here, even though intuitive visual cues are somewhat lacking. The redundancy of the info pane and form pane, with two expressions of an input field is not helpful to tablet/mobile users, or certainly occupies more screen space than worth any contribution to tablet operations. Regarding size of input objects, the defaults are plainly too small for easy pen interaction. Many objects are at or smaller than the half centimeter minimum sizing consideration for pen operation. Making everything larger to better utilize screen area would improve pen operability. Screen appearance of default Blaise operation roughly matches general business use software. The largest negative is placement of the menu bar which leads to more hand coverage of the screen display to get to regular application buttons (including the next button[15]), rather than less. For open text alternatives, default Blaise offers none beyond what Windows XP provides in the background. While features exist in Blaise 4.6+ to call a pop-up

[14] Referring here to the operation of the DEP on a programmed Blaise instrument.

[15] This is the only means to regularly advance to the next item that is display resident in default Blaise screen operations. The only other default method would be popping-up a keyboard and taping the enter key. This later method is much too burdensome for constant use between questions.

keyboard or HWR overlay, these amount to setting detailed field properties and therefore are more substantial than the default configuration. This leaves only standard Windows XP pop-up keyboard and HWR options. If the Windows desktop toolbar is not retained on top of the application, the activation icon for the Windows pop-up keyboard is unreachable. Also, some default Blaise operations require keyboard keystrokes to activate. Lastly, default Blaise is clearly screen oriented for standard PC monitors. It has landscape orientation dimensions with no easy configuration option or system states to vary the orientation. Further, even if the application were left as landscape orientation and the hardware rotated the screen presentation, significant screen components like the next button on the menu bar and later columns of multi-column answer fields start spilling off the right edge of the screen without any means to scroll in that direction to move them back onto the display for input interaction.

If the objective is running something very rapidly and there is recourse to using an external keyboard for the terminal issues, then perhaps one can show something. A summary judgment on the advisability of relying on this for production data collection is likely no. The system and application programs can run but users will quickly encounter multiple, and perhaps insurmountable, usability issues.

4.1.2 Blaise, Fully Tweaked Standard Environment

Another strategy for deploying Blaise for tablet PC slate operation is fully tweaking the Blaise DEP environment per provided configuration and programmable options. This is commensurately more development intensive than default configuration. Still, to a degree, one can manipulate the appearance as desired (although this can mean rigorous use of Blaise options or properties occurring in multiple places—some requiring coordination—and not all thoroughly documented). One note is that the continuum between default Blaise and fully tweaked Blaise in not as absolute as represented here (done as extremes to facilitate explanation and characterization of two strategies). Blaise configuration options largely stand-alone so they can be implemented individually yielding many incremental stages between default and full leveraging of options and configuration.

First, to achieve greatest color contrast to promote mobile visibility, the background and foreground colors of windows can be reset. To address the default small size of windows display content, new tags can be defined and added to the displayed text to make it bigger. Doing so comprehensively across a significant questionnaire requires moderate structuring of the tag scheme and organization and discipline in programming implementation. A problem arises when making large adjustment to the response labels displayed in the Blaise info pane. As the text gets bigger in font size, it also pushes horizontally to the right. When this causes wrapping onto the next row, the interactive objects (radio buttons and check boxes) can become misaligned with their matching large font labels. This confounds accurate data entry if interacting with these objects since it becomes hard to resolve which object to tap without scanning the object to label correspondence of most responses (for example, scanning down the rows of labels and buttons counting each occurrence to confirm the match-up).

Size of Blaise interface panes can be manipulated to eliminate, or at least minimize, one of the redundant panes (info or form). However, which to eliminate is somewhat problematic since desirable default behaviors reside in both. The default arrangement of the info pane for simple questions already fits desirable pen/slate objectives, so requires no change beyond object/text enlargement. In a typical instrument, the majority of responses are simple enumerated types. Thus, retaining the radio buttons or check boxes in the info pane is the easiest recourse. Yet, Blaise built-in open text and drop down entry fields (plus some further extended or callable entry

object types) are only available in the form pane. This difference means implementing one of two fully tweaked options. Either, vary the pane presentation per item context (info for most, form for some), which is specification and item detail intensive. Or, fully compose a form pane scheme to handle all item types (most likely done as type specific templates), which is more general implementation intensive. Since dimensions and placement of panes can be manipulated with some effort, making a portrait mode layout of Blaise screens is possible. However, a choice must be made whether landscape or portrait presentation is used; both cannot be accommodated.

