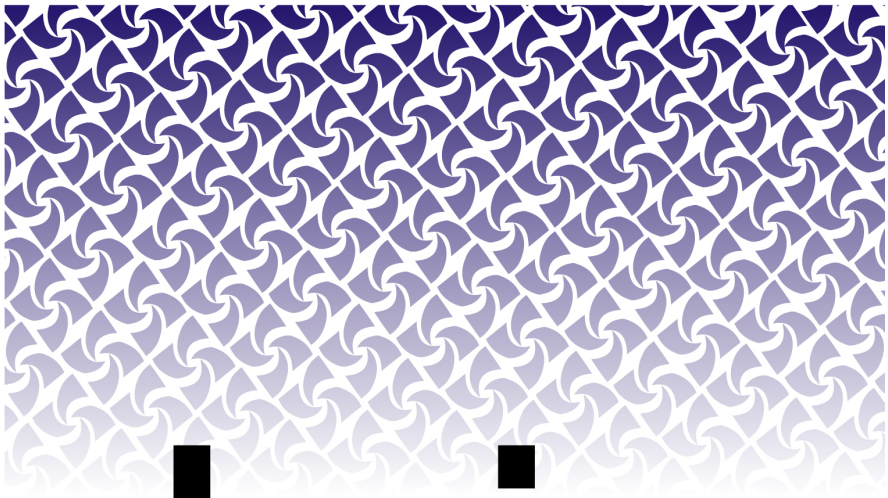
10th Users meeting



2006

blaise®

Proceedings of the

10th International Blaise Users Conference

Papendal, The Netherlands, 9-12 May 2006

Statistics Netherlands

Preface

This document contains the papers presented at the 10th International Blaise Users Conference held in the Papendal Hotel and Conference Centre in Arnhem, The Netherlands, from May 9th to 12th, 2006. The conference included four days of technical papers and presentations on the use of Blaise and related topics.

The Conference Programme was organised and planned by the Scientific Programme Committee, chaired by Vesa Kuusela, Statistics Finland. Members of the committee included:

- Vesa Kuusela (Statistics Finland, Chair)
- Karen-Ann Bagwell (US Bureau of the Census)
- Jelke Bethlehem (Statistics Netherlands)
- Bill Connett (University of Michigan, USA)
- Rebecca Lui (Statistics Canada)
- Tony Manners (Office for National Statistics, UK)
- Mark Pierzchala (Mathematica Policy Research, USA)
- Fred Wensing (Australian Bureau of Statistics)

The editing of the conference proceedings was carried out by Jelke Bethlehem.

IBUC 2006 was organised and hosted by Statistics Netherlands. The Local Organising Committee consisted of:

- Lon Hofman
- Han Hölsgens
- Peter Sinkiewicz

Contents

Session 1.1: Blaise in Practice

Labour Force Survey: Handling Multiple-Household Dwellings in Blaise
Zipora Radian, Evgenia Luskin & Chaggit Breuer 3

Listing Part 2 - Using Blaise 4.7 for Coding Listing Instruments
Malcolm Robert Wallace, Roberto V. Picha & Michael K. Mangiapane 17

The New, Adapted German Microcensus and the Integrated Labour Force Survey – User-specified Solutions with Blaise
Stephan Matthias 35

Training and Learning Opportunities using Blaise
Colin Setchfield 49

Session 1.2: History of Blaise

Blaise – Alive and Kicking for 20 Years
Jelke Bethlehem & Lon Hofman 61

Session 2.1: Standardising

In-house Support for Blaise Application Developers and Users
Janica Cajhen & Pavle Kozjek 89

Blaise Programming Techniques for Better Documentation
Peter Sparks & James Hagerman 93

Standardizing and Automating Blaise Output Test Data Delivery (TransSAS)
Latha Srinivasamohan 113

Developing an Integrated Household Survey (IHS) Blaise Questionnaire from Four Major Social Surveys
Tim Burrell 125

Session 2.2: Cati Considerations

Can We Reach You by Telephone?
Fannie Cobben 137

Experiences of Usage of Blaise in CATI Surveys
Stefania Macchia & Manuela Murgia 151

NASS CASIC Survey Administration System
Everett Olbert & Roger Schou 167

A Multi-mode CATI-Web Survey Experience with Blaise
Craig Ray 183

Session 2.3: Multi-mode

Disparate Modes and Their Effect on Instrument Design
Mark Pierzchala 199

Questionnaire Design in Blaise for a Multimode Survey
Hilde Degerdal & Jan Haslund 211

Distributed CAPI Interviewing with Blaise® IS for the Web for the Adolescent Medicine Trials Network for HIV/AIDS Interventions
Rick Mitchell 221

Blaise IS and Accessibility
Jim O'Reilly 231

Session 2.4: Challenges at Statistics Netherlands

The Dutch Annual Business Inquiry: Developing and Testing an Electronic Form
Ger Snijkers, Evrim Onat, Jo Tonglet, Rachel Vis & Robert 't Hart 243

CASI questionnaires with Blaise 4.7
Gerrit de Bolster 261

Basil, A New Tool for CASI in Blaise
Roger Linssen & Jo Tonglet 267

Session 3.3: Transformations

Computer Assisted Coding by Interviewers
Wim Hacking, John Michiels & Saskia Janssen-Jansen 283

MultiRelaX with BCP as a replacement for Cameleon
Marien Lina & Richard Frey 297

Session 4.1: Connecting

Experiences with Dynamic Link Libraries
Youhong Liu & Gina-Qian Cheung 309

Methods of Integrating External Software into Blaise Surveys
Lilia Filippenko, Joseph Nofziger, Mai Nguyen & Roger Osborn 317

Using the Blaise Component Pack from Within the .NET Framework to Develop Data Management Tools
Leonard Hart, Robert Thompson & John Mamer 335

Using Blaise to apply edits to data held in an Input Data Warehouse
Fred Wensing 349

Session 4.2: Alongside the Data (Paradata)

Blaise PlayBack and Recovery System
Youhong Liu & Gina-Qian Cheung 363

Active Management: A New Strategy for Managing Data Collection
Luc Tremblay 371

Work Flow for the Weighting of the German Microcensus Data Using Blaise Bascula
Kirsten Iversen 381

Posters

User-friendly web surveys
Hayo Bethlehem 395

Using the Blaise Component Pack in the .NET environment
Rob Groeneveld 401

A Calculator DLL
Roberto Picha 405

Programming for Navigation in the Blaise System
Mark Pierzchala 409

The Selection of Strata in Nonresponse Adjustment
Barry Schouten 417

CAPI System at Central Statistical Bureau of Latvia
Norberts Talers & Palvels Onutrijevs 425

Multiple Researchers working at the Blaise Benchmark Services for the Disabled Act (WVG)
Carlo Vreugde & Mark Gremmen 427

Session 1.1: Blaise in Practice

Labour Force Survey: Handling Multiple-Household Dwellings in Blaise
Zipora Radian, Evgenia Luskin & Chaggit Breuer
(Central Bureau of Statistics, Israel)

Listing Part 2 - Using Blaise 4.7 for Coding Listing Instruments
Malcolm Robert Wallace, Roberto V. Picha & Michael K. Mangiapane
(U.S. Census Bureau)

The New, Adapted German Microcensus and the Integrated Labour Force Survey – User-specified Solutions with Blaise
Stephan Matthias
(German Federal Statistical Office, Germany)

Training and Learning Opportunities using Blaise
Colin Setchfield
(Office for National Statistics, UK)

Labour Force Survey: Handling Multiple-Household Dwellings in Blaise

Zipora Radian, Evgenia Luskin, Chaggit Breuer (Central Bureau of Statistics, Israel)

1. Introduction

Israel's Labour Force Survey (LFS) introduced the use of Blaise to our Central Bureau of Statistics in 1999. The survey tracks fluctuations in Israel's labour force, its size and characteristics, the extent of unemployment and other trends. It also provides demographic information on Israeli households.

The LFS is a continuous panel survey, returning to a sample of approximately 12,000 households four times over the course of a year and a half. Households are interviewed face-to-face in Panels A and D, and by phone in panels B and C. Data collection thus far has been achieved using CADI and CATI respectively, but as of next year we hope to begin using CAPI instead of CADI.

The survey is sampled from a framework of dwellings. One of the problems we encountered during development of the CAPI questionnaire was related to Multiple-Household Dwellings (MHDs). A household is defined as a group of persons living in one dwelling who have a common expense budget for food. A household usually consists of a family, but may also consist only of one person or include persons who have no family relationships. Approximately 3% of all dwellings have multiple households sharing their space. This configuration is most often found among immigrants, students and foreign workers. Surveyors are expected to interview a representative of each household in a dwelling. A separate questionnaire must therefore be used for each household.

This paper outlines the complexity of the issue and the solution provided for conducting MHD interviews. Using Blaise and Maniplus, we have developed a user-friendly questionnaire, able to identify multiple households and to generate a separate questionnaire for each such household.

2. Description of the Labour Force Survey

2.1. General

Since 1954, the Central Bureau of Statistics (CBS) has been conducting annual continuous Labour Force Surveys, interviewing a sample of 12000 households each quarter. The survey population includes the entire permanent population of Israel, aged 15 and over.

The survey's primary aim is to gather information on changes in population composition and on the state of employment and unemployment in Israel.

2.2. Sampling

The Sample units are dwellings, which are sampled in a two-stage method. First, localities are sampled from a list of localities, and second, dwellings within these localities are sampled from municipal tax files.

2.3. The Mode of Investigation

Each of the survey's dwellings is investigated four times, twice in the first two quarters of their panel, and twice in the two quarters following a two-quarter break. If a participant dwelling's residents change between investigations, the new residents are investigated according to the investigation stages. This means that some residents are investigated throughout the survey (stages 1, 2, 3, 4), and some enter the investigation in stages 2, 3, or 4.

The investigation week is determined for all four investigations and remains fixed over the entire period of investigation.

2.4. Interviewing

The interviews are conducted by about 60 trained surveyors from the CBS, under the guidance and direction of regional coordinators and headquarter supervisors.

Stages 1 and 4 of the investigation are face-to-face surveys, while stages 2 and 3 are conducted by telephone. About 10% of the interviews during the field interview stages are also conducted by telephone, mainly due to difficulties in visiting the dwelling, absence of residents, or refusals. Similarly, about 10% of the phone interviews are conducted face-to-face, mainly because of communication problems, or lack of telephones.

In residential dwellings, the surveyors interview one of the household members from whom they receive information about all members of the household.

2.5. The Questionnaire

Each household is allotted a questionnaire containing information pertaining to the entire household, and a questionnaire for each member over 15. The household questionnaire gathers information on the number of persons in the household, the number of rooms in the dwelling, etc. The individual questionnaire inquires about the state of employment, employment seeking methods, and more.

3. Transition to Computer-assisted Data Capture in the LFS

3.1. The Transition to CATI

Prior to 1999, the interviews in all stages were based on paper questionnaires.
The Labour Force Survey was the first household survey in the CBS to introduce the use of computer-assisted interviewing (CAI).

In 1999 the CBS established its telephone-based Data Collection Centre, marking the changeover to computer based data collection. Stages 2 and 3 of the LFS are now conducted by telephone using Blaise's CATI management. The field interviews of stages 1 and 4 are still conducted using paper questionnaires. The data is afterwards typed into the computer using Blaise.

3.2. CAPI Development

The next stage of development in our transition to CAI will include data collection in field interviews using CAPI. The project of the transition to CAPI is currently in development. By 2007 we expect all data collection in the LFS to be computerized.

4. Household Investigation

The LFS investigates the behaviour of households as organizational and economic units.

A household is defined by the following three components:

1. Persons who belong to the population survey.
2. Permanent residents of the dwelling.
3. Manage a common expense budget for food.

A group of persons is considered one household *only* if all three of the above components exist. A household usually comprises a family, but may also comprise a single person or include persons who have no family relationship.

MHDs are most often found among students, immigrants, or foreign workers. In MHDs each household is investigated with a separate questionnaire. One adult household member is designated as proxy and interviewed about all members of his/her household. While the sample unit is a dwelling, each household has its own questionnaire, managed separately.

Investigation results may differ between households in one MHD. Hence, one household may complete the questionnaire, while another in the same dwelling may refuse and be catalogued as a ‘non-response’.

5. Work Process Flowchart – Paper Questionnaires – Stage 1

a) Who are the permanent residents of the dwelling?

First Name	Last Name	Relationship to first member
Roger	Simpson	**First member**
Margaret	Simpson	Spouse
Bart	Simpson	Son
Wilma	Smith	Daughter
Fred	Smith	Son-in-law

b) Do they share an expense budget for food?

NO **YES**

How many households are there in this dwelling?

New tables according to Number of households in the dwelling

Continuation of Household 1 Table

First Name	Last Name	Relationship to first member	Sex	*Date of Birth*
Roger	Simpson	**First member**	male	1.1.50
Margaret	Simpson	spouse	female	1.5.46
Bart	Simpson	Son	male	24.6.75
Wilma	Smith	daughter	female	4.8.78
Fred	Smith	Son-in-law	male	6.9.80

Household 1

First Name	Last Name	Relationship to first member
Roger	Simpson	**First member**
Margaret	Simpson	Spouse
Bart	Simpson	Son

Household 2

First Name	Last Name	Relationship to first member
Wilma	Smith	**First member**
Fred	Smith	Spouse

6. Investigating Multiple Households with a Paper Questionnaire (Stages 2, 3, 4)

Stages 2 and 3 of the LFS interview by phone using a Blaise CATI management system.

6.1. Multiple Household Dwellings

Since we do not yet have a computerized system enabling the investigation of MHDs, the dwellings found in stage 1 to comprise more than one household, are investigated with paper questionnaires in the following stages, rather than by CATI. The interviewer must first inquire whether anyone has joined the dwelling or left it, and then complete the information on each household on paper questionnaires. These are later typed into Blaise. After typing first the number of households in a dwelling, a corresponding number of questionnaires will be generated by Blaise for each household.

7. Problems with the Current System

7.1 The interviewer does not follow a constant set of automatic questions.
7.2 The question: "How many households are there in this dwelling?" is vague and confusing for interviewees.
7.3 Interviewers must type information from paper questionnaires into computerized ones, increasing the possibility of error.
7.4 The use of paper questionnaires, lacks the obvious advantages of computerized data collection such as logical error checks and automatic workflow processes.
7.5 There is no methodological unity regarding the survey population and stages.

8. Developing Stage 1 Questionnaires in CAPI

The transition of stage 1 of the LFS to CAPI has not changed the survey's definitions, sample units or household investigations. The interviewers work with a computer but do not change their workflow and must still inquire about the number of households, opening separate questionnaires for each household and interviewing a representative from each.

Planning the transition to CAPI, the following rules have been established:
1. There is no change to the current definitions of a household.
2. The work process and flow must correspond to the existing LFS process.
3. The system will be user-friendly.
4. The questions and their order must be clear and intelligible for the interviewee.
5. The interview will be constant, brief, and include no long pauses.
6. Since the majority of dwellings consist of only one household, it is important to keep the interview as simple as possible.

8.1. CAPI Development

The following principles served us in the selection of development platform and languages:
- The product must be simple to use
- The product must be stable
- The product must maintain the desired level of information security.

It was decided that the interviewers laptop computers must contains a minimal number of systems. The LFS CATI version in Blaise provided the basis for

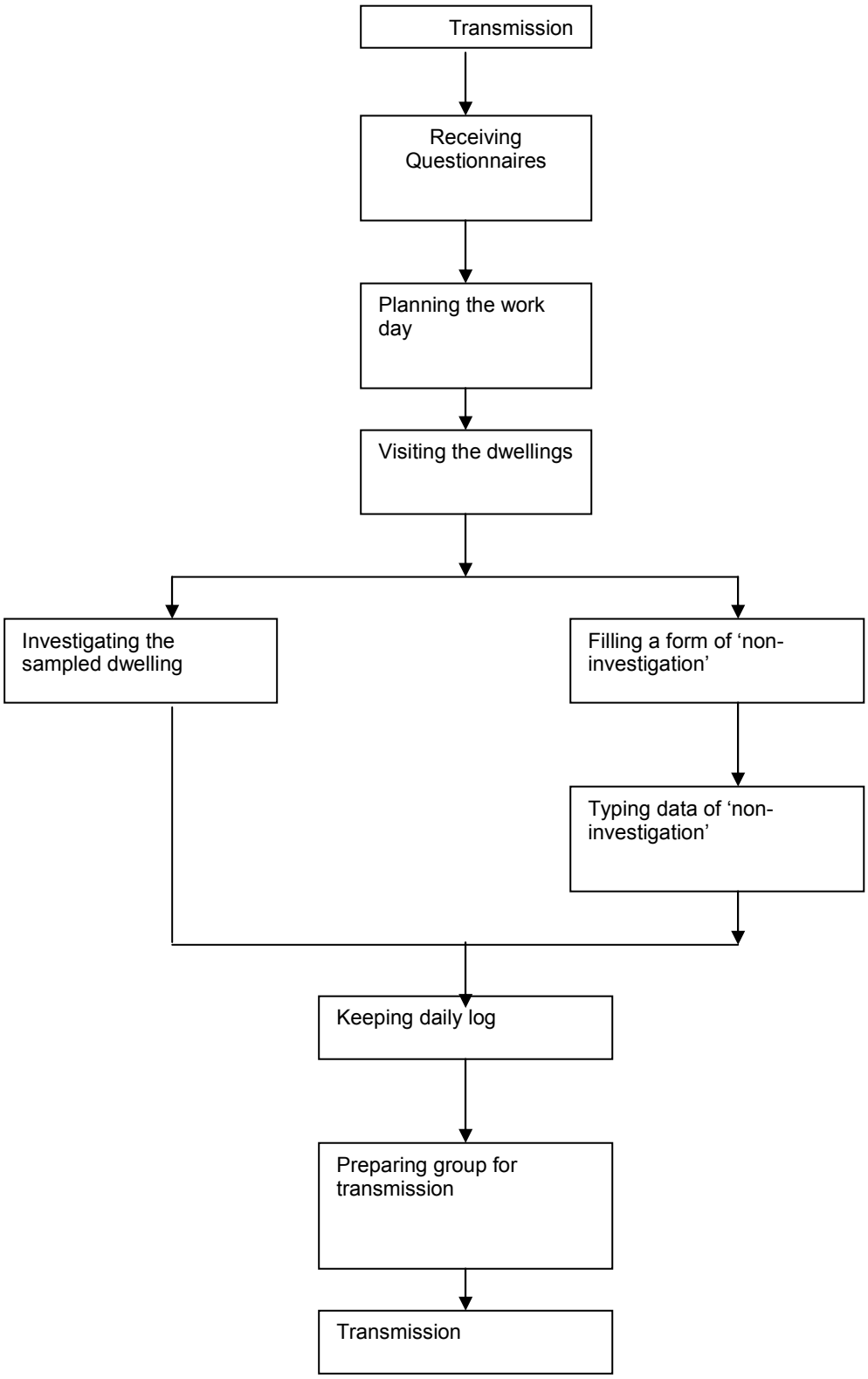
development of the CAPI version, which is why we have chosen to develop the survey management system using MANIPLUS.

8.2. Development the Interviewer Working Environment

The Interviewers' work consists of several actions:

1. Collecting data during the interview of a sampled dwelling.
2. Collecting further data in cases of 'non-investigation'.
3. Transmitting and receiving data.
4. Planning the work day.
5. Keeping a computerized log of activities and expenses.

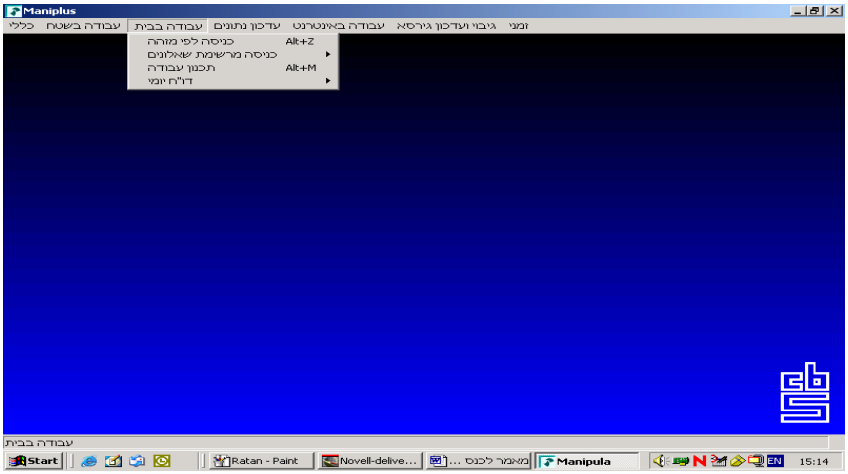
8.2.1. Interviewer Working Environment



8.3. Survey Management Menu

In order to enable the interviewer to conduct the necessary activities from home, we have developed a program including all possible activities in one menu.

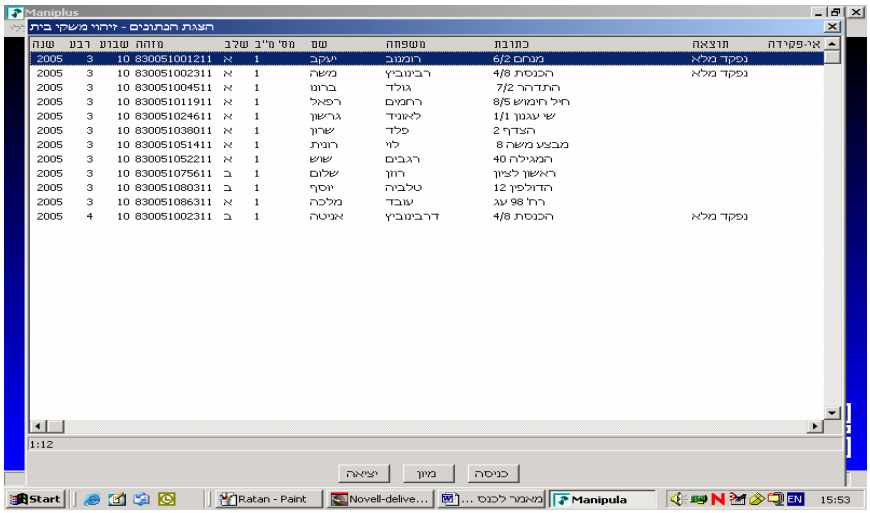
Figure 1 – Survey Management Menu



8.4. The List of Dwellings and Households

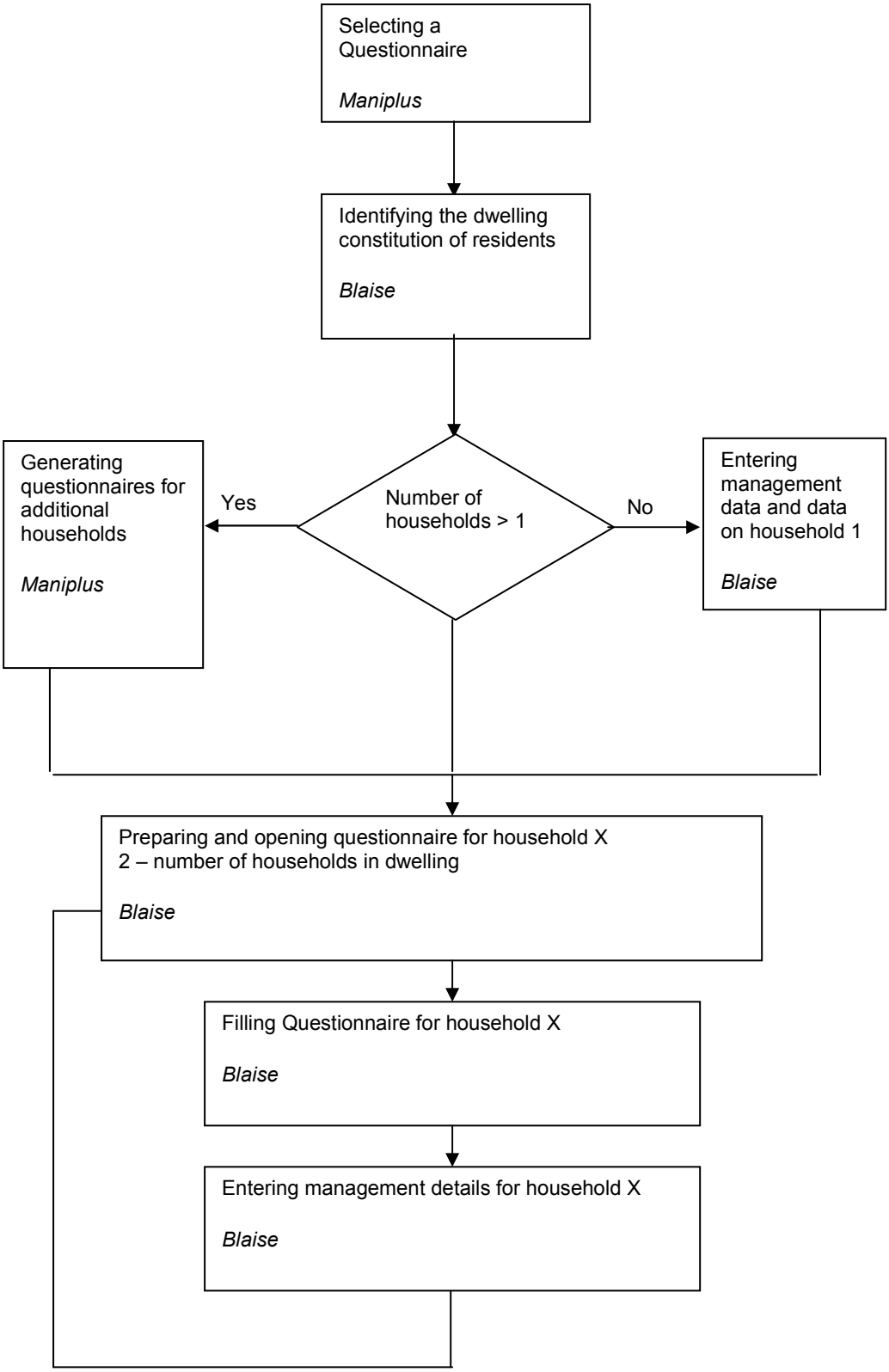
The interviewers receive weekly Blaise questionnaires of sample units. The questionnaires are generated at headquarters according to their investigation stage, either in CATI or in CAPI. The interviewer receives the questionnaires either with an empty block for dwelling constitution (including no details), or with the same block including details from the previous stage. The list of households includes the questionnaires received from headquarters, as well as those generated by the interviewer during the investigation of multiple households in the same dwelling.

Figure 2 – List of questionnaires for investigation



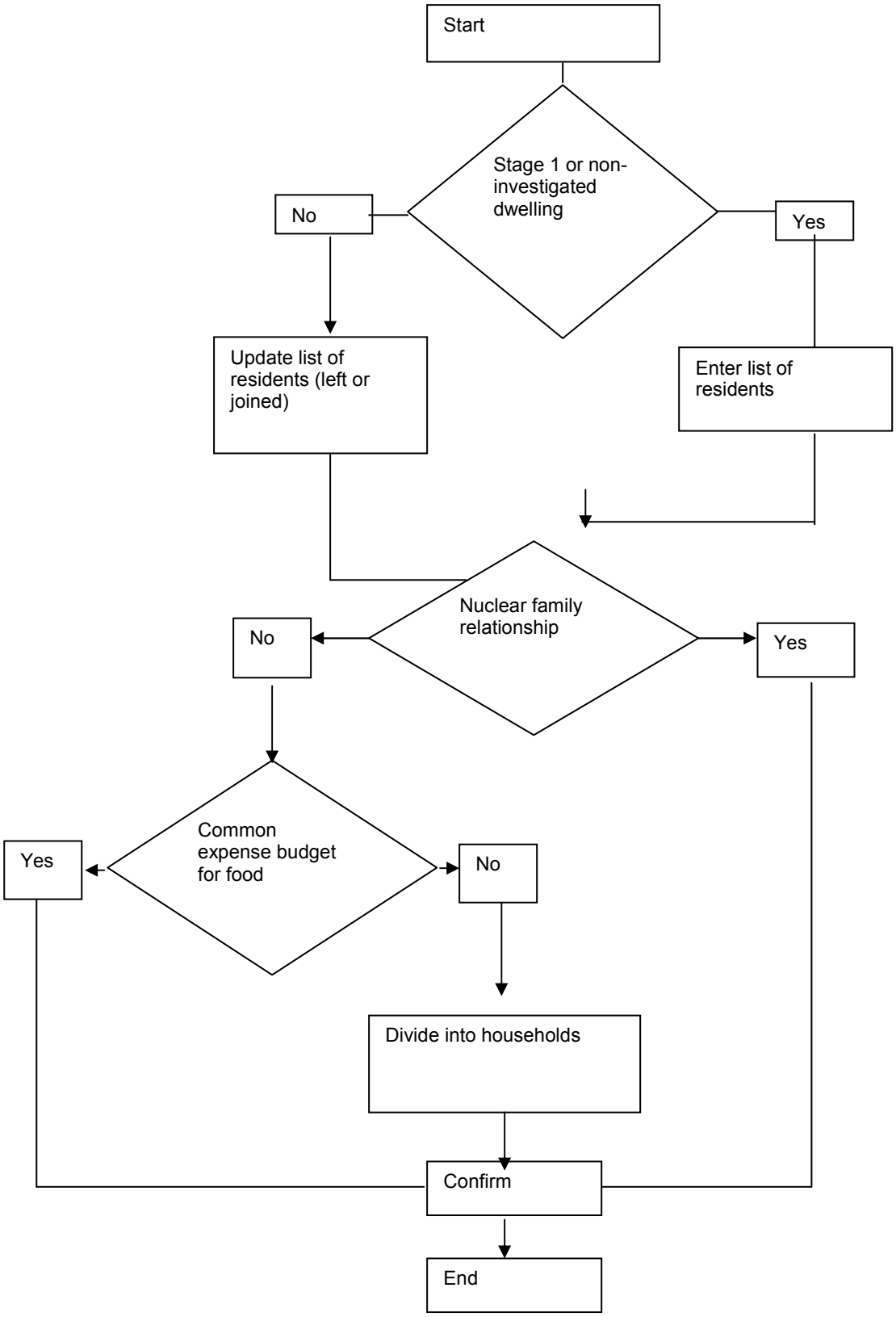
8.5. Field Investigation Process

The interviewer's investigation process of a sampled dwelling. The following chart details the interviewer's investigation process, including the development tool of each stage (Blaise or Maniplus).



8.6. Resident Identification Block

The identification block contains the list of residents in a dwelling during the period of investigation and the division into multiple households if necessary. If there is more than one household in a dwelling, the questionnaire for each household x (x=1,2…9), contains a list of residents transferred from the identification block.



8.7. Division of dwelling into multiple households

For the development of multiple household division we used SET fields. The interviewer selects members of a household from a common list of residents which s/he is shown. Those not selected from the list continue to be shown until they are selected for a different household.

Figure 3 – a list of residents

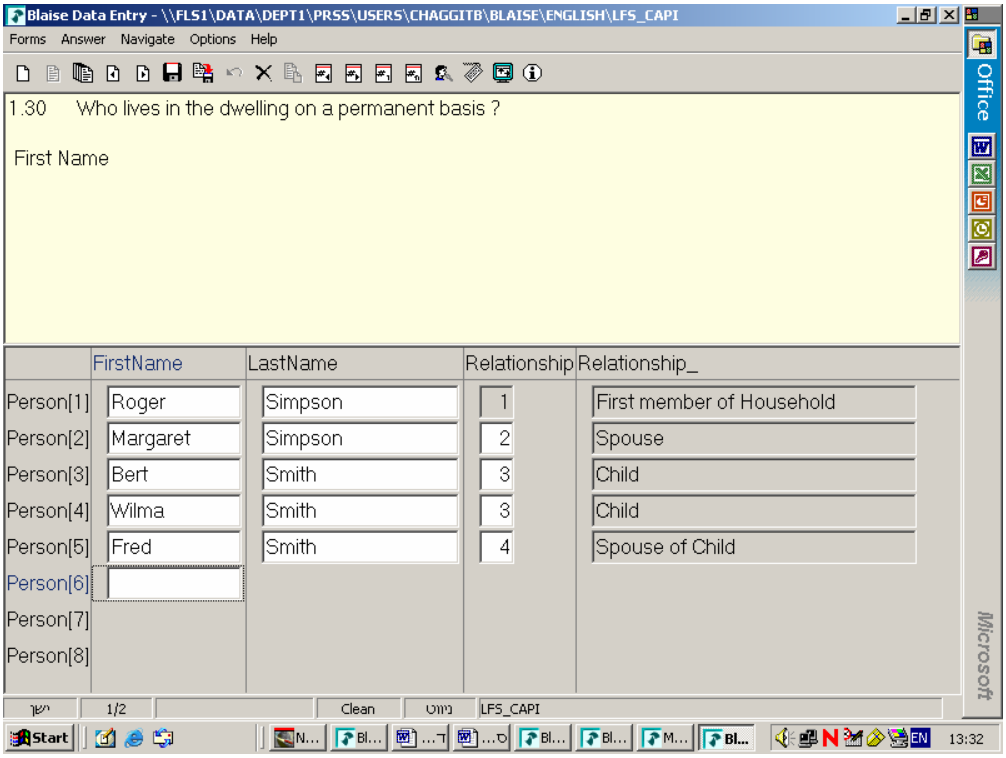


Figure 4 – question about common food budget

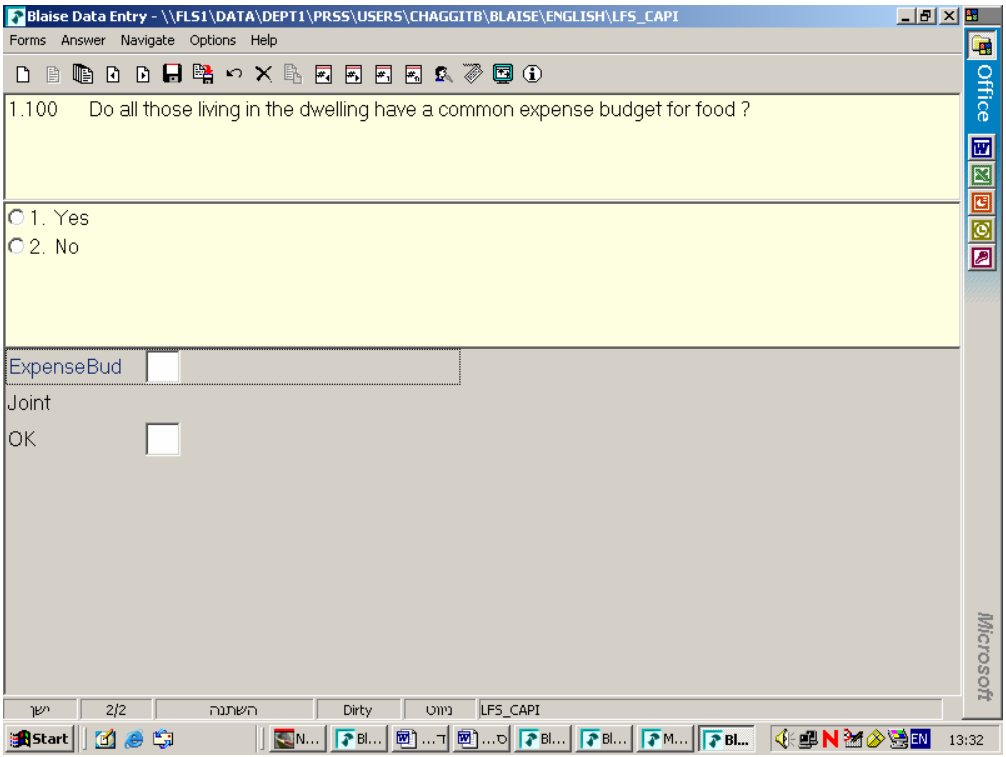


Figure 5 – division into households

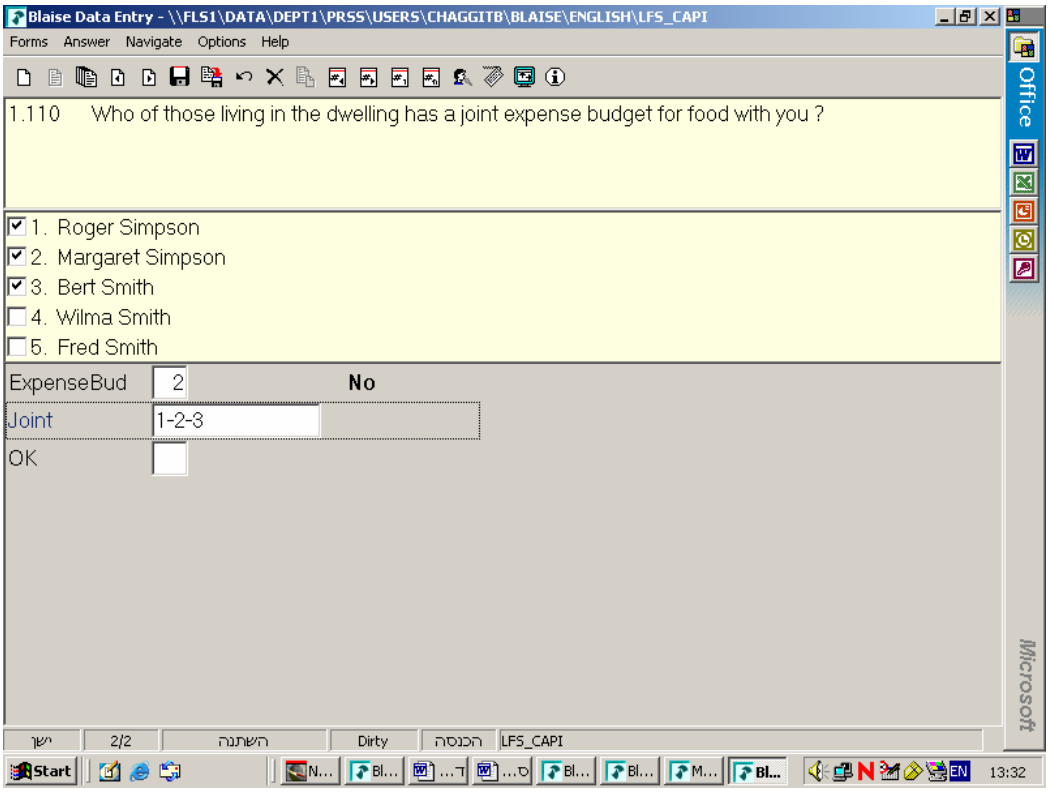
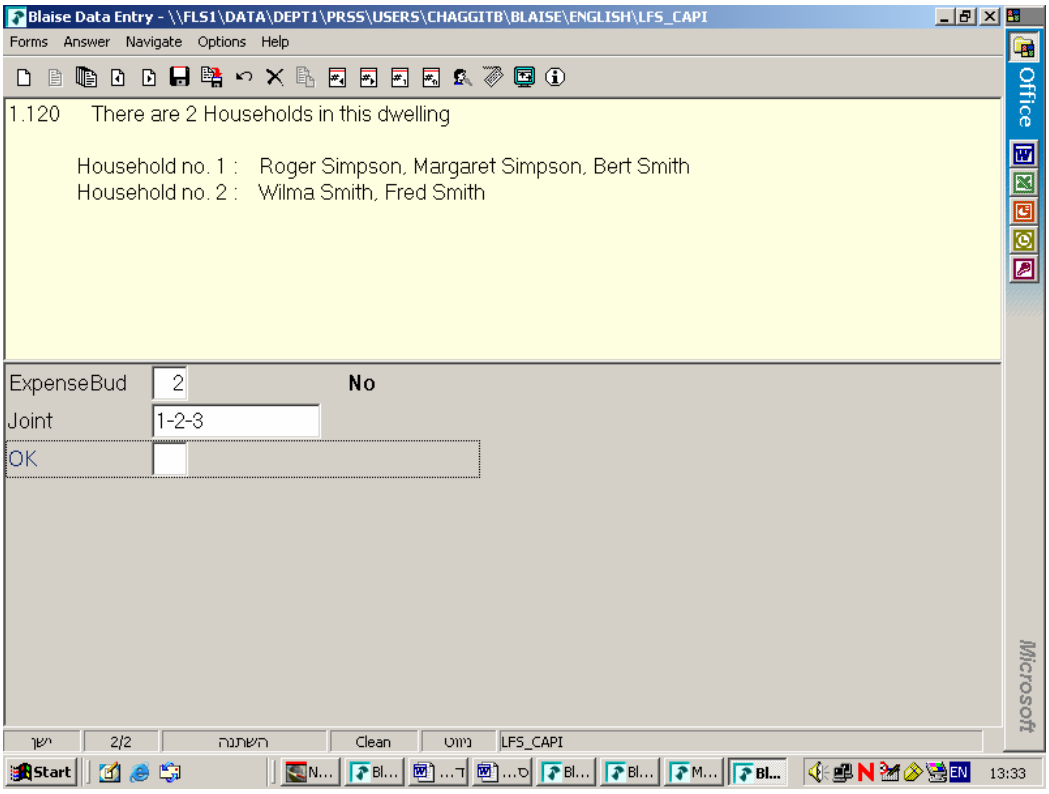


Figure 6 – confirmation of number of households



9. Conclusion

The LFS is the first of CBS's household surveys to use CAPI. Its panel system and varying methods of investigation are unique. The development of a data collection and management system for the entire survey is thus a complex task.

The need for a friendly system, able to identify multiple households in one dwelling, and to generate questionnaires accordingly, complicates the development, as well as the integration of the data collection system and the management system.

The system has been developed after several field experiments. Upon conclusion of development for all stages of the survey, the system should become available for production around October 2006.

Listing Part 2 - Using Blaise 4.7 for Coding Listing Instruments*

Malcolm Robert Wallace, Roberto V. Picha & Michael K. Mangiapane (U.S. Census Bureau)

1. Overview

The U.S. Census Bureau conducts many listing operations using computer assisted interviewing (CAI) instruments (referred to as listing instruments), as a means of obtaining sample for many of the surveys that it conducts. At the 2004 IBUC, the Census Bureau presented a listing paper that discussed different listing operations conducted by the Census Bureau, the unique challenges and functionality required by listing instruments, and how some of these requirements were addressed with Blaise 4.5. Since the 2004 IBUC, the Census Bureau has begun work on converting our two oldest and most challenging listing instruments – Permit Address Listing (PAL) and the Survey of Construction (SOC). This paper will discuss how the Census Bureau plans to use Blaise 4.7 to address some of the more challenging functionality required by these two surveys.

1.1. Paper Objectives

This paper will provide some background on the PAL and SOC listing operations, review some of the listing requirements requested, and discuss how we have addressed (or plan to address) these requirements using Blaise 4.7. This paper also includes some code examples.

1.2. What is a Listing Instrument?

A listing instrument is a CAI instrument that is used as a tool by the Field Representatives (FRs) to collect, and sometimes 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. Once the listing is complete, sample is either generated by the listing instrument, or back at Census Headquarters (HQ) after the listing data transmitted in.

1.3. Permit Address Listing (PAL)

The PAL survey is a monthly listing survey that has been in production in CAPI since 1996. The PAL instrument collects building permit information for new residential construction. 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 is a "true" listing instrument in that there is very little "interviewing" done with this instrument. The FR simply enters the listing instrument and starts keying data.

The data collected by the instrument is used to update the new construction sampling frame for the Census Bureau's current surveys.

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.

---

* ***Disclaimer:*** *The views expressed in this paper are those of the authors and not necessarily those of the Census Bureau.*

1.4. Survey of Construction (SOC)

The Survey of Construction is a monthly listing and interviewing survey that has been in production in CAPI since 1995. The data from SOC is used to produce the monthly housing starts statistics. The SOC listing instrument also collects permit information from building permit offices. It gathers the permit number, address information, and builder information. In addition to 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. Once the listing is complete, the FRs conduct interviewing using this other interviewing instrument.

1.5. PAL and SOC Redesign Efforts

Both the PAL and SOC listing instruments are currently written in Clipper and must be converted to Blaise. Since these two operations are similar, in that they both collect lists of data from building permit offices, we decided to combine their conversion efforts together and discuss requirements for both at the same time. Although both of these listing applications involve the listing of permit information, they also contain very unique and survey-specific functionality that did not allow us to combine them into one instrument. For example, the SOC listing instrument (LI) requires a complex sampling algorithm that is not needed by PAL. It also lists a lot of information that is not needed by PAL – such as builder contact information and specific information from the permit. PAL on the other hand requires two types of listing tables – one that lists all of the permit information and one that has some information already pre-filled which then requires special editing and handling.