Discussion above of the default menu bar sizing and placement apply in this strategy too since neither of these is reconfigurable in Blaise. A good pen friendly feature that can be enabled is a row of key-like buttons that appears at the bottom of the form pane for enumerated responses and also includes an on-screen Enter button/key. These are positioned well at the bottom of the screen and large enough for adequate threshold pen-capable sizing. Yet what appears on these buttons remains the numeric code representations of the enumerated responses so they are just screen proxies for the tangible keyboards keys. Consistent with the argument made earlier about the opportunity to interact directly with context sensitive on-screen answer objects as ideal for pen interaction, this still falls short of full pen tuning for Blaise applications.

The later Blaise releases include new object types that can be assigned to individual input fields in the form pane, such as drop down lists and date picking. Plus, an open user action calling mechanism to a specified external object is available. It appears to users as a button with ellipses at the end of the field. These are flexible and powerful extensions to the form pane operation and can be applied for pen/slate tuning. For example, drop down lists, while rather small in default sizing, still become configurable extensions of tweaked Blaise operation. The user action call button opens input operations further to supplemental or entire replacement objects. However, a glimmer of lack of pen tuning still remains here as these can be intensive extensions to specify per field or a better integrated behavior is desirable. For example, the user action can only call one object and requires the user to engage it. For ideal pen tuning, the relevant type of input object (pop-up keyboard or HWR box) per the context of the item should appear automatically. If the user must make a choice and action, it might be choosing between dual open text modes (pop-up keyboard or HWR) rather than whether to invoke something. Also, the consistent ellipse labeling of the user action button ends up masking any variation applied to this operation on subsequent fields. Allowing labeling (or better yet, an icon image) on this button could provide a useful field behavior cue.

For most survey organizations with existing Blaise experience, this is likely the optimal strategy. It continues to use the familiar Blaise development environment, with some configuration modifications, to construct, deploy and operate familiar Blaise DEP operations, just on a hardware variation, tablet PCs. Is there room for platform specific improvements? Yes. Perhaps this paper may identify and lead to some of those improvements.

4.1.3 Using Blaise IS Instrumentation

Another strategy for implementation of Blaise on a tablet PC, that surmounts issues noted above regarding the direct Blaise user interface, is using Blaise IS. This is the web extension system. This is an ideal solution when multi-modal administration of the instrument is desired and one of those modes is over the web. Even without the web requirement, a questionnaire instrument can still be developed and deployed for isolated local PC operation relying on a locally running web server. In doing, the user interface issues shift to what can web browser and server architecture support and then the details of the Blaise IS implementation that enable or limit the interface. Much of this subject is beyond the scope of this paper and still an emerging system and area of expertise. Generally, the web environment has two aspects that cross over well to pen/slate instrumentation. First, since most conventional web operations involve self-interacting users, web interfaces tend toward simpler methods. This corresponds well with pen/slate aims of clarity, easy interaction and screen appearance. Second, as a relatively late developing interface, web interaction supports newer conventional GUI item objects. Therefore, the available object set tends to have more potential breadth than straight Blaise DEP interface. Such objects as spin controls and rich forms can be supported within browser interfaces. Lastly, web interfaces emphasize point-and-click operations that matches the desirable point-and-tap interaction of pen/slate interaction.

The downside of this strategy is that it is more than just a presentation replacement. It involves a whole change in architecture, set-up and support to a web browser/server model and likely a new skill or tool set for the preparation of the interface. A metaphor for this change is that to get a new steering wheel and dashboard in an automobile, it comes connected to and only operates with a new engine under the hood and you have to go to new filling stations to get a new type of fuel.

4.1.5 Using Alien Router and Procedures Instrumentation

This strategy for implementing Blaise on a tablet PC is the first step in building a custom interface. One can get precise desired behavior at the impact of substantial programming to accomplish it. The implementation is replacing the presentation layer of Blaise operation with one custom developed and connected to DEP operation through alien router calls. The Blaise engine sustains the session operation but yields control to external modules to conduct the input interface. At Westat, we use this strategy for self-responding interview (CASI) instrumentation where a simpler or friendlier user interface is necessary. The same method could be applied to a targeted pen/slate interface.