Sponsors of both surveys were brought into the same workshop so common requirements could be addressed together.

1.6. Requirements Workshop

We held a week long requirements gathering workshop. The workshop participants included people from the Technologies Management Office (TMO) authoring and case management systems staffs, the Field staff (SOC/PAL supervisors, a senior Field Representative (SFR), and HQ staff), and the sponsoring divisions. During the workshop we reviewed the current system and defined the requirements that would be needed for the new, redesigned system. This included adding new functionality to the system that the sponsors/Field wanted and removing functionality that was no longer needed. It also included modifying existing functionality in order for it to work with Blaise. We also made an attempt to standardize similar requirements for both instruments. For example, we agreed upon standard field lengths for common fields and the address format for the permits and building permit offices (BPOs). Requirements and outstanding issues were documented and distributed to the design team. These requirements were also used as input into the case management systems requirements workshop that was held at a later date.

2. Functionality Requested for the SOC and PAL Listing Instruments

The concept of a listing seems quite simple. One would think that you could just create a table and key in some information. Of course, nothing is ever as easy as it sounds (or as it should be). This section documents some of the different requirements requested by our sponsors and Field staff.

2.1. Combine Questionnaire into Listing Instrument.

The current SOC listing operation uses 4 different instruments:

1. Listing Instrument (LI) – the Clipper program that collects the permit information
2. Questionnaire for Building Permit Officials (QBPO) – CASES instrument that collects information about the permit office. For example, it collects the location of permits, permit filing information, and information on the BPO boundary.
3. Little Questionnaire Instrument (LQI) – CASES instrument that collects more detailed information from the permit. For example, the lot size, sale price, number of rooms, etc.
4. Builder Table (BT) program – Clipper program that helps the FR manage their builder information

For the SOC redesign we are able to combine three of these programs into a single Blaise listing instrument – the LI, QBPO, and LQI. Since all three instruments are conducted at the permit office it makes sense to combine them into one program. This is one of the advantages of using Blaise 4.7 for the SOC LI – we can include a questionnaire within the LI.

2.2. Call an External Program and Insert Data Into Listing

During the SOC listing process, the FRs must collect the builder name, address, and contact information for each listed line (building) that falls into sample. To help the FRs keep track of the builders in their assignment area, they store the name and contact information in a "Builder Table" which acts like an address book. It is basically a database with the contact information the FRs need to complete the interviewing portion of their work. Instead of having to key in the builder name, address, and contact information for every sample line, the FRs only need to key this information once (into their Builder Table) and then select a builder contact from the BT and have the instrument automatically fill the data for these fields. This is functionality that they currently enjoy and must continue to have available when we convert to Blaise. The other requirement with this application is that the case management system also needs access to the BT database so that program can get the most up-to-date information as well.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.3. Creating Look-up Tables on the Fly

Since PAL and SOC both involve collecting addresses for new construction, many of the street names keyed do not already exist in some type of street name file for the BPO. However, the sponsors would still like to have a street name pick-list for the FRs to use during the listing. They want this street name pick-list to contain all of the street names listed for the month. Since builders often take permits out in groups, it is likely that there will be multiple permits issued for the same street name. The sponsors would like the FRs to use this street name pick list so that street names are listed consistently throughout the listing.

Similarly, the sponsors have requested a ZIP Code/Locality pick-list also be created "on the fly" so that ZIP codes and localities are listed consistently.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.4. Toggle between "Line View" and "Full View"

When thinking of a listing table, one thinks of the "Line View" display. This display simply collects data one row after the other (with many rows displayed on the screen at the same time) with columns of the row scrolling off to the right of the screen. Since the PAL LI collects a large amount of data, the sponsor has requested that we allow the FRs the option to enter their data all on one row ("Line View" with scrolling and many rows on the screen at one time) or to enter their data all in one screen, a "Full View" that displays all of the data for one permit with no scrolling.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.5. Using Different Listing Tables Based on FR's Preference

For the SOC LI, the FRs must list "additional information" for listed lines that fall into sample. This information includes the address of the permit, the builder / contract information, and some details on the structure (from the permit). Since permit offices collect and store their permit information in different ways, the sponsor requested that we give the FRs two options for collecting the additional information:

1. Collect additional information while listing. This means that all of the fields come on route if the permit falls into initial sample. The FRs would choose this option if it would be hard for them to go back and find the permit information after final listing is complete and sampling is run.

2. Collect additional information after listing is complete and final sampling is run. This means these fields do not come on route during initial listing. The FRs would pick this option if it would be fairly simple for them to go back and gather the additional information needed for sample permits. This option would require less keying by the FRs since Final sampling will screen out some of the initially sampled permits.

This paper will discuss how this functionality was addressed with Blaise 4.7.

2.6. Free Form Navigation

Blaise 4.7 Free Form Navigation (FFN) has been very useful in our design of the PAL listing instrument. It has helped with many of our navigation requirements. Listed below are some of the navigation requirements we have for our listing table along with how we are planning to handle these with Blaise 4.7.

- **"Force" some Fields to be "Must Enter" in the FFN Table.** Ideally, we would like to be able to control whether the fields "must enter" requirement is enforced while in the FFN table. For example, every listing line in the table must have a permit number or there would be no reason to have that record. This currently can't be done with the Blaise FFN (and may not make sense to do), but we have a work around for this requirement. To address this particular requirement for PAL, we do not bring the rest of the listing line on route until a permit number is keyed in.

- **Run Field Edits when data is entered into a Field**. If data is entered into a field, then the edits on this field should be run at that time. So, if the FR enters in an invalid date, the date range edit should pop immediately. (Blaise 4.7 now does this.)

- **Run "Error Involving" Edits when data is entered into a Field**. As with regular edits, edits on multiple fields specified in an "error involving" edit

should be run when the data for all fields involved is entered. For example, if the FR enters a month = June and day = 31, the edit check should run and pop an error as soon as "31" is entered in the "day" field. (Blaise 4.7 now does this.)

- **Allow for "must enter" fields to remain empty when exiting the FFN table**. One of the functional requirements for the PAL LI is to allow the FRs to key in partial permit data and exit the listing table (without popping an error). For example, it would be acceptable for the PAL FR to list the permit information, but not complete the ZIP code/locality information until a later time (maybe at home) since the permits may not contain this information. This means the FR would have to exit the case, leaving the data unfilled and the case as a partial, and then re-enter the case and complete the listing.

  Blaise 4.7 allows us to map a function key (F10 in our case) that takes the instrument to a parallel block (the back of the instrument) and will not enforce the "must enter" requirement for fields when doing this. This is all right since we would code this case as a partial when the FR exits in this manner. In order to complete the listing, we plan to have the FR exit through a "listing complete" screen which will then verify "must enter" fields are completed by using Maniplus to run edits on them.

Blaise 4.7 FFN will automatically enforce the "no empty" check if the user exits the table by completing the last possible line in the table or when moving to a new page. In this case, we would like Blaise to allow us to have some control over the error messages that are displayed to the FR in the Dialog box.

2.7. Exiting the Listing Table Instrument

This seems like a simple item, but of course there is a catch. Our current standard for exiting a table is to have the FR enter "999" in the first column of the last row. This doesn't work for listing since "999" could potentially be a valid entry for the permit number field. Since Blaise 4.7 allows us to create button bars, we went ahead and created a "Done" button for the FRs to use when they are finished with the listing. This button is associated with an action that will start a parallel block in the back and take the FRs to an "Is your listing complete" question. If they answer "yes", we call a Maniplus script that will run edits on the listed data to verify all "must enter" fields are complete and to verify the information listed passes various form-end edits. If they answer "no", we return them to the listing table.

3. Calling an External Program and Inserting Data into Listing

Blaise lookup table capabilities provide an excellent way to call external files and insert data into the instrument. It is fast, reliable and easy to use. However, in order to meet our sponsor's requirements for their Builder Table (BT) application, we need to be able to interactively add, edit, and select information from this external file. We must do all these operations while inside the DEP. Since this Blaise external file does not provide the desired flexibility, we had to figure out another way to accomplish our sponsor's requirements.

Blaise 4.7 is a more robust version than its predecessor. Data operations generally done outside the DEP are now possible while in it. Blaise 4.7 now provides us a way to achieve the sponsor's requirements by allowing us to use a combination of Manipula and/or Maniplus that will enhance Blaise capabilities for data collection.

The biggest challenge we have with the SOC listing operation is the interaction between the listing instrument and the builder table application. The BT

application is a separate program that can be accessed from multiple instruments and from the laptop case management system. It would typically be called multiple times from within the LI while the FRs are collecting data. The BT data is sent down to the laptop as a separate input file and will be stored in an Oracle database.

3.1. Possible Approaches

We discussed 3 possible approaches for addressing this requirement:

1. **Writing the BT Application in Blaise and Using ASCII Files** - This process would involve retrieving/updating data from the Oracle database into an external Blaise database and vice versa, by using ASCII files. Although this process could have been done in our current environment, we were concerned about the additional time needed for launching the instrument.

2. **Writing the BT Application in Blaise and Using BOI files** - This approach would be to use BOI to directly connect to the Oracle database, extract table information, and load it into a temporary Blaise database. Updates to Oracle database from the listing instrument can be done via BOI.

   Research concluded that this was possible; however, our prototype indicated that there was a significant delay connecting to the database (we used Access to mimic an Oracle database for our testing). This was a delay from within the DEP at the listing table and would not be acceptable for the FRs.

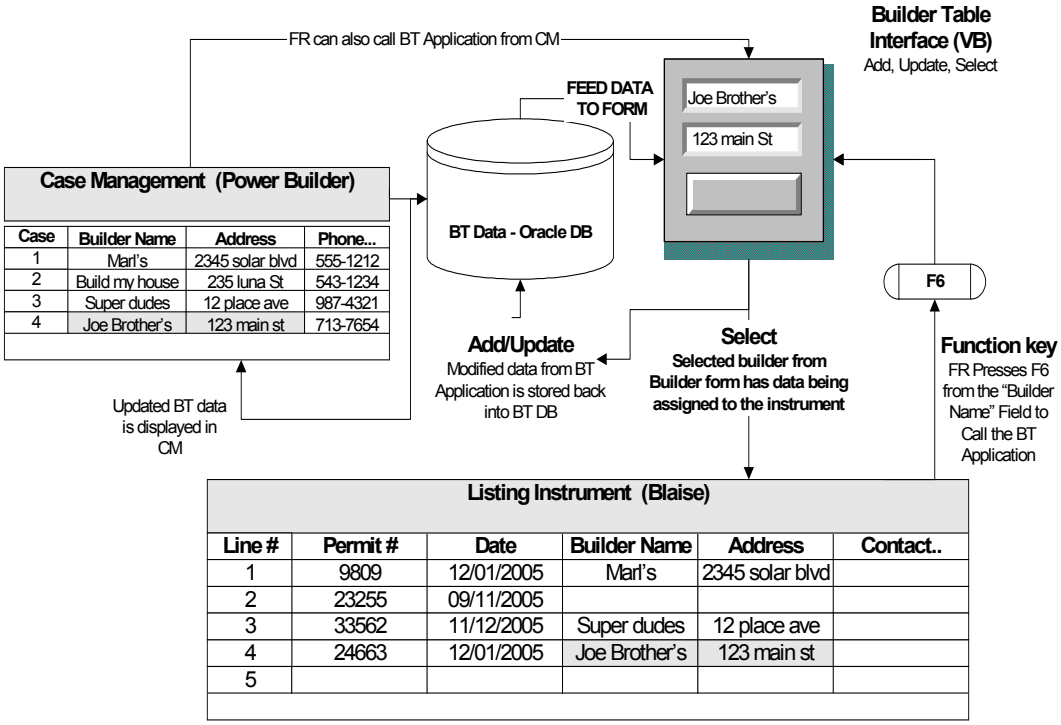
3. **Writing the BT Application in VB and Using the BCP -** For this approach, we plan to build a VB application that will connect to the Oracle database and allow user interface. This application will be called our Builder Table program and will allow users to add, edit, and select (transfer the data for the selected record down to the DEP – into the listing table). The BCP will be our mechanism to allow the DEP and VB program to interact.

   By using this approach, we accomplish two objectives: 1) allowing our case management system to interface with the BT data and 2) allowing two different instruments to interface with the BT data. Our initial testing indicates a fast response time.

   For us, this appears to be a feasible solution. For our case management system, this application can directly connect to the Oracle database on the laptop. For our instruments, the application can be "wrapped" via the BCP so the instrument can actually interact dynamically by transposing data to the DEP. This application would be converted into a wrapper DLL and installed on the laptops.

   The Builder Table interface, whether called from the case management system or the instrument, will retrieve information from the Oracle database and display it to the user based on parameters passed into the application. The updated information is then stored back to the Oracle DB so that the next time the application is called, the latest information is available for both case management and the instruments.

3.2. Data Flow of the Builder Table Application



Case	Builder Name	Address	Phone...
1	Mari's	2345 solar blvd	555-1212
2	Build my house	235 luna St	543-1234
3	Super dudes	12 place ave	987-4321
4	Joe Brother's	123 main st	713-7654

Line #	Permit #	Date	Builder Name	Address	Contact..
1	9809	12/01/2005	Mari's	2345 solar blvd	
2	23255	09/11/2005			
3	33562	11/12/2005	Super dudes	12 place ave	
4	24663	12/01/2005	Joe Brother's	123 main st	
5					

4. Creating Lookup Tables on the Fly

When listing information for SOC and PAL, some information such as the city and ZIP code or full street name of an address may be the same for multiple lines of the listing. It would be time consuming to the FR if they have to re-type this information over and over again, especially if the city or street name is long. This could also lead to keying errors and inconsistencies with the data. For those reasons it would be useful if the user could use an external lookup table to enter the information into the listing.

4.1. Previous Lookup Tables

Previous versions of Blaise have allowed for a static external lookup tables to be defined and used in the instrument. Data could be passed from the lookup to the instrument, but data couldn't be passed from the instrument back into the table. We could potentially send out a lookup table with the instrument to the FRs, but there would be major problems with taking this type of approach. We would have to either send out an extremely large table with the instrument that could require a lot of time to search through to find the correct data, or we would have to tailor a lookup table for each FR that conducts these surveys. With thousands of CAPI assignments sent out across the country each month, tailoring a lookup table for each assignment is not feasible. The other problem is that this table would most likely need updating by the FRs since they are listing newly created streets. If the FR needs to add data to the table or update data that is already in the table, they could not do it themselves. Instead, some sort of update process would have to be created by HQ. If the FRs could not update the table, then it would not be as useful during their listing.

4.2. Using Blaise 4.7 for Dynamic Lookup

An idea that was proposed was to use a dynamic lookup table that could be changed by the FR. With the release of Blaise 4.7, a way to have such a table was created. By calling a Maniplus script that uses the INTERCHANGE setting for the DEP, it allows data to be passed from a currently running DEP session to another Blaise table. Since we can call a Maniplus script as a menu item from the DEP menu, a button in the DEP button bar, or as an action button in the field, it is now possible to have the dynamic lookup table we desire. It also reduces the amount of data we send out to the field because we can simply send out an empty table and let the FR fill it at their own discretion for the survey. Since we don't have to get the table back from the FR, it lessens the amount of data that comes back to headquarters that would require some sort of processing.

4.3. Creating the Table

In order to accomplish this task, a table needs to be created that will hold the data for the lookup. It is a simple process of creating a datamodel that is separate from the listing instrument with the fields you want to hold data inside the table. Make sure the field type inside the table exactly matches the field type in the instrument to prevent any errors in the data. This is especially important for string fields. After you compile and run the datamodel to create the Blaise database, the **minimum** files you need are the BDB, BDM, BFI, and BJK files to have a working lookup table.

See **Example 1** in the Appendix for a datamodel example.

4.4. Creating the Maniplus Script

A Maniplus script is needed to tie the lookup table and instrument together. The nice thing about the INTERCHANGE setting is that it can be used for a TEMPORARYFILE, which is required to allow a DEP session to exchange data with the lookup table. Since we are using the current instrument DEP session, fully qualified field names from the DEP are allowed inside the script. You may share data directly between the DEP and the lookup table, or you may create auxfields to hold the data from the DEP before passing it on to the table. Choosing to use auxfields to hold the data is a good idea for debugging when working with Maniplus. It is recommended that the lookup table be set as an UPDATEFILE with the settings OPEN = YES, CONNECT = NO, and MAKENEWFILE = NO. This allows the table to be created and updated, and it will not erase all of the data when changes are made.

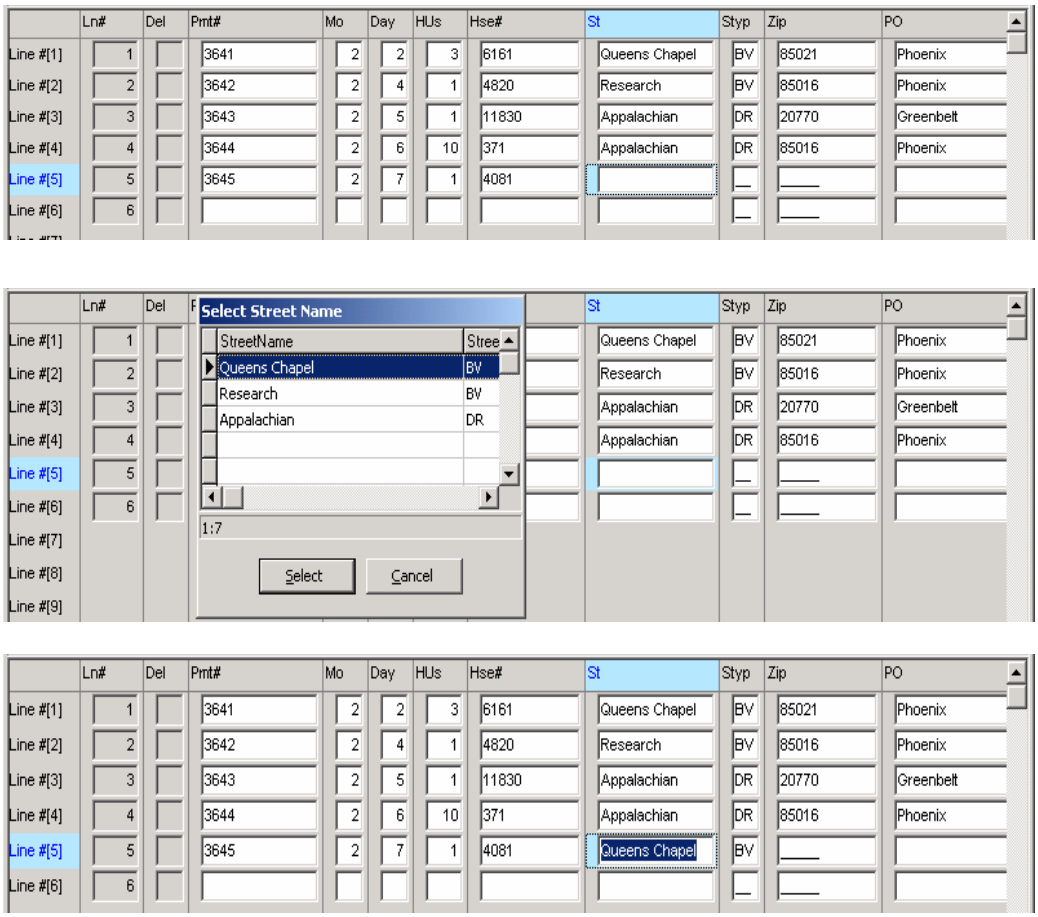
By using the ACTIVEFIELD, SUBSTRING, and POSITION functions in Maniplus we can pinpoint where (which field) in the listing the FR is at so that the script exchanges the correct data. Since the listing table in SOC is set up as an array of blocks, the field looks like this:

"TableName.BlockName[position].FieldName." We can use a substring that starts from the position + 2 of the right bracket to get the field name, and we can create another substring that finds the positions of the array position brackets and reads the text in between them to get the listing line the FR is on.

4.5. Retrieving Data from the Lookup

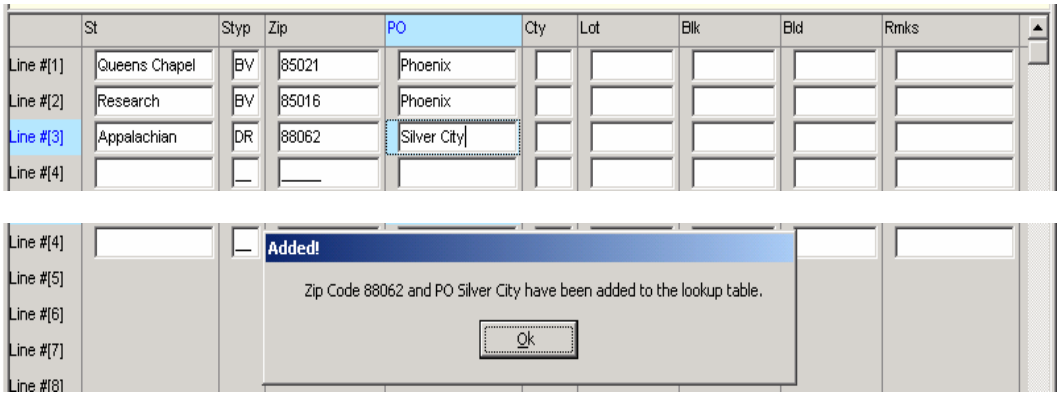
The FR will use a function key combination (Ctrl+L, for "list", in our case) to call the lookup table. A DIALOGBOX is used to hold the lookup table and it has two buttons, one to select the entry the FR decides to use and the other to cancel the lookup if they decide not to use any of the table data. When the user selects a line

in the lookup table and presses select, the data in the table is read and passed to the fields specified in the script to the DEP.



4.6. Adding Data to the Lookup

When the FR runs the Maniplus script (by pressing "Ctrl +A", for Add in our case) to add data to the lookup table, the data inside the fields that are selected are captured and put into auxfields in the script. The data is copied from the auxfields to the lookup table, and it is written to the table. We have also set up a DIALOGBOX to pop up and tell the FR that the information was added and what the information was.