Here the limit is the creativity, time and budget of the programming effort going into the external interface programming. The alien router to external module mechanism is pretty narrow. The external interface modules get limited information about any one screen instance through the calling mechanisms. The external modules can still read and write their own configuration and display aspects, to do more complex operations, yet this involves the burden of sustaining suitable external metadata and relevant hooks to the Blaise metadata to pull it all together for execution. A metaphor for this strategy is flying missions from an aircraft carrier. Those missions can do many things but one must supply the special airplanes that the vessel (Blaise) can carry and operate with the launching and landing system on board.

4.1.6 Using Open Blaise Architecture (OBA) Instrumentation

This strategy uses the Blaise backbone but drives the execution environment with a replacement program. Here an external process not only operates the interface but also sustains the program execution, only calling Blaise for specific operations. In Blaise system parlance, this is using the Open Blaise Architecture, the ability of external processes to reach into Blaise functionality through a program interface (primarily contained in the Blaise Component Pack (BCP)). This is a reversal of operation control in the alien router strategy described above. Operations that you might want Blaise to sustain include the data model and rules aspects as products of Blaise programming. From a response item (the first one must be primed), the external program returns the value to store to Blaise and invokes the Blaise rules to determine the next item to pose. Then the program queries Blaise for item metadata description and perhaps combines further external metadata, presents the question via its own interface and awaits further input. This input is returned to the Blaise engine and the cycle repeats.

Again the depth of the interface rests on the creativity, time and budget of the programming effort to build the external program. A metaphor for this strategy could be that you can fly to any destination, but first you must build the airplane, using construction modules for the crew/passenger compartment and avionics controls from Blaise, to do so.

4.2 Recommended Enhancements

The following list is reasonable enhancements that might be made to the user interface of the Blaise DEP, where system modifications could exert significant improvements in the pen/slate operation. Most are software system resolutions to usability problems identified in the first two strategies, mentioned above.

4.2.1 Misaligned and larger selection objects

The misalignment of selection objects, as font sizes of labels are increased and cause line wrapping, amounts to a bug in the DEP presentation layer. This bug should be addressed. Within reasonable sizing boundaries, the relatively simple controls (radio-buttons or check boxes) should line-up, displaying and operating as expected. Further, we suggest that the size of these controls should expand proportionally to their label text. In this usage, expanding the text is done to make it more visible, clearer, or easier to interface with the objects. Display objects that go along with these labels should adjust too, for the same reasons.

4.2.2 Richness in Placing Input Objects/Types

Resolving the duality of placing input objects between the info and form panes is desirable. The advantage in the info pane is easy implementation for the majority of simple enumerated items. The advantage in the form pane is necessary additional input object depth and greater display control, at considerable configuration burden. Perhaps it is time to drop the info pane and extend its easy default displays to the default form pane behavior while retaining the greater depth of field configuration in the present form pane. A further desirable enhancement is form-like combination of multiple fields for strongly related data fields (for example, collecting date of birth as distinct day, month, year fields). Layout specification through the ModeLib editor begins to support this, yet the row and column organization is klugey and imprecise. Refining this may amount to switching to a visual painter interface and undoubtedly represents significant effort to adopt a new scheme.

4.2.3 Expanding Keyboard-only Actions

Systemic actions that only can be invoked by tangible keyboard keystrokes should be expanded to on-screen buttons. A good example is the need to press a backspace key to launch a lookup list from its input field. Also, anything set to a function key is not accessible when operating without a keyboard. We suspect there may be further instances that we have not encountered. An additional level of enhancement here might be providing for application described action buttons in the Blaise GUI interface, although a workable scheme to relate a displayed object to <u>any</u> action may prove elusive. Some desired actions might be external calls, others might be proxies for existing Blaise operations.

4.2.4 Sizing or Placing Menu Bars, Sizing or Adding Menu Buttons

The standard Blaise menu bar should be larger for pen/slate operations. It now verges on too small for effective pen use. Further relocating it to another placement on the screen (specifically the bottom) is desirable. Also, adding application determined menu buttons is desirable. The menu bar is the regular place for general interaction invocations (such as backup or entering missing values). A good solution for placing a pop-up keyboard button is on the menu bar (more below).