4.7. Accessing the Maniplus Scripts

Blaise 4.7 now allows Maniplus to be called as part of the actions available from the DEP Menu (both menu items and button bar buttons) and the Action Dropdown in the Datamodel Properties. Since the actions dialog calls Maniplus via a command line, we can use parameters to control how the script is run. This has given us the ability to have the code that reads data from the lookup table and the code that adds data to the lookup table reside inside the same script. In our instruments, we have set the parameter of 'L' to perform the lookup and 'A' to add to the lookup table. To go even further, Blaise can look at the name of the field that the FR is on when the script is called (based on ACTIVEFIELD) and read from or write to the appropriate table. This is beneficial in that the same function key combination (Ctrl + L for us) can be used to call different dynamic lookup tables since only one script is needed to access all of the tables. A standard function key makes this functionality easier on the FRs.

See Example 2 in the Appendix for an example of a Maniplus script with parameters controlling the flow of data

5. Toggle Between "Line View" and "Full View"

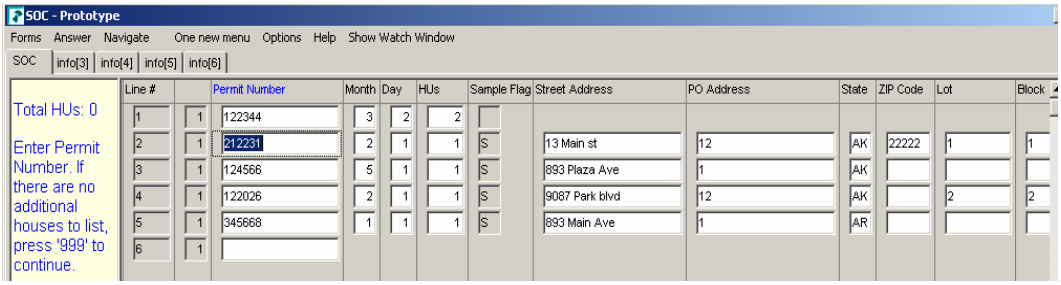
Due to the number of variables displayed/collected in the row of a table (about 25 variable per row), we have been required to allow for a "Full View" display/collection screen. The Sponsor requested that this option be a "Toggle" that goes from "Line View" to "Full View" and vice versa. This option should allow the FR to collect data in both views and toggle at any time.

5.1. Initial Approach – Setting Block Row Inside Table as Parallel

The toggle requirement can be achieved by setting the block row inside the table as parallel. When we initially tried this inside the Listing instrument we observed some degradation in the performance from field to field. This was mainly because we were setting a large number of rows (two thousand rows) as parallel. Additionally, there was no way to associate each tab to one function key so it would be very hard to use this approach.

```
PARALLEL
Info=Listing2.Blisting2
```

Where Info is set as parallel and where Blisting2 is an arrayed block up to 2,000 rows. Since a normal listing operation ranges from 800 to 1500 permits, the amount of tabs would be hard to visualize and cumbersome to the FR. See the graphic below.



5.2. Better Approach – Use Maniplus

After consultation with Statistics Netherlands, the approach we decided to use to implement the "toggle" functionality in a more robust manner was to use Maniplus.

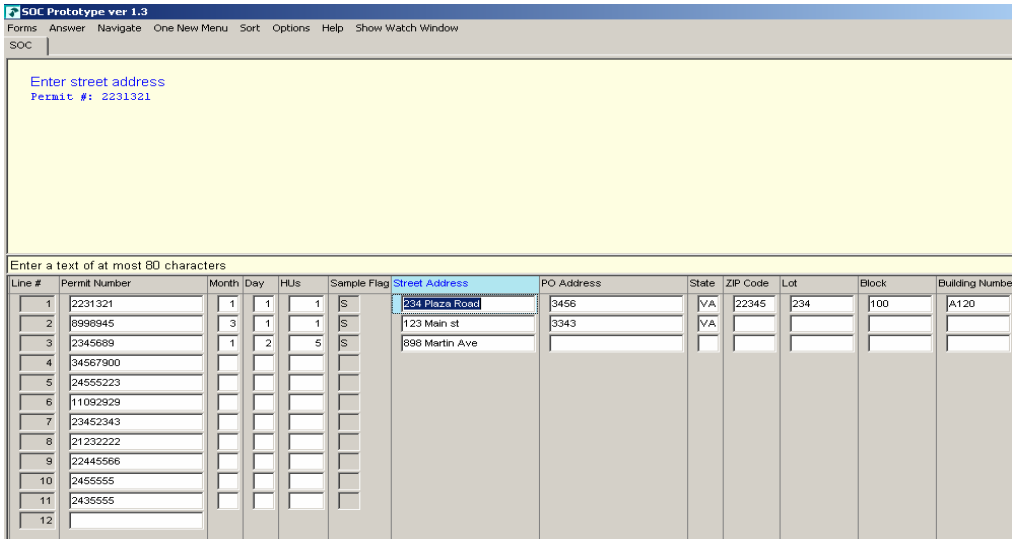
The steps to do this are as follows:

1. Create an instance of the block used in the table and set this new Block as parallel.

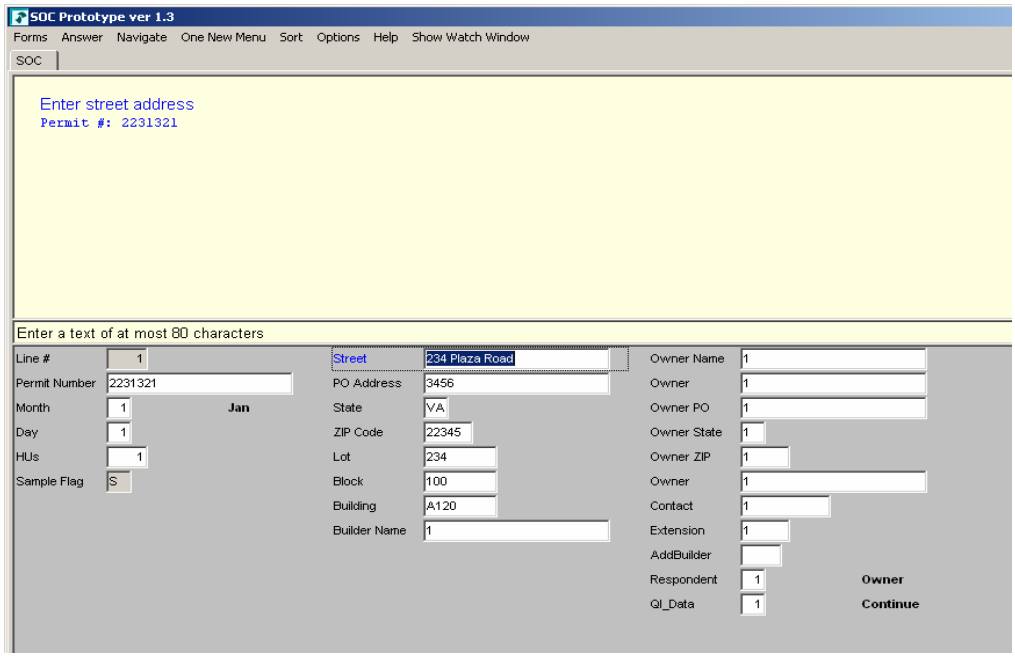
2. Create a Function Key (button) and associate it with an event.

3. Have this "event" call a Maniplus script that would capture the data of the arrayed block from which it was invoked (its instance or line row) and transfer to the parallel block defined. Then set the focus to this parallel via the Maniplus script.

4. In order to return to the main path - either by using a function key, button, or at the end of the parallel block - another event would take place using Maniplus script sending data from the current parallel block to the arrayed block.



Example of the "Full View" After Executing the Toggle:



See **Example 4** of the Appendix for an example of code that shows the "Line View" to "Full View" switch.

6. Using Different Listing Tables Based on FR Preference

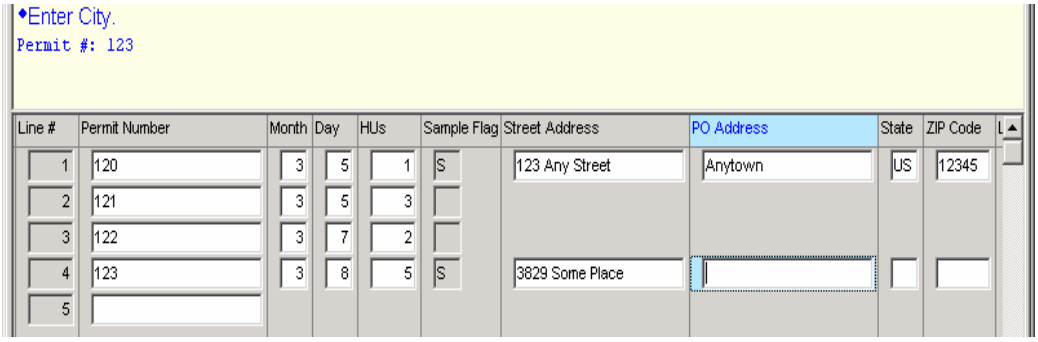
In the SOC Listing Instrument, not all permits that are listed are going to fall into a final sample that requires "additional information" on the permit to be collected. As discussed earlier, the sponsor has a requirement to allow the FR to choose how they wish to collect the permit information:

1. Collect minimal information for every permit first (Permit Number, Month/Day Issue, and # of HUs) and then collect "additional information" after "final" sampling has been run for only those permits that fall into final sample.

2. Collect all information needed (including additional information) for permits that may fall into final sample while they have the permit in hand. Once "final" sample is run, some of those permits will fall out of sample.

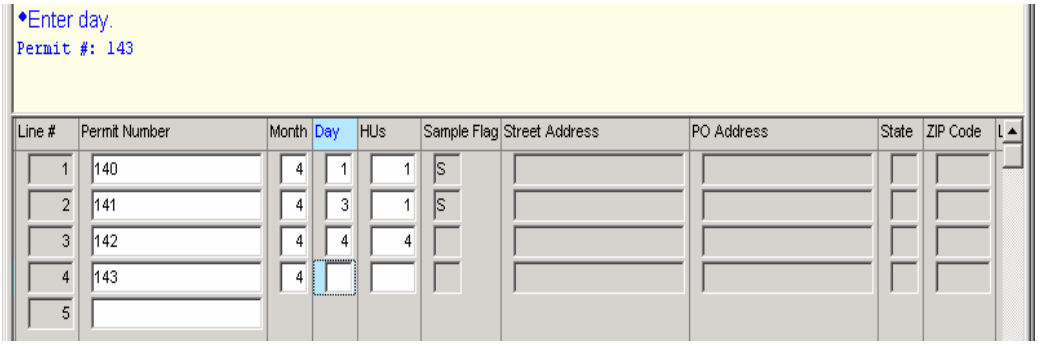
Through the use of flags, we can set the behavior of the listing table to collect the full permit information now, or to collect the information after we run our final sampling. In the beginning of the interview, the FR is asked if they want to collect additional permit information while they are listing or after they are done listing.

If they decide to collect the additional information during listing, the table will collect the permit number, permit month, permit day, and number of housing units for each permit line. If the permit falls into a potential sample, additional information questions are brought on path to be answered by the FR.



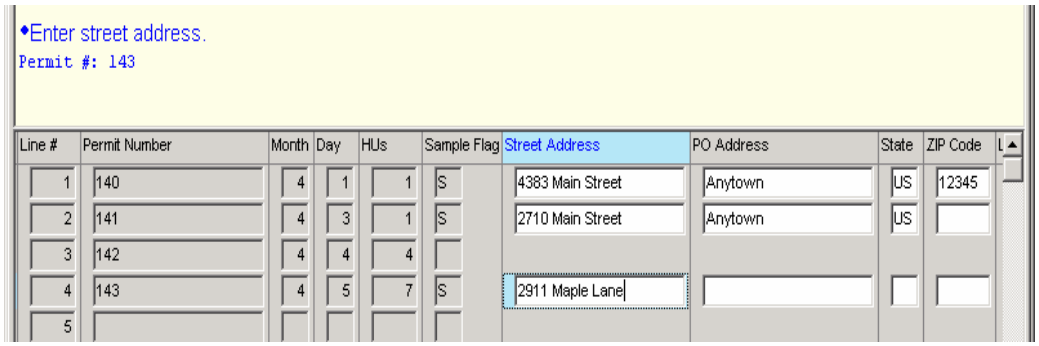
After listing is complete, the final sampling is run and since the additional information is already collected, the user can continue moving through the survey.

If the user opts to collect additional information after completing the listing, they will enter the permit number, permit month, permit day, and number of housing units for each line, but no additional information will be collected until after the final sampling is run.



When the user leaves the listing table they are asked if the listing is complete. If they answer "no", they would return to the listing table. If they enter "yes", the

final sampling is run. After final sampling is run, the additional information questions are brought on path.



After entering the additional information, the FR will move on through the rest of the survey.

The code to accomplish this process needs one flag to help set the path; the rest of the pathing is based on answers given by the FR.

See **Example 3** in Appendix for an example of how to accomplish this task.

7. Controlling When to Pop-up a Lookup Table

7.1. Why We Want to Control This

Normally, if an external lookup table is used in the instrument, the lookup is set to pop up when the FR begins entering data into the field. Typically, these lookup lists are only called one time in an instrument so it makes sense to have it automatically pop up.

For listing instruments, however, this approach does not make as much sense. It would mean that the lookup list would pop up for every line listed. For example, if a street type lookup list is used and the FR listed 400 lines of data, the FR would see this list pop 400 times. Since the FRs conduct many listing assignments, they will most likely already know most of the common street type abbreviations to use. But, for times when there is an unusual street type, the sponsor would like the FRs to have the ability to select that street type from a list.

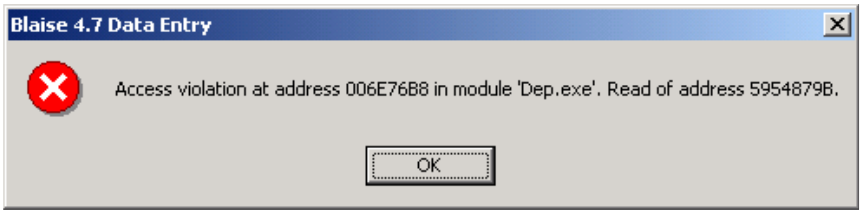
7.2. The Approach

Our first approach to controlling how the lookup table behaves was to turn on the "Use Normal Asker" option in the ModeLib editor and then set a custom startup behavior for the asker in that field. In our case we used the pressing of the "Insert" key to pop up the lookup table. This meant that fields using a lookup had to be set up as a type in order to show up in the Datamodel properties. This approach worked until a problem developed with using external lookup tables in the rules.

Our workaround for this problem was to call the lookup tables from Maniplus and use a DIALOGBOX, but that meant that Maniplus needed to be called from within the instrument. This was accomplished by using a menu item that was enabled only when the cursor was on a certain field ($FIELDTAG = 'FieldTagName'). We went further and created an action button in the field as well. The usage of this button worked okay, but it would force our FRs to use a mouse to click on the button. The issue here is that we don't like to require the FRs to use the mouse pad to conduct interviewing so we ended up taking the button out and sticking with the menu item.

7.3. Memory Access Error

During the course of this research, a problem developed in Blaise that forced us to change our approach to using external lookup tables. Normally we define the EXTERNALS to be Blaise databases and only use the filename when it is needed. However, in Blaise 4.7.1 b989 and Blaise 4.7.1 b1000 certain Maniplus scripts would not run in the DEP session. It occurred with Maniplus scripts that used the INTERCHANGE setting to access the data in the DEP session. If we tried to run one of the Maniplus scripts while in the DEP session, the following error would occur:



If we were running Maniplus in quiet mode, this error caused our instrument to lock up because there was no known way to stop Maniplus from continuing to run. To get around this problem, we changed the setting for the external tables from BLAISE to MEMORY. It fixed the problem with the access violations. However, this solution presents us with another problem. That is if we used multiple lookup tables inside the rules, we will have to load each one into memory. This means the instrument needs to take up more memory, and it could affect performance.

Ultimately we removed the offending lookup table because it was a static table of abbreviations for street types (DR, ST, RD, etc.) that an experienced FR would be familiar with. If the FR did not know a certain street type abbreviation, we created an item-level help screen for them that they could reference. We also added edits to this field to verify a valid street type was entered.

8. Conclusion

Blaise 4.7 has been extremely helpful in allowing us to address many of the different and complex requirements requested by our sponsors. Without this version of Blaise it would be extremely difficult, if not impossible, to address some of the requirements. Since our various prototypes work with Blaise 4.7, we feel pretty confident that it will allow us to give our sponsors most of the listing requirements that they have requested.

9. Acknowledgements

We would like to acknowledge the following people for input into this paper and for input into how to address some of the requirements requested:

Statistics Netherlands
Karen Bagwell, U.S. Census Bureau
Tom Spaulding, U.S. Census Bureau

Appendix: Code Examples

Example 1 – Creating the Lookup Table

```
DATAMODEL ZipAndPO
  SECONDARY
    ZipFirst = ZipCode, PO
    POFirst = PO, ZipCode
  FIELDS
    ZipCode "Enter the Zip Code" / "ZipCode" : STRING[5], EMPTY
    PO "Enter the PO Name" / "PO" : STRING[28], EMPTY
  RULES
    ZipCode
    PO
ENDMODEL
```

Example 2 – Maniplus Script with Parameters Controlling the Flow of Data

```
PROCESS LookupTown
  USES
    inst              {Datamodel of the DEP}
    ZipAndPO          {Datamodel of the Zip Code and PO lookup table}

UPDATEFILE        {Used to read in and write to the lookup table}
    UpdateFile1 : ZipAndPO('ZipAndPO.bdb', BLAISE)
      SETTINGS
        OPEN = YES
        CONNECT = NO
        MAKENEWFILE = NO  {Append the current file}
TEMPORARYFILE
    DEP : inst                              {File used to link to the DEP}
      SETTINGS
        INTERCHANGE = TRANSIT      {Used to link Manipula to the current DEP session}
AUXFIELDS
    LineNum_in : INTEGER    {The line number the instrument is on when this script is called}
    NumString : STRING[25]  {Used to collect the current line number from ACTIVEFIELD}
    Button_Select : INTEGER  {Button for the dialogbox}
    Button_Ok : INTEGER
    Param_In : STRING[1]     {Holds the parameter used when this script is called}
    ZipCode : STRING[5]
    PO : STRING[28]
DIALOGBOX TownLookup  {A dialog box with the Zip and PO lookup table}
    ESCAPE = NO
    LOOKUP UpdateFile1
      SEARCH = YES
    BUTTON Button_Select
      CAPTION = '&Select'
      VALUE = 1
    BUTTON Button_Select
      CAPTION = '&Cancel'
      VALUE = 0
DIALOGBOX ZipAdded
    SIZE = (400,75)
    ESCAPE = NO
    TEXT = ('Zip Code ^ZipCode and PO ^PO have been added to the lookup table.',30,10)
    BUTTON Button_Ok
      CAPTION = '&Ok'

MANIPULATE
NumString                                                                    :=
SUBSTRING(ACTIVEFIELD,POSITION('[',ACTIVEFIELD)+1,(POSITION(']',ACTIVEFIELD)-
POSITION('[',ACTIVEFIELD)-1))
    LineNum_in := VAL(NumString)
    Param_In := parameter(1) {Get the parameter that was used when the script was called}

    IF Param_In = 'L' THEN {Use the lookup table}
```

```
            TownLookup('Select Town/Zip Code')
            IF Button_Select = 1 THEN {Zip and PO Selected}
               UpdateFile1.READ   {Take the record from the table and fill the appropriate DEP fields}
               Listing.BPALListing[LineNum_in].ZipCode := UpdateFile1.ZipCode
               Listing.BPALListing[LineNum_in].PO := UpdateFile1.PO
            ENDIF {Button_Select = 1}
      ENDIF {Param_In = 'L'}

      IF Param_In = 'A' THEN {Add the Zip and PO from the current line to the lookup table}
         ZipCode := Listing.BPALListing[LineNum_in].ZipCode
         PO := Listing.BPALListing[LineNum_in].PO
         UpdateFile1.ZipCode := ZipCode
         UpdateFile1.PO := PO
         UpdateFile1.WRITE
         ZipAdded('Added!')
      ENDIF {Param_In = 'A'}
```

Example 3 – Code for Controlling Different Listing Tables

```
BLOCK blkListing
   IMPORT in_LineNum : INTEGER
   IMPORT in_CollectInfo : TaddInfo
   IMPORT in_ListingFlag : TYESNO

 FIELDS
     LineNum (LineNum) / "Line #" : 1..3000  {Var to hold the location of the permit in the table}
     PermitNum (PermitNum) "Enter Permit Number, if there are no additional houses to list, press
     '999' to continue /    "Permit Number" : STRING[24]
      HUs (HUs) "Enter the number of housing units on the permit." / "HUs" : 1..1500
      SampleFlag (SampleFlag) / "Sample Flag" : STRING[1]
      Address (Address) "Enter the street address" / "Address" : STRING[80]
       …

   RULES
      LineNum := in_LineNum            {Set line number based on array index}

      IF in_ListingFlag = Yes THEN  {FR collects info after listing & indicate listing is done..}
         PermitNum.SHOW               {…then show the permit number, date of permit, HU's, etc}
         SampleFlag.SHOW
         …
      ELSE           {Required information if listing is not done, or FR opts to collect everything now}
         PermitNum
         HUs
         IF HUs = 1 OR HUs >= 5 THEN   {Condition for setting potential sample}
            SampleFlag := 'S'
         ELSE
           SampleFlag := EMPTY
         …
         ENDIF
      ENDIF {in_ListingFlag = Yes}

      IF PermitNum = RESPONSE THEN  {Bring the rest of the listing line on path when   answered}
         IF in_CollectInfo = AfterList AND (in_ListingFlag = No or in_ListingFlag = EMPTY) THEN
            Address.SHOW                              {User opts to collect info after listing}
            …
         ELSEIF (in_CollectInfo = WhileList OR in_CollectInfo = AfterList) AND SampleFlag = 'S'
         THEN
            Address {Additional information to be collected because listing is done or user opts to
            collect all info while listing}
                  …
         ENDIF
      ENDIF
ENDBLOCK




TABLE tblListing
```

```
TYPE
      tAddInfo (WhileList “While Listing”, AfterList “After listing is complete”)
LOCALS
      I : INTEGER[4]   {Counter to set position of the table.
FIELDS
      Listing : ARRAY[1..3000] of blkListing
      CollectInfo (CollectInfo) “Would you like to collect additional info during listing or after listing
          is complete?” /  “CollectInfo” : tAddInfo
      ListingFlag :  tYesNo
RULES
      ListingFlag.keep
      CollectInfo                          {Ask user how they want to collect permit info}
      FOR I := 1 TO 3000 DO
         IF I = 1 OR (BListing2[I-1].PermitNum <> EMPTY)THEN
            BListing2[I](I, CollectInfo, ListingFlag)       {Go into the block and add information}
         ENDIF
      ENDDO
ENDTABLE
```

Example 4 – Switching from “Line View” to “Full View”

PROCESS SwitchParallel (**Manipula Code)**

```
AUXFIELDS
 ParallelName: STRING
 i: INTEGER
PROCEDURE prc_Parallel     {find out to which parallel the active field belongs...}
 i:= position('.',ParallelName,position('.',ParallelName)+1)
 IF I>0 THEN
   ParallelName:= substring(ParallelName,1,i-1)
 ENDIF
 IF uppercase(ActiveParallel)=uppercase(ParallelName) then
   setactiveparallel('ParallelJumpDemo') {back to main}
 ELSE
   Setactiveparallel(ParallelName)
   setactivefield('')              {clear active field = go to first field of active parallel }
ENDIF
ENDPROCEDURE

MANIPULATE
 ParallelName:= activefield
 prc_Parallel
```

Blaise Code Example

```
DATAMODEL ParallelJumpDemo
PARALLEL
   ATable.Rows
BLOCK TABlock
FIELDS
   field1, field2, field3: STRING[2], EMPTY
ENDBLOCK

TABLE TATable
 FIELDS
   rows:ARRAY[1..10] OF TABlock
ENDTABLE
FIELDS
 Q : String[4], EMPTY
 ATable: TATable
ENDMODEL
```

The New, Adapted German Microcensus and the Integrated Labour Force Survey – User-specified Solutions with Blaise

Stephan Matthias (German Federal Statistical Office, Germany)

1. Introduction

The Microcensus, with approx. 820,000 respondents, is by far the largest official household questionnaire in Germany. It also integrates the EU's Labour Force Survey, which must be carried out in all Member States.