4.2.5 Pop-up Keyboard Controls

Adding pop-up keyboards as a user action property of input fields (then appearing as an ellipse button) is a quick means to get this external call associated to individual fields. Yet this may still be one user action too many to simply start entering known open text for items or not consistent enough for a core object of pen interfaces. Mentioned tablet usability criteria (clarity and easy interaction) suggest that the keyboard should pop-up automatically when needed or background access to it should appear on the menu bar. The user action control then might remain available for supplemental HWR/ink boxes on such fields, while a menu button could remain the fallback method of entering open text. Or, if pop-up keyboard access was optionally configurable on the menu bar, it could become available for out-of-the-box Blaise use on tablets.

4.2.6 Dual Screen Orientation Support

Presently, only one screen set-up is provided in Blaise operations. To fully support dual screen orientation options, parallel set-up control for a second orientation is required. This may prove a large change. It may resemble the support relevant to multi-language support.

References

Binzer, G. and Hill, D. *NHANES mobile pentop interviewing environment*. (2002, August) Invited presentation, Joint Statistical Meetings, American Statistical Association, New York, NY. (2001, March) Originally presented, 2001 Federal CASIC Conference, Washington DC.

Cortez, J. *Exploring the use of Blaise in pen-based tablet computers*. (2004, March) 2004 Federal CASIC Conference, Washington DC.

Della Torre, K., Hill, D., O'Reilly, J., and Landerman, L., *Extending CAPI data quality to new settings with Tablet PCs*. (2003, December) Washington Statistical Society Symposium, Arlington, VA.

Hill, D. and Huey, R., *Technology enabling practical mobile data collection*. (2004, May) Invited presentation, Washington Statistical Society, Washington DC.

Hill, D. and Binzer, G. *Preliminary evaluation of Microsoft XP Tablets for household interviewing*. (2003, February) Invited presentation, 2003 Federal CASIC Conference, Washington DC.

Hill, D. and Montalvan, P. *Pen-based portable computers and handwriting recognition in large demographic household surveys*. (2002, February) 2002 Federal CASIC Conference, Washington DC.

Jarrett, R. and Su, P. Building Tablet PC Applications. Microsoft Press, Redmond, Washington, 2003.

Van West, J. Tablet PC Quick Reference. Microsoft Press, Redmond, Washington, 2003.

Contacts

Using Blaise for Coding Listing Instruments
Robert Wallace, U.S. Census Bureau, USA
malcolm.robert.wallace@census.gov

Utilisation de Blaise dans la répartition automatique des fiches adresse entre enquêteurs des enquêtes ménages
Georges Bourdallé, INSEE, France
georges.bourdalle@insee.fr

Contact History Instrument
William E. Dyer Jr., U.S. Census Bureau, USA
william.e.dyer.jr@census.gov

Michigan CATI Sample Management System (SMS) – Blaise CATI Enhanced
James Hagerman, University of Michigan, USA
jrhags@umich.edu

The Progamma Library of Blaise Routines
Dirk Sikkel, Sixtat, The Netherlands
dirk.sikkel@uvt.nl

Understanding the Blaise Selective Checking Mechanism and the Appropriate Use of KEEP Statements
Pamela Ford, Statistics Canada, Canada
pamela.ford@statcan.ca

Instrument assembly, documentation and release
Michael Cross, Australian Bureau of Statistics, Australia
michael.cross@abs.gov.au

Using Arabic in Blaise
Shifra Har, Central Bureau of Statistics, Israel
shifra@cbs.gov.il

Codification automatique en ligne
Stéphane Humbert, INSEE, France
stephane.humbert@insee.fr

Blaise Instrument Design for Automated Food Coding
Lois Steinfeldt and Ellen Anderson, U.S. Department of Agriculture, USA
lsteinfeldt@rbhnrc.usda.gov

Generating Blaise with Blaise
Gerrit de Bolster, Statistics Netherlands, The Netherlands
gblr@cbs.nl

CodeBuilder - An Initial Authoring Aid
Mike Egan, Statistics Canada, Canada
mike.egan@statcan.ca