Up until and including 2004, each set of Microcensus data was collected in a fixed report week. As a result of the ever-accelerating processes of change among the population, however, the data obtained had already lost much of their topicality by the time they were published.

As a reaction to this, Council Regulation (EC) on the organisation of a labour force sample survey in the Community stipulated back in 1998 that collection should be spread evenly over all calendar weeks of a year. What is more, average quarterly and annual results were to be reported to Eurostat. This requirement was transposed into national law by the Microcensus Act (Mikrozensusgesetz) as on 1 January 2005, and hence was used for the first time in the 2005 Microcensus survey.

The major goals of this continuous collection were the improved topicality and quality of the data collected, as well as the concrete implementation of the requirements of the EU Labour Force Survey.

2. Conception

Because of the federalist structure of the Federal Republic of Germany, the Microcensus is only carried out locally by the individual Federal Länder, the Federal Statistical Office being obliged to provide the collection tools to the Länder.

In order to be able to change from a collection which is partially carried out in paper form to a procedure which is completely computer guided, all the Land Statistical Offices had to be converted to a uniform operating system (Windows XP). Each Land obtained for itself the requisite number of laptops and tablet PCs.

To implement the collection during the year, it was necessary to develop programs which were suitable for use on laptops, i.e. to collect the data, as well as for processing of the data at the Land Office.

We were able to use only the interview program which had been previously deployed as a Blaise program, and which had to be adjusted to the conditions encountered in continuous surveys. The organisation papers, in other words the total logistical information as to who must be asked which questions and at what time, was previously sent to the interviewer by post and returned by post after completion. Now it had to be completely re-developed, and particular account had to be taken of data protection. In addition, a routine had to be developed which could assign the corresponding organisation paper to the interview without the personal data, such as name or address, being stored in the interview. Since some households prefer to complete a written questionnaire (a "self-completion sheet"/

“Selbstausfüllerbogen”) instead of the personal interview, a possibility also had to be created here to incorporate these self-completion sheets into the new structure.

Since neither the Microcensus nor the European Union's Labour Force Survey is a voluntary questionnaire in Germany, but both are obligatory, major questions emerged as to the process-orientated structure in conversion to a continuous survey. A list of all of them does not fall within the framework of this talk.

It should be pointed out that the interviewers deployed are not staff of the Statistical Offices, but work on a freelance basis, and in many cases have inadequate PC knowledge. The program had to be made as user-friendly as possible for these interviewers. It was therefore necessary amongst other things to organise training for the interviewers, who were to learn about both the specialist Microcensus-specific aspects and be taught how to use the laptop and the program. The organisation papers previously available in paper form were to be implemented one-to-one in the Blaise model, so that the interviewers did not have to make further adjustments in the introduction to continuous surveys, and the recognition value for them was high.
Further, it was important to clarify how and by what means the data transmission between the Land Statistical Offices and the interviewers was to take place.
The greatest handicap in building up a data administration with BLAISE unfortunately proved to be the lack of a Blaise Component Pack.

3. Implementation

In view of the different tasks which the interviewers and the Land Offices have to perform, we developed a program which also displays different menus, depending on the start function used for launching the program.

Figure 3.1 Menu navigation for the laptop user (interviewer)

The organisation papers (such as the *building list* - *Gebäudeliste*, the *household characteristics* - *Haushaltsmerkmale*, the *distribution list* - *Verteilungsliste*, the *notes* - *Notizen*, the *final report* - *Notizen* and the *permanent sample* - *Dauerstichprobe*), are shown in the laptop program under the menu item "**ORGANISATION**". These organisation papers contain the protected personal data, which must be stored separately from the interview. They are also input to the survey's logistical process.

An interview can only be implemented if the address is available in the organisation papers. The laptop user can only access the interview via the menu item "Organisation".

Under the menu item "**TOOLS**", one can find the tools "*Save*", the display of the "*Delete list*" and the "*View of all interviews*". The latter tool shows the remaining "*Incorrect households/accommodation units*".

The menu item "**DATA TRANSFER**", which has been completely newly developed for the electronic data dispatch, contains the tools "*Send data*", "*Receive work packages/data*", "*Input work packages/data*", "*Send errors*", "*Update errors*", "*Info text*" and finally the tool "*Start again*".

All the information that is important for the interviewer is kept in pdf. format in the penultimate menu item "**DOCUMENTATION**". This is to make the special knowledge of the Microcensus, as well as the labour force statistics, understandable for the interviewer. The most important point in this context is the instructions for the laptop, which describe the use of the laptop program and are thought of as a help function.

Using the tool "*Information from Land Office*", the Land Office can specifically transmit its own reports to the interviewer (e-mails are not currently permitted for reasons of data protection).

The last menu item "*Info*" shows the interviewer the respectively valid version number of the program on the laptop.

Figure 3.2 View of version number

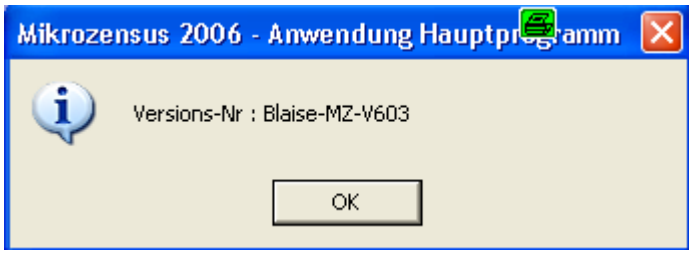
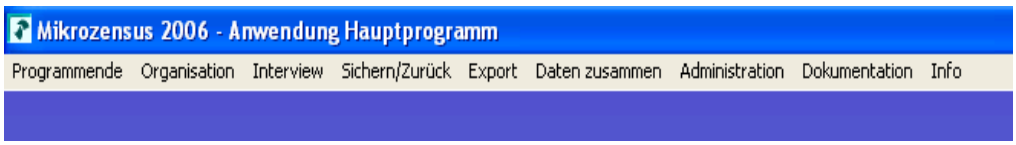


Figure 3.3 Menu navigation for the Land Office application



The menu item "**ORGANISATION**" corresponds to that of the laptop application.

The menu item "**INTERVIEW**" contains the tools "*Process interviews*", "*Count incorrect surveys*", "*Count households/districts*", "*Count clean households*", "*Print notes*", and the tool "*Delete list*".

Under the menu item "**SAVE/BACK**", it is possible to backup data to a drive that can be selected at will, and to transfer data back.

Under the completely newly-designed menu item "**EXPORT**" one finds the tool "*Export to ASCII*", which facilitates the export of the interviews and of the organisation data from BLAISE format to ASCII. The tool "*Export to Excel*" makes it possible to export both the Microcensus data and the organisation papers for a quick check to Excel using freely-selectable parameters. The "*Final report*", which shows the conclusion of a district by the interviewer, is also shown for further processing to Excel.

Several fields are also integrated here in which the interviewers can make direct entries on invoicing.

The "*Export permanent sample*" tool exports addresses of households which have volunteered for further statistics.

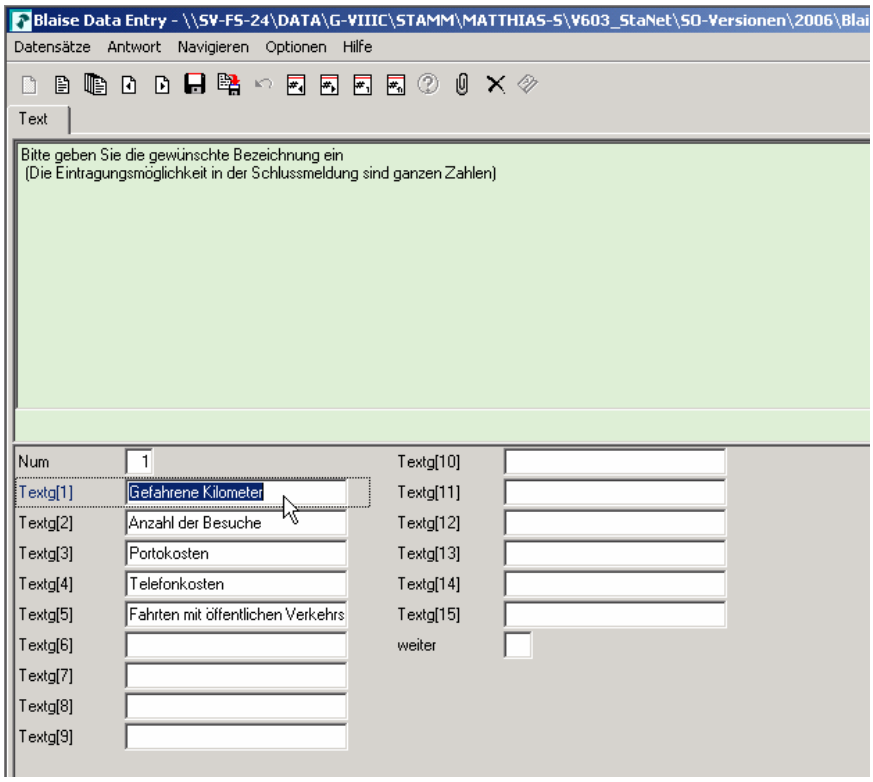
The last export tool is the "*Export plausibility check*". The survey program is equipped with up to 260 plausibility checks. A record is kept in the background of which plausibility check was set off how often. This file is exported here and sent

to the BLAISE TEAM of the Federal Statistical Office, and is used as the basis for improving the question structure, and hence also for improving quality in a new question program.

The menu item "**COMBINE DATA**" comprises the possibility to combine all data transmitted by the interviewers into the existing stock.

The menu item "**ADMINISTRATION**", with the tool "*Staff in the Land Statistical Office*", offers the possibility to define the distribution of rights and ID of the interviewers and staff of the Land Statistical Office.
The "*Final report texts*" tool affords the Federal Länder the possibility to freely design the texts appearing in the final report, and hence also of the invoicing procedure.

Figure 3.4. View of the free formulation of the question texts



The tool "*Combine all data*" is then needed in case of interviewer absenteeism for instance as a result of illness, or if data are lost in transfer. We built in this tool as an additional security. This also allows us to incorporate those interviews in the overall stock which have not yet been released for further processing by the interviewer. This menu item is exclusively provided for the exceptional case. In fact, the last existing data item of the interviewer in question must once more be copied into the folder "Data dispatch/receipt". With the menu item "Combine all data", these data are then input into the overall stock. Data items subsequently received on these districts are rejected since the final report is already available in the Land Office.

The hospital run of BLAISE is launched with the "*Hospital*" tool.

The "*New model*" tool is also provided only for the exceptional case, and is only needed if a new model must be delivered. The available data are saved and after installation are adjusted to the new model and played back once again (Blaise to Blaise).

There are steps in the Land Office in which it is necessary to be alone in the Blaise system. This can be ascertained using the "*Monitor*" tool without having to leave one's workplace.

It is possible here to select between the various organisation programs (Gebäude, Haushaltsmerkmale, Verteilungsliste), the interview (Interview), the final report (Schlussmeldung) or the permanent sample (Dauerstichprobe).

Figure 3.5 Dialogue view

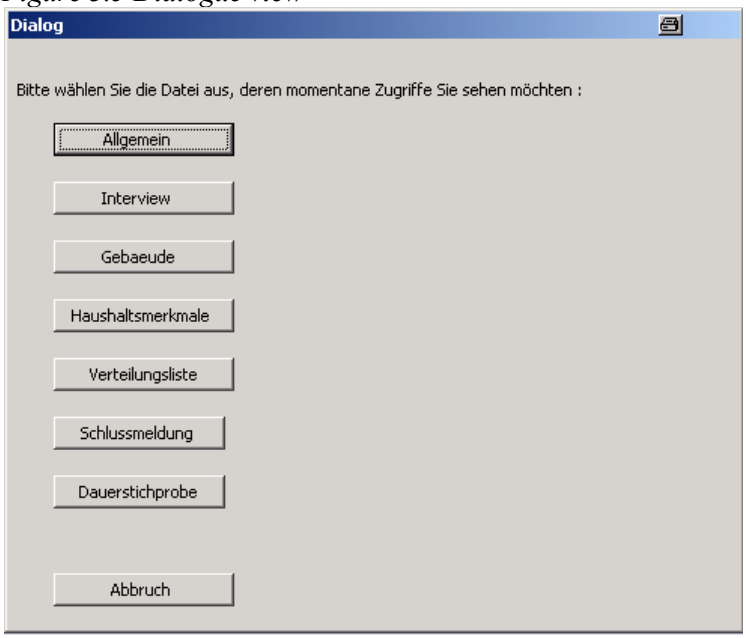
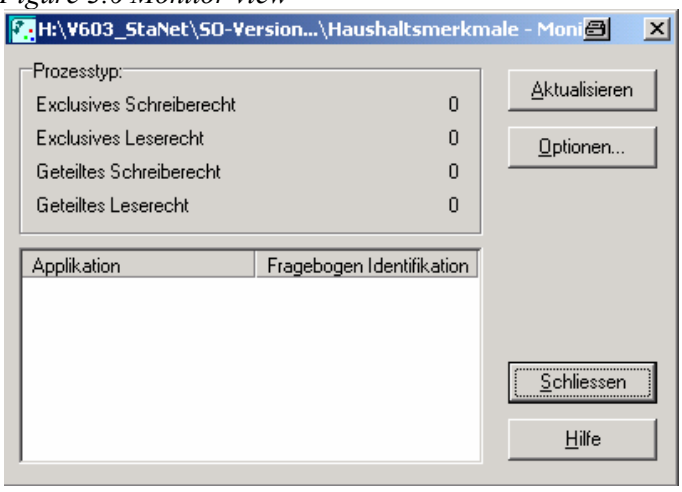


Figure 3.6 Monitor view



The final "*Annual transition data export*" tool serves to take on into the next year and process interviews of one year which have not yet been dealt with.

The menu items "**DOCUMENTATION**" and "**INFO**" have the same functionalities as in the laptop application.

4. Brief description of the functions

4.1. Work packages

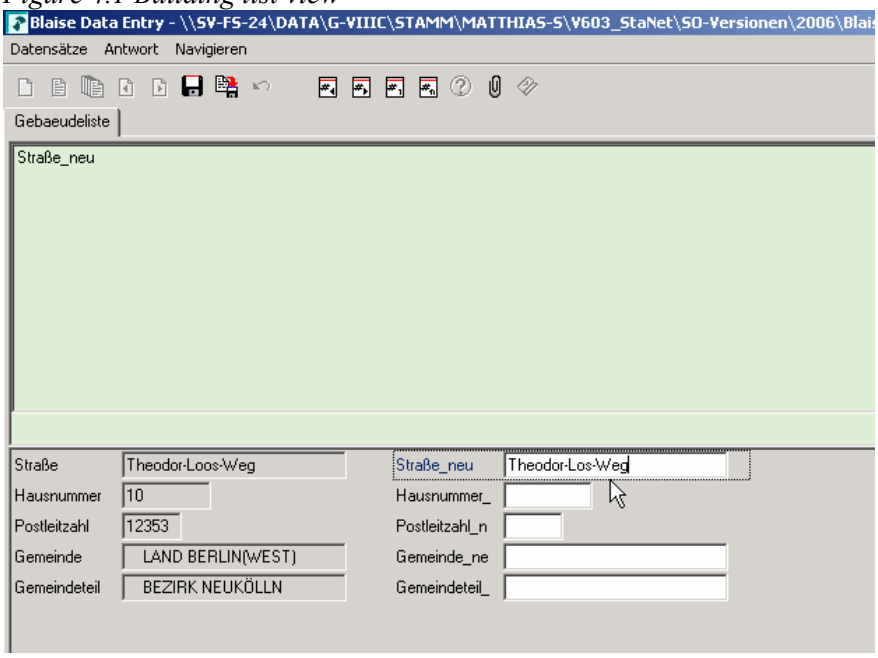
The Land Statistical Office in question sends interviewers one or several work packages to their laptops. When logging onto the laptop, a routine examines whether a work package is available for the interviewer. The interviewer calls up the work package via the menu item "Receive data"; if the transmission was successful, he/she calls up the menu item "Input work package". The work packages provide the interviewers with all the important information, such as the address of the households to be interviewed.

Since a household is asked once per year in four consecutive years, a variety of approaches had to be planned.

4.2. Building list

The interviewer therefore calls up the building list on the first survey and examines whether all the information in the address is correct, or whether there are deviations. In this case, he/she can make changes which are sent along the next time the data are transmitted to the Land Statistical Office. The Land Statistical Office then determines whether the changes are to be accepted for future use.

Figure 4.1 Building list view

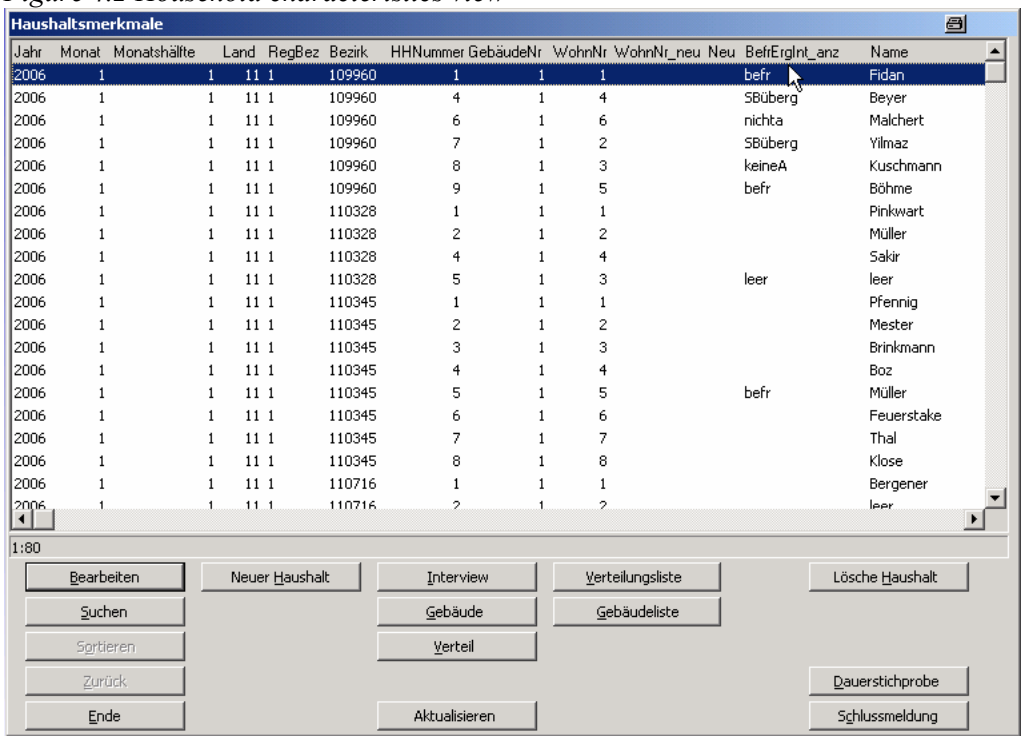


It is now possible to change from the building list directly to the household characteristics.

4.3. Household characteristics

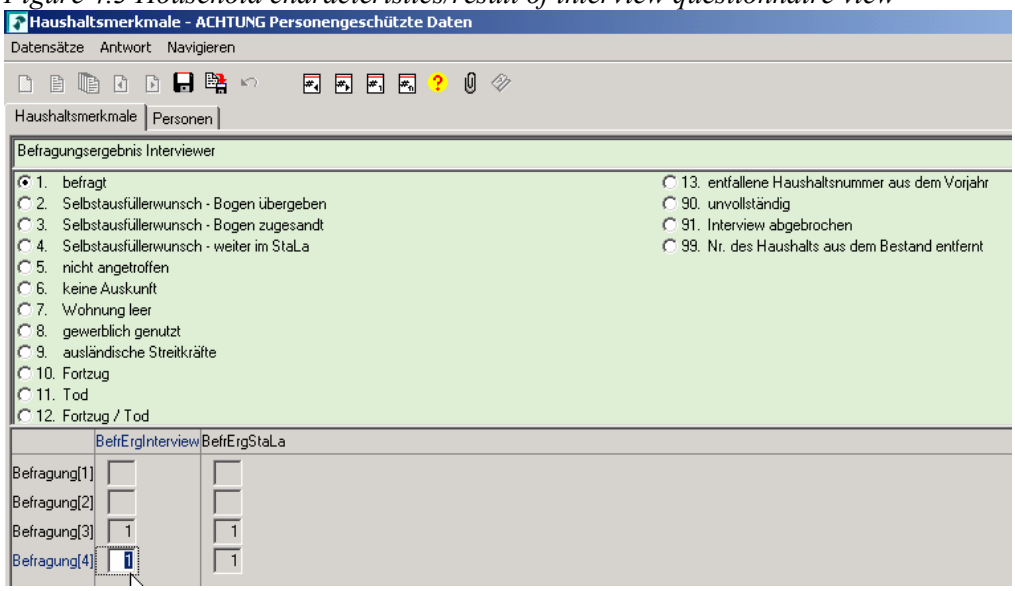
In the overview of the household characteristics the interviewer can see the entire district, in this instance district "109960", at a glance. Under the item "Questionnaire result of the interviewer (*BefrErgInt_anz)*", he/she can see the status of the individual households, here "responded ***(befr)***".

Figure 4.2 Household characteristics view



The existing, unchangeable household data can be added to at the item "Edit". With an initial survey, amongst other things the names of the persons in the household and the telephone number are determined here. Finally, the interviewer enters the result of the survey. In the following year, the data of the previous year are automatically shown, and need only be updated.

Figure 4.3 Household characteristics/result of interview questionnaire view

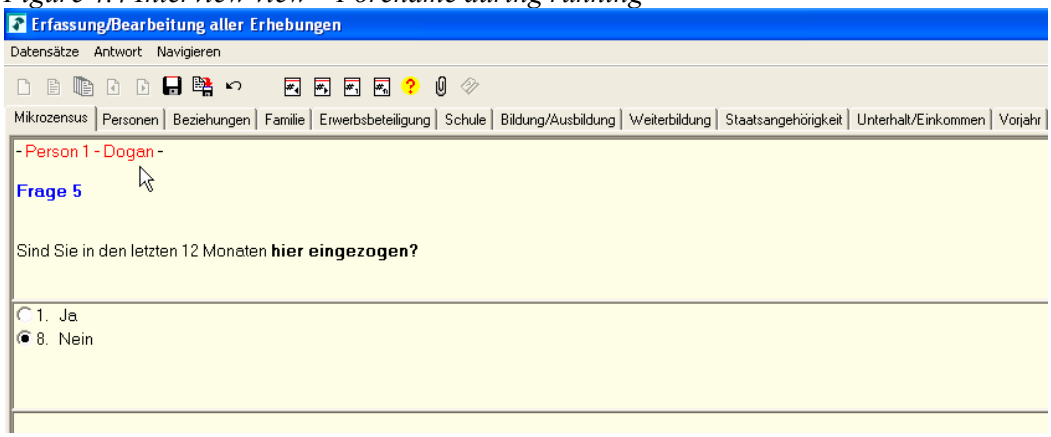


The interviewer can now call up the actual interview directly from the household characteristics.

4.4. Interview

All unprotected data are automatically inserted into the interview here. As an aid for the interviewer, the respective "Forename" is shown during running. The name is however not stored along with the data.

Figure 4.4 Interview view – Forename during running

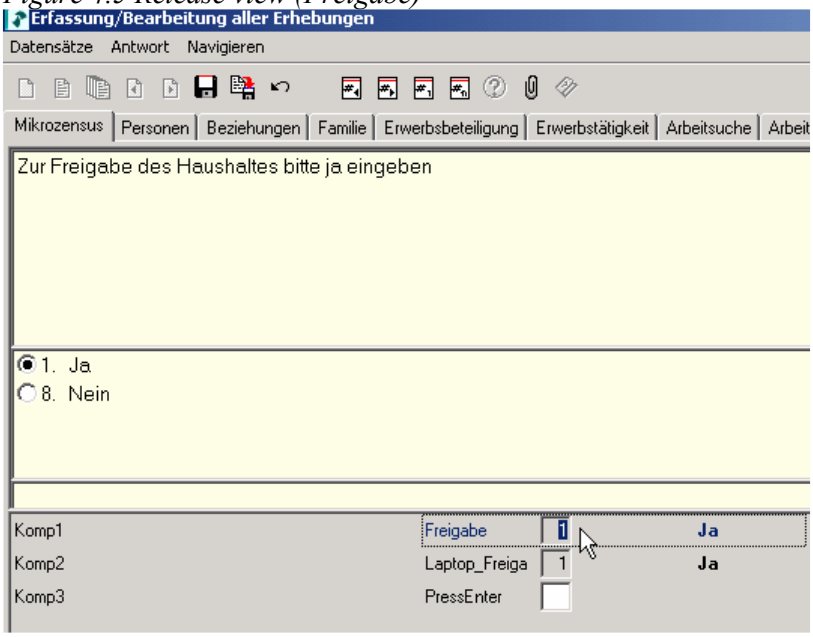


4.5. Release

Datasets which have been completely processed by the interviewer must be marked for transmission to the Land Statistical Office. The possibility is offered to release the dataset as soon as the household data have been entered.

"Released" means that in the view of the interviewer the dataset has been completely processed and can be sent to the Land Office. These can be complete interviews, or indeed interviews which have been interrupted.

Figure 4.5 Release view (Freigabe)



For safety, so that it is not possible to accidentally release a household that is not yet ready, the question is asked twice. It absolutely must be confirmed twice by entering a "1", as otherwise the release is not effective!

When the interviews are entered in the Land Office, the "Release" field is automatically pre-set at "8" (No) since no further release or transmission is necessary.
After this, the interviewer can no longer change anything about the interview. He/she can call up the dataset again, but it is then read-only and no more changes can be made.

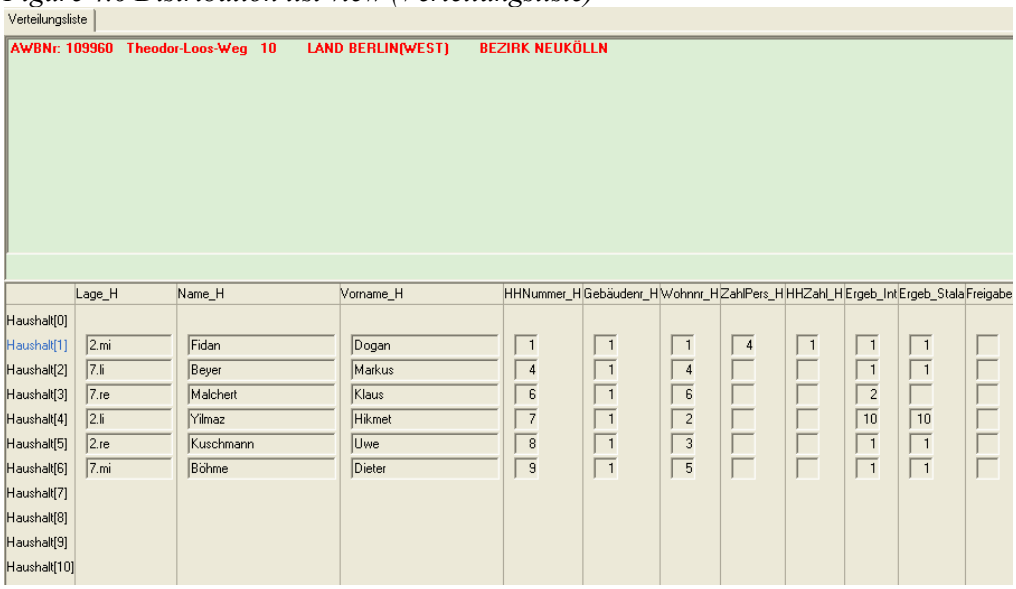
4.6. Send data

All interviews that have been released are sent to the Land Statistical Office using the "*Send data*" tool. The complete stock of the organisation papers available is always sent. An automatic test is now carried out in the Land Office to determine whether any datasets with the same ID already exist. If there is no dataset with the same ID, it is incorporated into the overall stock.

4.7. Distribution list

The "Distribution list" provides the interviewer with an additional overview as an orientation tool which always shows him/her a complete district.

Figure 4.6 Distribution list view (Verteilungsliste)



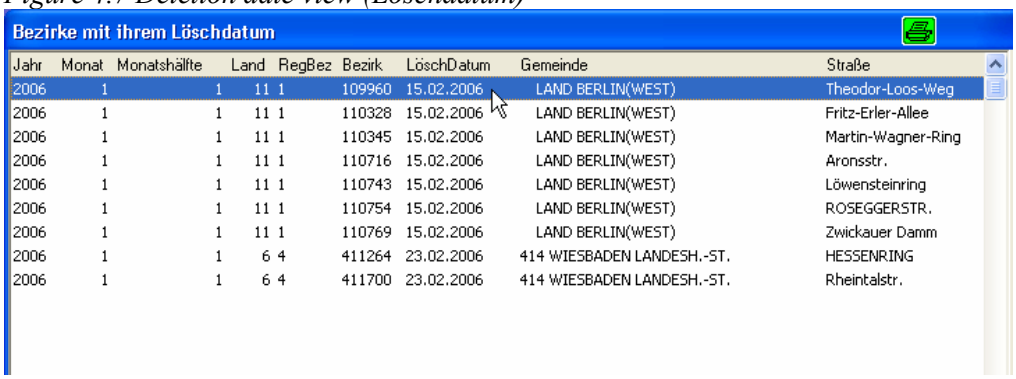
In addition to the selection district, the address, the name (*Name_H*), the household number (*HHNummer_H*), the building number (*Gebäudenr_H*), the apartment number (*Wohnnr_H*), the number of persons in the household (*ZahlPers_H*) and the interviewer's questionnaire result (*Ergeb_Int*), it also is shown whether the household has been released (*Freigabe*) for dispatch to the Land Statistical Office.

4.8. Delete list

The interviewer has six weeks to work through a work package. After five weeks, he/she receives a reminder when starting up the laptop that the datasets will be deleted from the laptop in one week. Having said that, only selection districts are checked here of which no final report is yet available.

The "Delete list" tool shows the interviewer all districts available on the laptop, as well as their deletion dates. The same tool can be used in the Land Statistical Office to check when which districts are to be deleted from the interviewers' laptops.

Figure 4.7 Deletion date view (Löschdatum)



Jahr	Monat	Monatshälfte	Land	RegBez	Bezirk	LöschDatum	Gemeinde	Straße
2006	1	1	11	1	109960	15.02.2006	LAND BERLIN(WEST)	Theodor-Loos-Weg
2006	1	1	11	1	110328	15.02.2006	LAND BERLIN(WEST)	Fritz-Erler-Allee
2006	1	1	11	1	110345	15.02.2006	LAND BERLIN(WEST)	Martin-Wagner-Ring
2006	1	1	11	1	110716	15.02.2006	LAND BERLIN(WEST)	Aronsstr.
2006	1	1	11	1	110743	15.02.2006	LAND BERLIN(WEST)	Löwensteinring
2006	1	1	11	1	110754	15.02.2006	LAND BERLIN(WEST)	ROSEGGERSTR.
2006	1	1	11	1	110769	15.02.2006	LAND BERLIN(WEST)	Zwickauer Damm
2006	1	1	6	4	411264	23.02.2006	414 WIESBADEN LANDESH.-ST.	HESSENRING
2006	1	1	6	4	411700	23.02.2006	414 WIESBADEN LANDESH.-ST.	Rheintalstr.

The Land Statistical Office can naturally change the processing date at any time, and hence also the deletion date. Short processing times, on the one hand, and as even a spread of the surveys over the whole year as possible, on the other, can lead to a situation in which the interviewer cannot reach a household in the set time. The Land Statistical Office can however continue the questionnaire after it has been released by the interviewer. This also includes the self-completion sheets in paper form, which can be input in the BLAISE program in the Land Statistical Office.

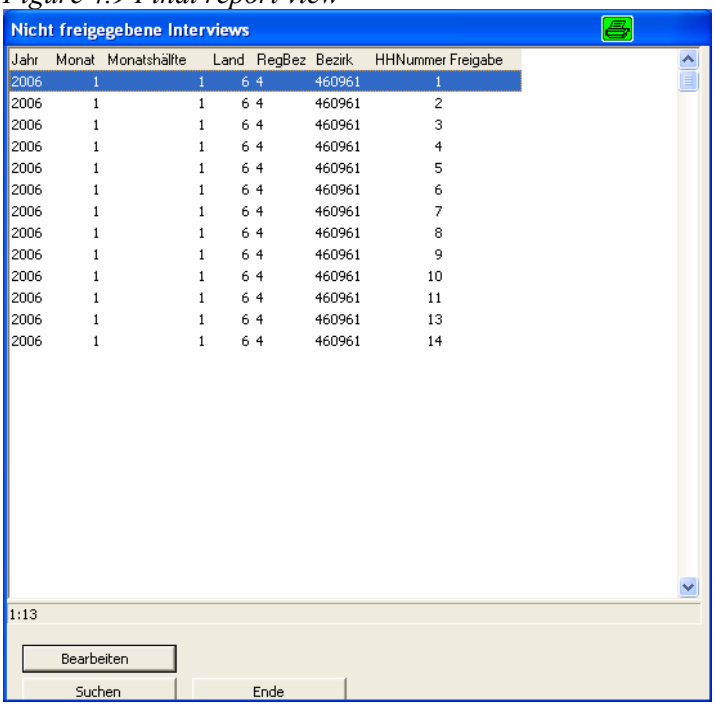
4.9. Final report

So that the Land Statistical Offices know when a selection district has been completely processed by the interviewer, the interviewer must call up and complete a so-called "*final report*" for its district. Only the interviewers ultimately know when all their households in a district have been completely processed. At this point, a check is also made to see whether there are datasets in the material which have not been released. Should this be the case, an information message appears to say that the district cannot be released for this reason.

Figure 4.8 Final report view

It is then possible to look at and appropriately process the as yet unreleased interviews in a dialogue view.

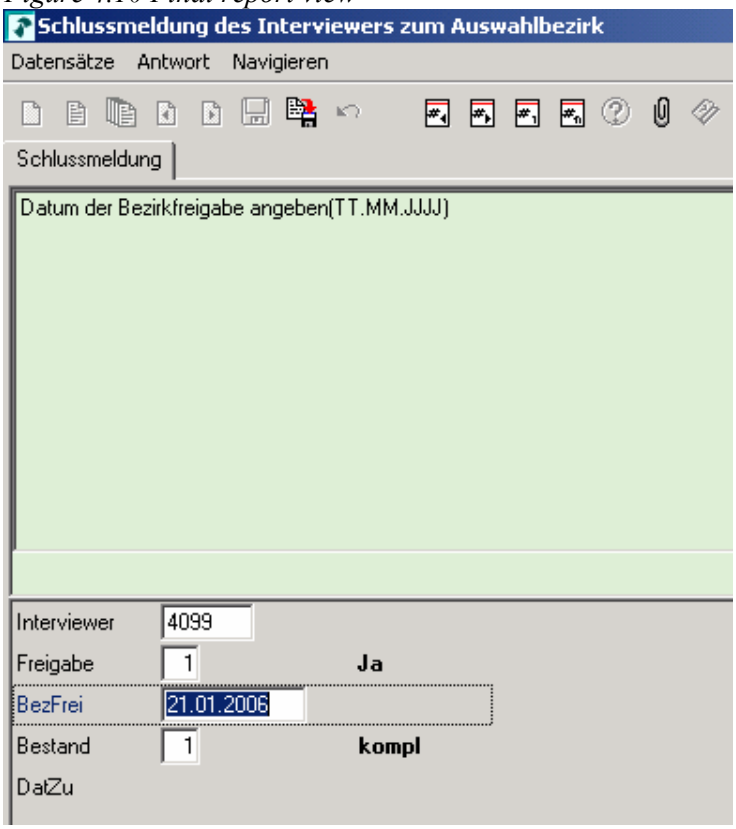
Figure 4.9 Final report view



Once released, the final report can only be opened by the interviewer in read-only form.

This final report is sent to the Land Statistical Office with the other organisation papers.

Figure 4.10 Final report view



4.10. Multi-user concept

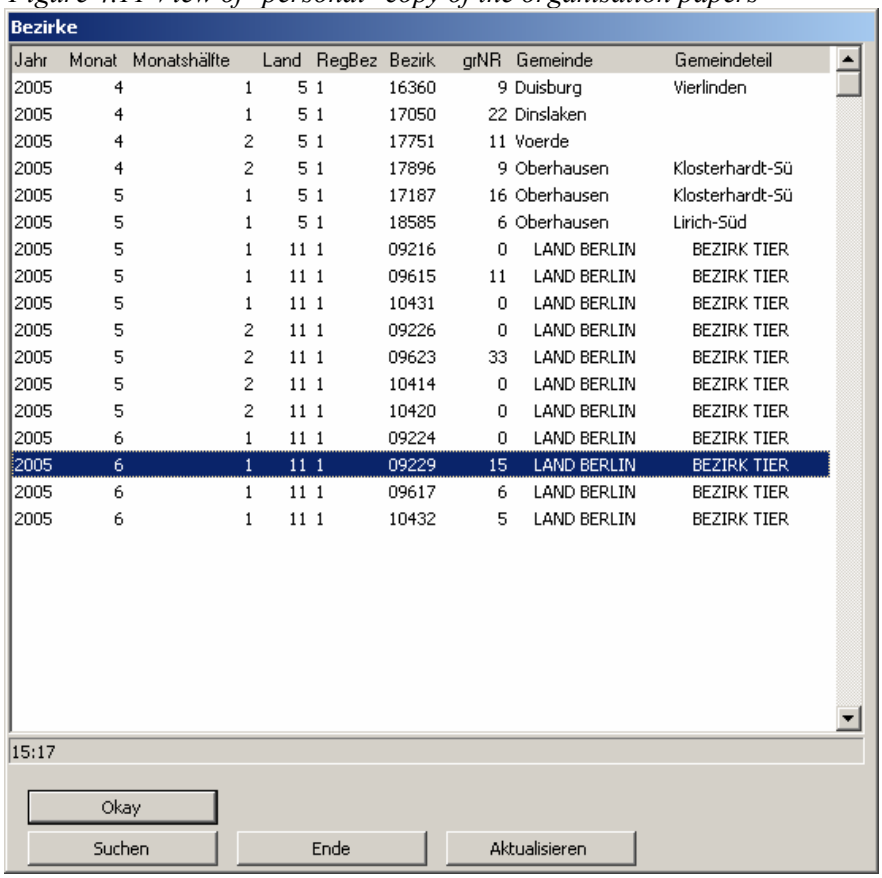
We have also developed our own multi-user concept.
So that for instance there are no problems when entering or processing datasets in the interview or in the organisation papers, we have barred datasets which are being processed from access by other staff members. To do so, it was necessary to copy files into a temporary, user-specific file.

I would like to show you this using an example:

As soon as a member of staff calls up the building characteristics, a "personal" copy of the organisation papers is created for each staff member for them to work in.

"Personal copy" of the organisation papers means that the views, for instance of the household characteristics, are based on the copy created at this time.

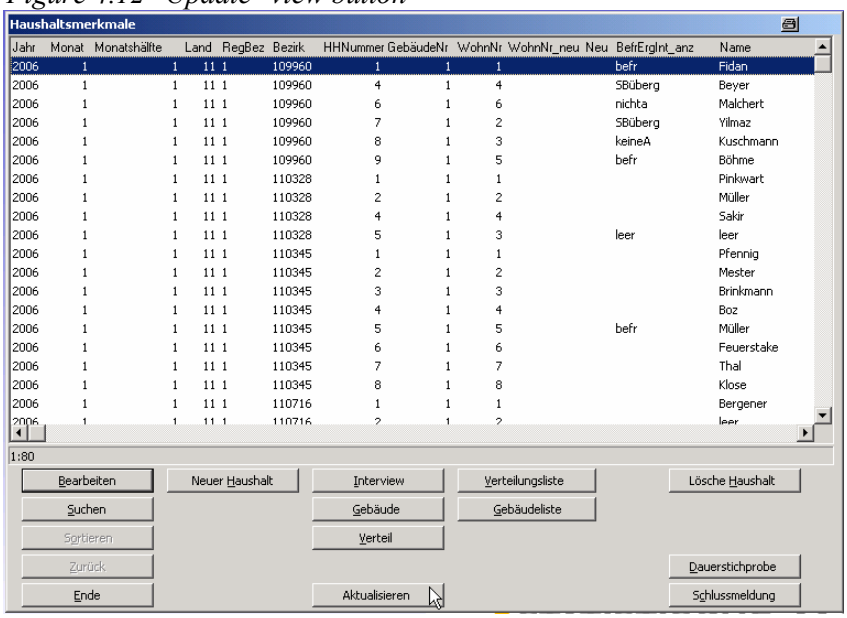
Figure 4.11 View of "personal" copy of the organisation papers



The staff however always make changes in the datasets in the original data material!

These changes are however only visible in the view masks when the "*Update*" button is pressed. This button repeats the copying of the data stocks, and the newly-changed data are shown.

Figure 4.12 "Update" view button



The processing of the households within a district, and the change to another district or to the appropriate distribution or building characteristics, takes place much faster if one works in a copy.
An automatic update of the data shown takes place on leaving the organisation papers. The next time it is opened, an up-to-date copy is created once more.

4.11. Update

If an update was developed by the Federal Statistical Office and successfully tested, this is provided to the Land Statistical Offices via a routine, and the Offices in turn pass it on to the laptop interviewers. When the interviewers send the data, which they are to do every working day, they receive a message as to whether an update is available. They must download the update and install it. This procedure only involves two mouse clicks for the interviewer. The old program is automatically copied to a temporary folder, the update is also automatically installed and tested. The interviewer can continue to work with the new version. If problems occur with the update, the interviewer can continue to work using the stored back-up version.

4.12. Combine data

I would now like to briefly explain to you how the files sent by the interviewers are input in the total stock of the Land Office by running the menu item "**COMBINE DATA**":

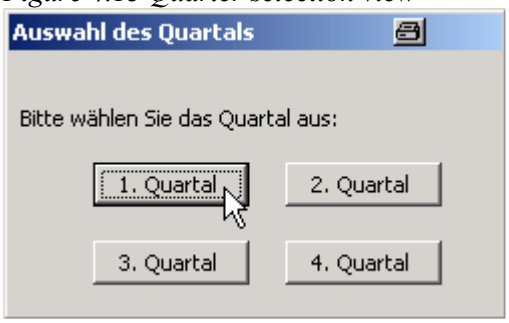
- All organisational data, as well as the previously-released final reports and interviews, are incorporated into the overall stock.
- If released interviews are found once data are input once more, the household characteristics, as well as the interview, are not overwritten.
- If the building data have already been inspected by staff, and the changes labelled with "accept" or "don't accept" in the "Orders" indicator, this building set is no longer overwritten the next time the combination function is carried out.
- As soon as a released final report is available in the dispatch package, it is accepted into the total stock.
- The next time interviewer data are combined, no input is carried out for the selection district for which a released final report is available in the Land Office!