Generic System Interface Instrument
Curtis Ziesing, U.S. Census Bureau, USA
curtis.n.ziesing@census.gov

Randomly Generating Interviews to Validate a Complex Survey
Lindsay Walton, National Research Bureau, New Zealand
lindsay@optimumhealth.co.nz

Moving from 43 Generated Unique Instruments to One National Instrument
Roger Schou, U.S. Department of Agriculture, USA
roger_schou@nass.usda.gov

Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey
Robert Bumpstead, Office for National Statistics, United Kingdom
robert.bumpstead@ons.gov.uk

Replaying Blaise Audit Trail Files for Data Verification
Jason Langfahl, University of Michigan, USA
langfahl@umich.edu

Replaying ADK Files for Testing Blaise Applications
Jason Ostergren and Rhonda Ash, University of Michigan, USA
jostergr@isr.umich.edu

Blaise AT Report System
Youhong Liu and Gina Cheung, University of Michigan, USA
yliu@isr.umich.edu

Converting Translated Blaise System Texts To A New Version
Rob Groeneveld, Statistics Netherlands, The Netherlands
rgnd@cbs.nl

USDA's Experience Using Blaise for Collecting Food Intake Data
Jaswinder Anand and N. Raper, U.S. Department of Agriculture, USA
janand@rbhnrc.usda.gov

Handling Post Processing of a Large Complex Blaise Instrument in a Continuing National Survey
John Clemens and Lois Steinfeldt, U.S. Department of Agriculture, USA
jclemens@rbhnrc.usda.gov

Automatic Customization of Datasets Using Cameleon
Barbara Bibb and R. Suresh, RTI International, USA
bibb@rti.org

Error Reporting and User Interfaces for Post-collection Data Editing
Pavle Kozjek, Statistical Office of Slovenia, Slovenia
pavle.kozjek@gov.si

The new German Microcensus starting in 2005, organisational and technological aspects, the use of Blaise
Holger Hagenguth, Federal Statistical Office, Germany
holger.hagenguth@destatis.de

Instrument Design for a Blaise Multimode Web, CATI, and Paper Survey
Mark Pierzchala, Claire Wilson, and Debra Wright, Mathematica Policy Research, USA
MPierzchala@Mathematica-Mpr.com

System Implementation for a Blaise Multimode Web, CATI and Paper Survey
Leonard Hart, Yuki Okada, and Amanda Foster Sardenberg, Mathematica Policy Research, USA
lhart@mathematica-mpr.com

Mode effects and maintenance of data quality in a CAPI/CATI survey
Tim Burrell, Office for National Statistics, United Kingdom
tim.burrell@ons.gov.uk

USING CAWI IN HOUSEHOLD SURVEYS – FINDINGS FROM OFFICIAL GERMAN STATISTICS
Anke Gerhardt and Dr. Thomas Pricking, Statistical Office of North-Rhine-Westphalia, Germany & Birgit Kuchler, Carola Kühnen and Anette Stuckemeier, Federal Statistical Office, Germany
anke.gerhardt@lds.nrw.de

Some Ideas On Securing And Processing A Blaise IS Survey
Rob Groeneveld, Statistics Netherlands, The Netherlands
rgnd@cbs.nl

Exploring the capabilities of Blaise IS
Jim O'Reilly, Westat, USA
jamesoreilly@westat.com

Developing the On-line Milk Production Survey using Blaise IS
John O'Connor, Central Statistics Office, Ireland
John.OConnor@cso.ie

Programming Guidelines for Good Data Documentation
Sue Ellen Hansen, University of Michigan, USA
sehansen@isr.umich.edu

Blaise Documentation System
Peter Sparks and Youhong Liu, University of Michigan, USA
zebulon@isr.umich.edu

An evaluation of Delta, a documentation tool
Rebecca Gatward, Office for National Statistics, United Kingdom
rebecca.gatward@ons.gov.uk

Recording Interview Sound Bites Through Blaise Instruments
Rita Thissen and Gilbert Rodriguez, RTI International, USA
rthissen@rti.org

Deploying Blaise to Tablet PCs for Mobile Use
Dave Hill, Westat, USA
hilld1@westat.com