N.B.: No final report can be established in the Land Office itself.

4.13. Export

Since the continuous survey necessitates that we no longer publish our data only annually, but also quarterly, we naturally also had to convert the data export functionality. When exporting the data, it is now possible to select the desired quarter which one would like to export. The corresponding datasets and the appropriate organisation data are then exported.

Figure 4.13 Quarter selection view



The collected organisational data from the first survey year are used for the follow-up questionnaires, and then only updated on an annual basis.

The data material created is used directly for expansion with Bascula. My colleague Kirsten Iversen will be explaining this to you in greater detail later.

4.14 Result

Looking back, it is possible to say that the introduction of the continuous Microcensus was a success. Although we only had the "Blaise Basic System" licence, the flexibility of BLAISE was very helpful to us in finding the necessary solutions.

Not all the functions ran perfectly from the start, but in the course of the year, we were able to incorporate many functions which served specialist purposes, or which improved the program, making a total of eight updates, without ever losing a dataset in Germany. We did underestimate the training needed on the new BLAISE application for the freelance interviewers and the staff of the Land Statistical Offices. Much more training will be necessary for these users in the following years.

Also with the 2006 Blaise application, we were able to implement new ideas once more which emerged from the first year of continuous application.

I would like to thank you for your generous attention, and also to wish all BLAISE developers continuing success.

Training and Learning Opportunities using Blaise

Colin Setchfield (Office for National Statistics, UK)

1. Introduction

Though the option of providing a WinHelp-based Question-by-Question Help (Q-by-Q) is not new to Blaise, ONS has only recently introduced such a system, as one of the developments for its proposed Integrated Household Survey (IHS), formerly known as the Continuous Population Survey. The IHS is currently in its pilot stage, and due to be launched in 2008. This paper gives details both of this development, and of ONS's use of a Blaise questionnaire to test interviewers' learning as part of their pre-fieldwork training. Other international organisations have developed similar systems in the past, however, particularly with the Q-by-Q, ONS developed its versions independent of what had already then currently existed elsewhere.

1.1. Background to Integrated Household Survey

The IHS will bring together, into one Blaise instrument, four existing UK surveys: the Labour Force Survey (LFS), the Expenditure and Food Survey (EFS), the General Household Survey (GHS), and the National Statistics' Omnibus Survey. Fieldwork trials began in 2005 (Bumpstead, 2004).

Prior to this development work, ONS employed three mutually-exclusive field forces of interviewers: two of which each worked wholly on a single survey – one group working on the LFS, the other working on the International Passenger Survey. All other surveys (including the EFS, the GHS and the Omnibus Survey) were administered by the third group: the largest of the three – the General Field Force (GFF).

Each of these field forces had their own distinctive training and working practices; the combination of surveys forming the IHS involved two of these field forces: LFS and GFF. The IHS project therefore was not simply creating one questionnaire out of four, but rather extended to the whole of the survey and interviewing process. ONS's previous use of Blaise was reconsidered; this paper details how the use of Blaise was extended to meet the needs and challenges of creating a new model for training interviewers, assisting their learning, and providing an interactive reference resource both for briefings and within the interview situation. The paper looks both at the processes and some of the standards beginning to be developed before the systems are rolled out wider within ONS.

1.2. Existing training: pre IHS

Pre 2005, the initial training of interviewers at ONS generally followed the same principles as set out in the paper presented at the 3rd International Blaise Users Conference in 1998,[1] though with modifications in its detail[2].

LFS interviewers – only working on the one survey – received their survey briefing at the same time as undergoing their initial training following recruitment. GFF

[1] Goodger, Chris (1998): Training Interviewers in the Use of Blaise, Office of Population Censuses and Surveys, London, UK. Paper presented at 3rd International Blaise Users Conference 1995, Helsinki.

[2] Details of these were presented (by Chris Ash, Office for National Statistics, UK) in a training session at the 7th International Blaise Users Conference 2001, Washington D.C.

interviewers – working on a range of continuous, repeating and once-off ad hoc surveys – received separate survey briefings subsequent to their initial training but prior to first working on the specific survey. Most survey briefings were conducted on a face-to-face basis, though owing to its short timetable Omnibus received a postal briefing.

In addition to initial training and these face-to-face briefings, during the field period itself reference material was provided by extensive paper survey manuals supplemented by more targeted interviewer instructions within the Blaise instrument. These instructions were divided between (1) those displayed on screen below the question text (for brief information that needed to be readily available at the time of asking the question) and (2) those contained in a separate language specified in Blaise and called up by using a hotkey.

The first feasibility test of the IHS, in February 2005, adapted the existing training/learning systems to the requirements of the survey. The team developing the new survey, however, was keen to explore more interactive approaches and requested that these be developed in readiness for its second feasibility test in September 2005. The result was a Q-by-Q utilising a WinHelp facility, able to be activated both outside of and within Blaise, and a Blaise questionnaire designed to test the effectiveness of the interviewers' learning.

2. Q-by-Q: a Searchable Help facility

2.1. WinHelp and Authoring Tools

In 2000, Couper et al recommended that, ideally, on-line help should be provided for interviewers by using WinHelp. Their argument was mainly drawn from system operational considerations. The benefits of using this option, however, for many organisations, were outweighed by the complicated preparation of such files with its dependency on software not used or tested within the organisation. Some organisations, such as Statistics Canada, however, have successfully introduced a WinHelp on its surveys.

To ONS, the WinHelp option seemed best suited to meet the complex on-line help requirements of the IHS. Its main benefit was in reducing the learning burden for interviewers, particularly for those who were recently-appointed and had little or no exposure to the individual surveys that IHS comprised. By utilising it, help could be provided at the point when it was immediately required by the interviewer without having to carry around hefty paper documents for reference and without as much interruption to the interview situation. It provided not only immediate answers to issues raised while interviewing but also could be used as a stand-alone document acting as a detailed reference manual for study and at home. Additionally, there were also cost benefits in the reduction of printing and postage.

ONS, however, faced many of the same issues as experienced by other organisations, such as the Australian Bureau of Statistics (Wensing et al (2003)). ONS had explored WinHelp in the past but never used on any of its social surveys. On these previous occasions, the software RoboHelp from Macromedia had been used for the creation of HLP files. Though it was no longer actively used within the office, RoboHelp had become part of its standard software suite from these previous occasions. RoboHelp, therefore, became the preferred software to use, simply on the basis that it had been used in the past. Despite this, previous uses of the software were not recent. With no past users being available, the developers of the new on-line help consequently found themselves having to learn how to use the software unaided.

As it was part of the office's software suite, no comparative work between the functionality and usability of RoboHelp and other similar packages was undertaken. Nevertheless, without much experience in producing WinHelp files, a single source authoring tool (such as RoboHelp, Doc-To-Help or Microsoft Help Workshop: which was initially considered) would have been selected in any case, as these are better suited for importing existing interviewer instructions written in Microsoft Word and for producing printed copies should these be required.

2.2. Creating a House Style

Help document

The most straightforward approach would have been to take the existing paper instructions and simply to have processed the document as it existed, creating topics at all major chapters and sections within the document. This would have, however, simply created an on-screen version of what had previously been available in a paper format, without taking into account the different layout considerations of these two formats.

In order to provide comprehensible help screens, the following general rules were agreed.

- Where possible, help on screen should be contained within the display window, so that scroll bars are not necessary to view the full text.

- Help should be presented plainly and limited only to information that the interviewer would immediately need while conducting the interview.
- Additional detailed information (if required) should be provided in linked pages within the Help

Few of the existing instructions for individual questions within the questionnaire met the first two rules, and therefore work was required on how to divide them meaningfully for presentation in the Q-by-Q. The schema selected was that each question in the questionnaire could have up to a maximum of four associated help screens; these were named as follows (listed in order of priority/use).

- **Summary:** Advice most likely to be required by an interviewer in the live interview situation. Normally, this would relate to the help that interviewers previously saw on screen immediately below the question text. The text needs to be brief and to the point.

- **Question:** Providing details of the Field Text and types was felt useful if Help was being used outside of Blaise, such as in briefing or study. This screen is used currently also for linking to other definition and question screens, by creating jumps/hyperlinks to those screens using corresponding word(s) in the question text or answer categories. (This information might in the future be usefully extracted direct from the Blaise questionnaire, such as by using Delta.)

- **Instruction:** More detailed instruction directly related to the question to enable the interviewer to understand how to administer the question depending on the circumstances of the respondent(s). Normally, this would relate to the help previously contained in a separate language within Blaise.

- **Background:** This provides more general background to why the question is asked. It is not intended for information required to be known to answer the question itself, but rather should address the answer to an inquiry from the respondent such as "Why is this question being asked?" or "How will this

question be used by you?" or "What results have been shown by this question in the past?"

Within the Help project, each page was named first according to its Field Identifier followed by the relevant page type (e.g. the summary screen for the question on legal marital status was designated as 'MarSta summary').

It was acknowledged that some questions did not require all four options and others required no help screen at all. Additionally, this schema was not applied to pages for definitions of concepts and terms used within the questionnaire or for general help screens not directly related to individual questions. In both of these cases, each instance was given its own individual screen.

A template was created for authors of the new instructions, which set the following layout standards.

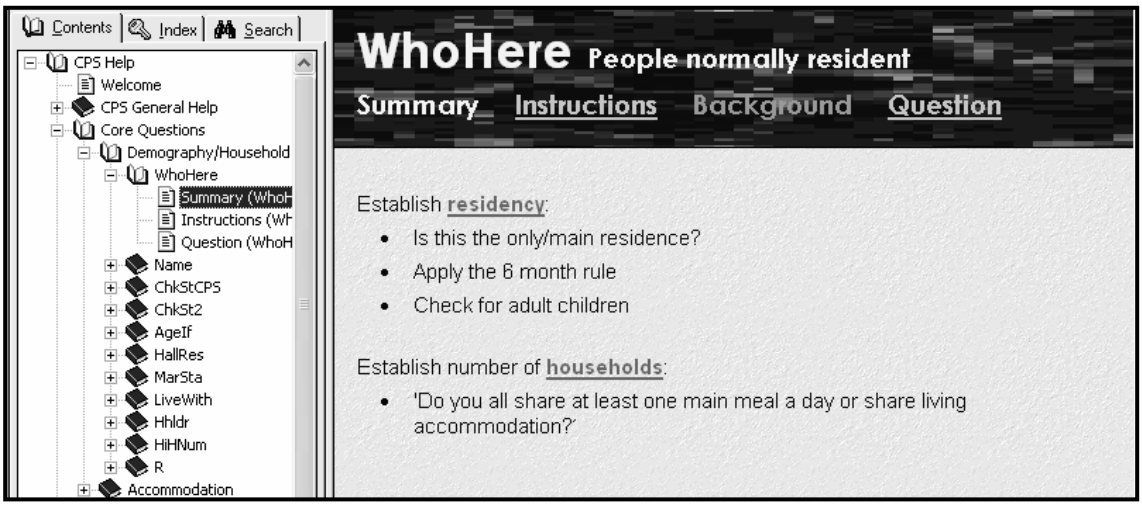
- Each screen was to be headed by a non-scrolling region comprising three lines. The first line displayed the Field Identifier and Field Description; the second line, listed the names of the four associated screens, coloured-coded according to their availability (light turquoise for the live screen, light blue – and underlined – for an available screen accessible using a jump, and 40% grey to indicate screens not available for the specified question), and the third line remaining blank for aesthetic effect.

- Fonts without serifs were preferred as being more suitable for people with visual impairment. The non-scrolling header was formatted in Century Gothic with the main body of the text in Arial. Standards were also defined for font sizes, with the main body of text set at Arial size 11 (Gatward, 2004).

- Font colours were selected to provide contrast to the screen background colour: with light colours used against the dark non-scrolling header, and dark colours against the light background used for the remainder of the screen.

- Jumps to help screens other than those specified in the header were linked to a word or combination of words within the body of the text. These were underlined (following the usual convention for hyperlinks) and formatted in bold text of a different colour to the rest of the text.

Figure 2.2.1: Example Q-by-Q Screen, showing the hierarchical structure of the Table of Contents, and the formatting of the non-scrolling header [dark pane] and the provision of jumps [underlined text].



Initially, icons were considered for indicating the four associated screens in the header, as the limited use of icons had proved successful within Blaise Screen and Layout Standards introduced at ONS in April 2004. Ultimately, however, these were dispensed with in favour of text, as the icons selected proved not to be intuitive, as there had not been any pre-existing text to provide a reference against.

Table of Contents

The Table of Contents was similarly structured to the actual help document, following the flow of questions throughout the questionnaire. All pages directly relating to a question were placed within a book with the name of that question, and these books were categorised first by topic, and then these topics by whether they formed part of the core questions of the IHS or part of one of the modules formed from the compoent questionnaires. All help screens not directly related to the questions asked were placed together in the contents in a general help category. Similarly, all definitions were also placed together into their own book.

2.3. Problems with approach

Though they addressed the needs of users, both the house style and the choice of authoring tool created a very labour-intensive production. The process was divided into two parts: (1) composing and collating of instructions into a single Word document, undertaken by the survey team, and (2) processing of the file through RoboHelp, undertaken by the team holding the product licence: in this case, this was the ONS Blaise Development Standards & Support Team.

Creating the files for the IHS took two months to produce. Much of this time was invested in the first part of the process, getting the document correctly formatted both by division of information and also layout. Some of this, particularly in reformatting and rewriting long-standing documents, can be seen in terms of short-term initial start-up costs. Similarly, the maintenance of the Word document containing the reformatted instructions for processing through RoboHelp should prove to be minimal.

There were disadvantages, however, in using a single-source authoring tool, such as RoboHelp, in that it is particularly demanding of system resources. With just under 1,400 pages in the IHS help Word document, the processing time increased with each page processed. Though it was possible to split the Word file so that each part could be processed concurrently by more than one person, the final project file with its contents could only be processed as a single document.

Since the successful introduction of Q-by-Q on IHS, it has begun to be rolled out on the individual surveys ahead of their incorporation into the IHS. Some of the work in preparing Q-by-Qs for these surveys has been reduced as relevant sections from within the IHS version have used or adapted. Despite this, however, the initial setting up of these Q-by-Qs, even with some of the work being already untaken, is still intensive. As previous papers have shown, this is one drawn-back of producing WinHelp Q-by-Qs, and cannot be mitigated without limiting the functionality and design of the finished product.

2.4. Functionality and Usability

Keystroke versus Mouse navigation

Until recently, at ONS, interviewer laptops had not been provided with an external mouse, and where a mouse panel was incorporated on the laptop model this was deactivated, as keystroke navigation was seen as giving greater control when interviewing within respondents' homes. Since 2005, such mouse panels have been

activated, allowing interviewers the option of navigating by keystroke or by mouse click.

WinHelp Q-by-Qs lend themselves more easily to mouse navigation. Keystrokes can navigate the user around the panes of the WinHelp screen, though these are less obvious, particularly in the order of rotation on the Index tab. ONS found it was, therefore, important that both methods of navigation – particularly that by keystroke – were clearly explained in supporting documents for the Q-by-Q.

Annotation and Bookmarks

Some consideration was given to the desirability of allowing interviewers to bookmark useful pages and to annotate pages within the Q-by-Q with their own notes and aide-memoires. There had been a concern that bookmarks might lead interviewers to an over-reliance on these pages and possibly may lead to them overlooking relevant information on other pages that may be circumvented. Similarly, there were concerns of an over-reliance on the shorthand annotations rather than the full instruction on the page.

Both, however, proved popular with interviewers, as they replicated the option of making notes on the previous paper documents, and also those training and monitoring the interviewers supported their use. The influence of annotation and bookmarks on adaptation of instructions still needs further investigation.

Contents, Index and Search tabs

When Q-by-Q was first looked at by ONS, it was simply in order to provide a quick searchable help facility within the questionnaire. As it developed, so the potential of it also being used as a manual or pre-fieldwork learning document came to be seen. Whereas the Search tab was originally seen as the main way interviewers would access and use the Q-by-Q, the Contents tab (in which the full document was listed categorised in logical order) came to be recognised as the interviewers' preferred way of using the Q-by-Q at home.

The Index tab (listing all pages in alphabetical order) seemed at first to have the least useful purpose. Though as pages were named by Field Identifier, this tab proved useful for interviewers using the Q-by-Q as a reference manual at home.

Jumps and mid-topic IDs

Jumps provided in the Q-by-Q only referenced other pages within the document. Jumps primarily existed within the headers to link the four associated pages relating to a single question. Other jumps were kept to a minimum, and were only provided when referencing another page not directly related to that question (such as a definition).

The option of using mid-topic IDs (jumps to a specific place on a given page) was investigated early on, though were not incorporated within the final version as overly-complex and potentially confusing for interviewers.

2.5. Feedback and success

Interviewers were specifically asked to provide feedback on the Q-by-Q, in order to evaluate its success. The overwhelming majority of comments from interviewers were positive and welcoming of the innovation. Typical comments included those listed below.

- "I was unsure whether this [an apprenticeship for ground working] would be strictly accurate but [the] help screen clarified this for me very easily."
- "I used the help screens twice to check qualifications: very straightforward and easy to use."
- "It was good to be able to confirm topics [on which] I had only partial/ incomplete knowledge."
- "Looked up definition of annualised hours contract: very useful. Respondent now aware of its meaning."
- "Felt a little alien at first, but soon got used to its use."
- "New [interviewer] recruits would benefit from this."

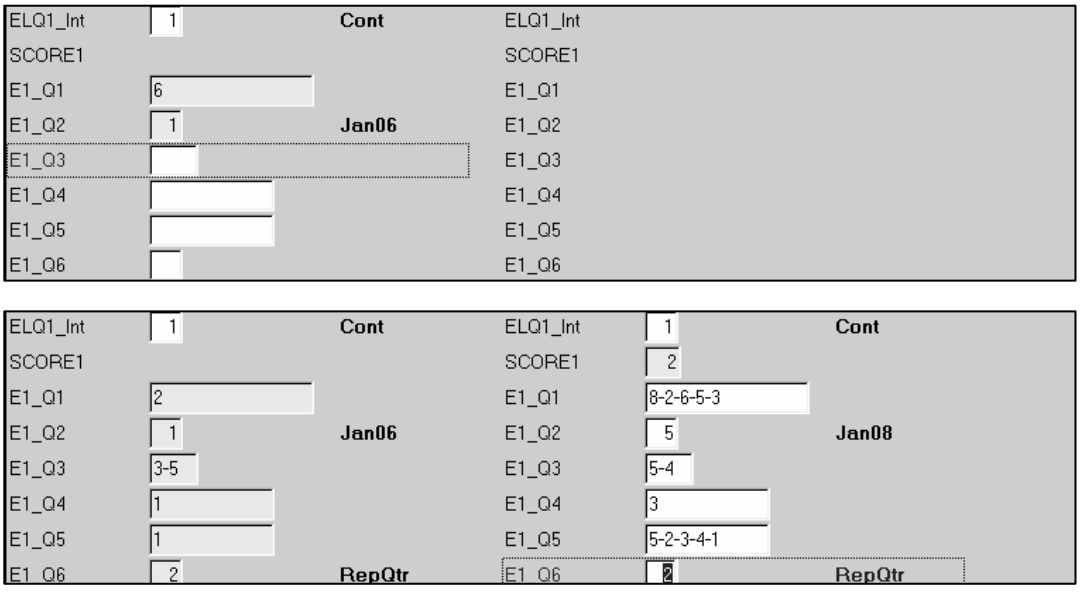
The only negative comments highlighted limitations in the written instructions and miscoded jumps, enabling them to be amended for the subsequent pilot. Two interviewers also highlighted a limitation in using any help document in the interview situation – no matter how refined – as it could interrupt the flow of the interview. One summarised the concern as "Generally, anything which slows the interview down, even by seconds, is bad news as far as I'm concerned, unless the respondent is really relaxed about their time being interviewed."

3. Electronic Learning Questionnaires

At the 2003 IBUC, Statistics Norway presented its work in using Blaise to create interactive training instruments. Some of these principles were incorporated into ONS's work on Electronic Learning Questionnaires, which also designated a separate language within the Blaise Datamodel, in order to display images on screen and to play audio-recordings. The method for incorporating these features is detailed in Hilde Degerdal's paper (2003) and was used by ONS.

The first version of the ELQ was produced for the second IHS feasibility test in August 2005. Sets of questions were prepared for each questionnaire combination within the IHS.

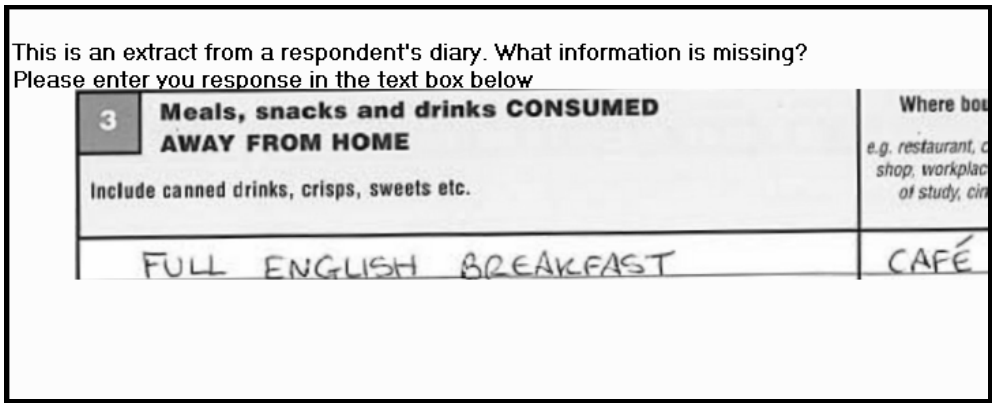
Figure 3.1: Example first version of ELQ: first showing the first loop in the array in which the interviewer would insert her answers; secondly showing the second loop of the array in which the score and the correct answers were automatically displayed.



The questionnaire was programmed to take the interviewer through the questions, once the question was answered the Field was changed from ASK to SHOW, each answer was scored, and then the interviewer was taken through the question a second time and provided with the correct answers. A similar method was used at the first IHS pilot in November 2005.

Multimedia functions were introduced at the second IHS pilot in February 2006. Doorstep scenarios were audio-recorded for playing at particular questions in the ELQ in order to test how interviewers would respond to situations, and images (such as excerpts from food diaries) were displayed on screen in order to test whether interviewers could correctly recognise errors contained on them. Problems were experienced with the size of the image (*.BMP) and sound (*.WAV) files, resulting in extended transmission times for downloading files to interviewers' laptops; alternative methods of transmission (electronic and postal) still need investigating.

Figure 3: Example of a Bitmap image file used in the ELQ



Additionally, the method of sealing an interviewer's answers on entry and displaying the correct answers once all the questions in a series were answered was replaced. This method had proved frustrating to interviewers who found that an accidental miskeying of an answer could not be changed or who had to wait till all questions were answered before learning why an early answer given was incorrect.

The revised method allowed the interviewer two attempts at the question. If the first attempt was incorrect, a 'hint' of how to locate the correct answer was displayed on screen by means of variable text and a second attempt was allowed. If the second attempt also proved incorrect, the correct answer would then also be displayed on screen by means of variable text.

4. Summary

Extending the previous use of Blaise within ONS beyond basic questionnaire provision has proved successful. Q-by-Q has been demonstrated as not only an effective way of providing help within the interview but also has been used as part of the training material prior to fieldwork and as a reference manual for use at home during the field period.

ONS's experience of producing a WinHelp Q-by-Q highlighted the same issues of complicated and time-exhausting production highlighted by other organisations. In ONS's experience, though, the benefits of providing a more interactive and searchable help facility as well as the comparative reduction in effort to maintain the Q-by-Q has outweighed the initial start-up costs. On the basis of its success on the IHS, other surveys within Blaise have begun to move towards similar Q-by-Qs for their questionnaires.

Similar to recent work on screen layouts, ONS found that it was important that certain standards should be agreed in order to maximise the usefulness of the Q-by-Q. Key considerations were packaging instructions into smaller less wordy divisions, and group instructions into common categories for easier and quicker reference by interviewers.

Pages within the Q-by-Q should be given useful names within the authoring tool in order to aid quick navigation to pertinent when it is needed. Each of the Contents, Index and Search tabs within the Q-by-Q proved useful depending on how the system was being used (at home or in the field, as training or for reference), and therefore it was important that naming conventions for information in each tab reflected that use.

Further work is needed on some aspects to ensure that familiar use of bookmarks and annotations does not limit the correct reliance on the full instructions.

Finally, ELQs proved a useful tool for evaluating the success of an interviewer's learning, though there continue to be outstanding technical issues on how to best transport the larger files to interviewers.

5. References

Bumpstead, R.A. (2003): Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey, Office for National Statistics, UK, Presented at 9th International Blaise Users Conference, 2004, Gatineau, Ottawa, September 2004.

Couper, M.P., Beatty, P., Hansen, S.E., Lamias, M., Marvin, T. (2000): CAPI Design Recommendations: Report submitted to the Bureau of Labour Statistics, University of Michigan.

Degerdal, H. (2003): Interactive Training for the interviewers, Statistics Norway, Presented at the 8th International Blaise Users Conference 2003, Copenhagen, May 2003.

Gatward, R.M. (2003): Developing and updating screen layout and design standards, ONS, UK, Presented at the 7th International Blaise Users Conference, 2003, Copenhagen, May 2003.

Wensing, F., Barresi, J., Finlay, D. (2003): Developing an optimal screen layout for CAI, Australian Bureau of Statistics, Presented at the 8th International Blaise Users Conference 2003, Copenhagen, May 2003.

Session 1.2: History of Blaise

Blaise – Alive and Kicking for 20 Years
Jelke Bethlehem & Lon Hofman
(Statistics Netherlands)

Blaise – Alive and kicking for 20 years

Jelke Bethlehem & Lon Hofman (Statistics Netherlands)

1. Introduction

The first version of the Blaise System for computer-assisted data collection was released in 1986. So, the year 2006 marks the 20-th birthday of the system. Although operating systems have changed over time, and new data collection techniques have emerged, the basic ideas and concepts of Blaise have remained the same over the years. The heart of the system, the Blaise language, has proved to be a powerful means to define data collection instruments. Designed in a time where most statistical agencies still used mainframe computers, and the first MS-DOS computers came on the market, Blaise was able to keep up with rapid developments in information technology, and still plays a leading role in the world of data collection for official statistics.

This paper gives an account of the 20 year long history of the Blaise System. It described why it was developed and how it evolved over time.

2. The Data Editing Research Project

It is the task of national statistical offices like Statistics Netherlands to collect data on sources like persons, households, farms and establishments, and to transform these data into relevant, accurate and timely statistical information. There is an ever growing demand for all kinds of statistical information, and this calls for an organised and efficient approach to the entire process of data collection, data analysis and publication. Still, executing a survey and processing the collected information can be expensive and time-consuming.

Data editing is a vital component of the survey process. Data editing activities attempt to detect and correct errors in the individual records, questionnaires or forms. These activities are carried out with the intention to improve the quality of the results of surveys. Since statistical offices always attach much importance to this aspect of the survey process, a large part of human and computer resources are spent on it.

Notwithstanding a long time experience Statistics Netherlands had not much insight in the nature of the data editing activities, back in the early 1980's. Therefore, it started the Data Editing Research Project in 1984. Editing procedures of four surveys were investigated. Objective of this study was to get insight in the different kind of activities carried out during the data editing process, the frequency of occurrence of these activities, the time spent on them, the number and qualifications of the people involved, and the effect on data quality. The data editing research project is described in more detail in Bethlehem (1987).

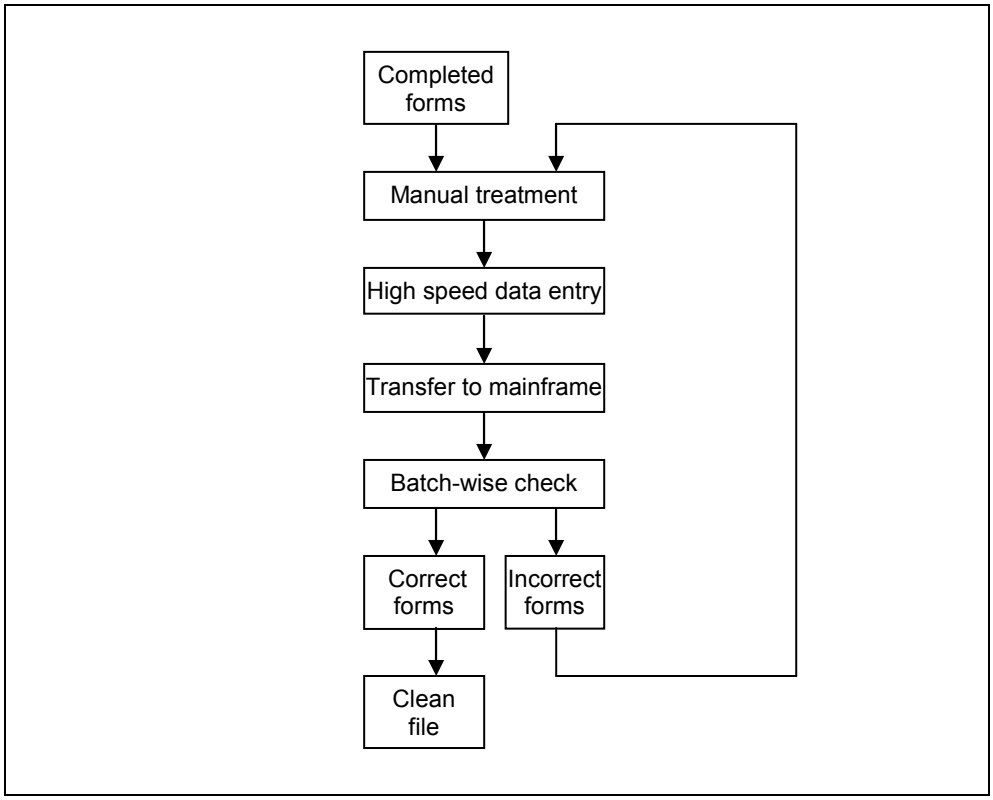
Four surveys were selected in the study: two economic surveys, the Foreign Trade Survey and the Survey on Transport of Goods by Road, and two social surveys, the Labour Force Survey and the Survey on Well-being of the Population. Two surveys (one economic and one social) were very large. For these surveys only an extensive inventory of all activities was made. The two other surveys (one economic and one social) were of moderate size. For these surveys not only a

detailed process analysis was carried out, but, because of their size, it was also possible to carry out a comprehensive analysis of the data produced by the surveys. Considerable differences were observed between the editing procedures of economical and social surveys. In most economical surveys, the questionnaire was straightforward. Questions were answered one after another without complex routing structures. Many range checks and consistency checks were carried out. Totals of individual entries had to be calculated or checked. Often figures of firms were confronted with corresponding figures from previous years. To correct detected errors, it was sometimes possible to contact firms, either by telephone or by an employee of Statistics Netherlands which visited the firm. In short, economical surveys could be characterised by simple questionnaires with comprehensive checking.

Social survey questionnaires tended to be large with complex routing structures. Error checking mainly concerned the route followed through the questionnaire and some range checks. Only a few consistency checks were carried out. To be able to correct detected errors, contact was necessary with the supplier of the information. Since this was hardly ever possible, generally correction resulted in setting a value to 'unknown'.

Although the data editing process differed from survey to survey, still some general characteristics could be observed which held for nearly all surveys. The general scheme of the editing process is summarised in figure 2.1.

Figure 2.1. The traditional approach to data editing



After all forms had been collect, subject-matter specialists checked the forms for completeness. If necessary and possible, skipped questions were answered, and obvious errors were corrected on the forms. Sometimes, the forms were manually copied to a new form to allow for the subsequent step of fast data entry. Next, the forms were transferred to the data entry department. Data typists entered the data in the computer at high speed without error checking. The computer was a dedicated

system for data entry. After data entry, the files were transferred to the mainframe computer system. On the mainframe an error detection program was run. Usually, this was a dedicated COBOL program. This program produced a list of detected errors. This list was sent to the subject-matter department. Specialists investigated the error messages, consulted corresponding forms, and corrected errors on the lists. Corrected forms were sent to the data entry department, and data typists entered the corrections in the data entry computer. The file with corrections was transferred to the mainframe computer. Corrected records and already present correct records were merged. The cycle of batch-wise error detection and manual correction was repeated until the number of detected errors was considered to be sufficiently small.

After the final step of the editing activities, the result was a 'clean' data set, which could be used for tabulation and analysis. Detailed investigation of the data editing procedures for the four selected surveys lead to a number of conclusions. These conclusions are summarised below.

- Various people from different departments were involved. Many people dealt with the information: respondents filled in forms, subject-matter specialists checked forms and corrected errors, data typists entered data in the computer, and programmers from the computer department constructed editing programs. Transfer of material from one person/department to another could be a source of error, misunderstanding and delay.

- Different computer systems were involved. Most data entry was carried out on Philips P7000 minicomputer systems, and data editing programs ran on a CDC Cyber 855 mainframe. Furthermore, there was a variety of desktop (running under MS-DOS) and other systems. About 300 interviewers had been equipped with laptops running under CP/M. Transfer of files from one system to another caused delay, and incorrect specification and documentation could produce errors.

- Not all activities were aimed at quality improvement. A lot of time was spent just on preparing forms for data entry, and not on correcting errors. Subject-matter specialists had to clean up forms to avoid problems during data entry. The most striking example was assignment of a code for 'unknown' to unanswered questions.

- The process was going through *macro cycles*. The whole batch of data was going through cycles: from one department to another, and from one computer system to another. The cycle of data entry, automatic checking and manual correction was in many cases repeated three times or more. Due to these macro cycles, data processing was very time consuming.

- The structure and nature of the data had to be specified in nearly every step of the data editing process. Although essentially the same, the 'language' of this meta-data specification could be completely different for every department or computer system involved. The questionnaire itself was the first specification. The next one was with respect to data entry. Then, automatic checking program required another specification of the data. For tabulation and analysis, e.g. using the statistical package SPSS, again another specification was needed. All specifications came down to a description of variables, valid answers, routing and possibly valid relations.

There are a lot of similarities between statistical production processes as described above and business processes in general. Therefore, approaches to improving business processes can also be applied to the survey process. Two such approaches

are discussed here. One is the theory of Deming on improving quality in industrial production, and the other is the more recent Business Process Redesign (BPR) approach.

Quality control has always been an important issue in industrial production. The theory of Deming (1986) about improving quality and productivity in industry is well-known. Many of his famous 14 points for management also apply to the production of statistical information. One of these points states that one should cease dependence on mass inspection. Inspection of final products to improve quality is too late, ineffective and costly. Quality must be built in at the design stage. These statements particularly apply to data collection and data editing. By trying to detect and correct errors in answer to questions well after they have occurred, one fails to locate the source of the error, and consequently, these errors can not be eliminated.

Some of the problems mentioned by Deming could be solved by introducing *computer-assisted interviewing* techniques. The rapid advent of the microcomputer in the 1980's made it possible to use microcomputers for computer-assisted interviewing. The paper questionnaire was replaced by a computer program asking the relevant questions. The computer took control of the interviewing process. It could perform two important activities:

- *Route control.* The computer program determines which question is to be asked next, and displays that question on the screen. Such a decision may depend on the answers to previous questions. Hence it relieves the interviewer of the task of taking care of the correct route through the questionnaire. As a result, it is not possible anymore to make route errors.

- *Error checking*. The computer program checks the answers to the questions which are entered. If an inconsistency is detected, the program gives a warning, and one or more of the answers concerned can be modified. The program will not proceed to the next question until all detected errors have been corrected.

The concepts of Business Process Redesign (or Re-engineering) are well described in the book by Hammer & Champy (1994). They claim that many of today's companies operate on principles formulated more than two centuries ago by Adam Smith in his book "The Wealth of Nations". These principles worked in an expanding market for mass products and services, but they offer not guarantee to survive in today's quickly changing market conditions and competitive climate. The traditional approach focuses on splitting business processes in simple tasks. The advantage of such a task oriented approach is that the relative simple tasks do not require advanced skills or knowledge. However, simple tasks demand complex processes to knit them all together. For two hundred years, companies have accepted the inconvenience, inefficiencies and costs associated with these complex processes in order to be able to take advantage of the benefits of simple tasks.

Typically, a task oriented process involves different people in different departments. The processes are sequential in the sense that one department cannot start working on a task until another department has finished it. Moreover, some business processes have iterative components. The process goes through a cycle several times before it can be completed. Such sequential and iterative processes are time-consuming and error prone.

Preaching quality is not sufficient to improve business processes. Even if each separate task is carried out efficiently, the process as a whole need not be. Application of the developments of state of the art information technology will not help if it is only used to automate current tasks. More is needed to accomplish

substantial improvements. Business Process Redesign means "the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service, and speed".

BPR stresses the importance of the creative use of information technology. However, new developments in information technology should not be used just to automate the tasks in the traditional processes. This will only lead to marginal performance improvements. Far more substantial improvements can be obtained by using computer technology in a fundamental new way.

There is a striking degree of similarity between the descriptions of the traditional survey process and that of the traditional business process. Both processes share the same problems: process fragmentation, focusing on tasks instead of processes, involvement of several departments, sequentially and cycling, it is all there. Statistics Netherlands followed a BPR-like approach in redesigning its statistical production processes.

3. Blaise 1: Integrating data entry and data editing

The first phase of the redesign of the survey processes of Statistics Netherlands concentrated on the data editing activities. The idea was to improve the handling of paper questionnaire forms by integrating data entry and data editing tasks. The traditional batch-oriented data editing activities, in which the complete data set was processed as a whole, was replaced by a record-oriented process in which records (forms) were completely dealt with one at a time.

The new group of activities was implemented in a so-called CADI system. *CADI* stands for Computer Assisted Data Input. CADI was an intelligent and interactive system for combined data entry and data editing. The CADI system was designed for use by the workers in the subject-matter departments.

Data could be processed in two ways by this system. In the first approach, subject-matter employees worked through a pile of forms with a microcomputer, processing the forms one by one. They entered the data, and after completion of the form, they activated the check option to test for all kinds of errors. Detected errors were reported and explained on the screen. Errors could be corrected by consulting forms, or by contacting the suppliers of the information. After elimination of all errors, a 'clean' record was written to file. If an employee did not succeed in producing a clean record, he could write the record to a separate file of ‘dirty’ records. A specialist can deal with these hard cases later on, also using a CADI system.

In the second approach, data typists used the CADI system to enter data beforehand without much error checking. After completion, the CADI system checked in a batch run all records, and flagged the incorrect ones. Then subject-matter specialists handled these ‘dirty’ records one by one, and correct the detected errors.

To be able to introduce CADI on a wide scale in the organisation, a new standard package was developed. The name of this standard package was *Blaise*. It derived its name from the famous French theologian and mathematician Blaise Pascal (1623-1662). The basis of the Blaise System was the Blaise language, which was used to create a formal specification of the structure and contents of the questionnaire. The Blaise language had its roots, for a large part, in the programming language Pascal.

The first version of the Blaise System ran on microcomputers (or networks of microcomputers) under MS-DOS. It was intended for use by the people of the subject-matter departments, therefore no computer expert knowledge was needed to use the Blaise system. The design goal of the system was to provide subject-matter experts with a powerful but user-friendly tool that enabled them to input their knowledge about a survey into the system, and to take care of all subsequent data processing steps.

In the Blaise philosophy, the first step in carrying out a survey was to design a questionnaire in the Blaise language. Such a specification of the questionnaire contains more information than a traditional paper questionnaire. It did not only describe questions, possible answers, and conditions on the route through the questionnaire, but also relationships between answers that had to be checked.

The Blaise language had to be powerful enough to be able to deal with large and complex surveys. Since the specification was used to define the survey, the language had to be readable enough. In fact, a Blaise questionnaire had to be self-documenting, i.e. it is the basic description of the survey which could be used by all people involved. Figure 3.1 gives an example of such a simple questionnaire in Blaise.

Figure 3.1. A simple Blaise 1.0 questionnaire specification

```
QUESTIONNAIRE LFS "The Labour Force Survey";

QUEST
    SeqNum  (KEY) "Sequence number of the interview?": 1..1000;
    Age     "What is your age?": 0..99;
    Sex     "Are you male or female?": (Male, Female);
    MarStat "What is your marital status?":
            (Married "Married",
             NotMar  "Not married")
    Job      "Do you have a job?": (Yes, No);
    JobDes   "What kind of job do you have?": STRING[20];
    Income   "What is your yearly income?":
             (Less20  "Less than 20,000",
              Upto40  "Between 20,000 and 40,000",
              More40  "More than 40,000");

 ROUTE
    SeqNum; Age; Sex; MarStat; Job;
    IF Job = Yes THEN
       JobDes; Income
    ENDIF

 CHECK
    IF Age < 15 "Respondent is younger than 15" THEN
       MarStat = NotMar "he/she is too young to be married !"
    ENDIF

 ENDQUESTIONNAIRE.
```

The first part of the questionnaire specification was the *Quest section*, containing the definition of all questions that can be asked. A question consists of an identifying name (for internal use in the questionnaire), the text of the question as presented to the respondents on the paper form, and a specification of valid answers.

The next part of this sample Blaise questionnaire is the *Route section*. It described under which conditions, and in which order the questions had to be asked.

Consistency checks were specified in the *Check section*. They checked whether a specified condition was true for the answers to the questions involved. If the condition was not satisfied, an error message was generated.

Figure 3.2. The main screen of Blaise 1.1

Preparation of a CADI program required a number of steps. Figure 3.2 shows the development environment. First, the questionnaire specification was entered using a text editor. The default editor was PC-Write 2.71. Next, the parser was activated to check the questionnaire for syntax errors. If no errors were encountered, the questionnaire specification was transformed in Turbo Pascal source code. Then, the Pascal code was compiled using a Turbo Pascal compiler (Turbo Pascal 3.0), and the result was an executable program. This program was called the *CADI machine*. Figure 3.3 shows how the CADI machine of the specification in figure 3.1 looked like.

Figure 3.3. A Blaise 1.1. CADI machine

The CADI program could be used in three different ways to process survey data:

- Subject-matter specialists entered and simultaneously edited the data. Figure 3.3 shows an example of a screen after a form had been entered. Note that questions Age and MarStat have error counters in front of their input fields. For these questions an inconsistency has been detected.

- Data entry typists entered the data at high speed, without much error checking. After completion, the CADI program carried out an integral check on all records, and flagged the incorrect ones. Then subject-matter specialists corrected the wrong forms in an interactive session.

- If a data file was created and supplied by another agency, the data could be imported in the Blaise system, the CADI program carried out an *integral check*, thereby flagging incorrect records. The subject-matter specialists did the interactive editing of the incorrect records.

Version 1.0 of Blaise was completed and taken into production in 1986. After an initial period in which people had to get used to the new way of working, the new approach was accepted enthusiastically. The Blaise system turned out to be sufficiently flexible for survey processing activities in many departments. Use of Blaise lead to greater speed, lower operating costs, and to an improvement in data quality, even at a time where Statistics Netherlands faced substantial reductions in staff.

The benefits for the people in subject-matter departments were notable. The work gave them much greater scope. They appreciated the increased autonomy and responsibility in organising their work. It lead to reduced specialisation, the acquisition of a wider range of skills, and the possibility to adapt and devise systems for their own purposes. These advantages generally outweighed the need, in some cases, to undertake lengthy periods of training, and a reluctance on the part of some to work with computers for long periods.

4. Blaise 2: Computer-assisted interviewing

After version 1 of Blaise had been in use for a short time, it was realised that such a system could be made much more powerful. The questionnaire specification in the Blaise system contained all knowledge about the questionnaire and the data needed for survey processing. Therefore, Blaise should be capable to manage other data processing applications. As a next step, the system was extended to be able to handle computer assisted interviewing.

In computer assisted interviewing, the paper questionnaire is replaced by a computer program containing the questions to be asked. The computer takes control of the interviewing process. It performs two important activities:

- *Route control*. The computer program determines which question is to be asked next, and displays that question on the screen. Such a decision may depend on the answers to previous questions. Hence it relieves the interviewer of the task of taking care of the correct route through the questionnaire. As a result, it is not possible any more to make route errors.

- *Error checking*. The computer program checks the answers to the questions which are entered. Range checks are carried out immediately after entry and consistency checks after entry of all relevant answers. If an error is detected, the program gives a warning, and one or more of the answers concerned can be

modified. The program will not proceed to the next question until all detected errors have been corrected.

Application of computer assisted data collection has three major advantages. In the first place, it simplifies the work of interviewer (no more route control), in the second place, it improves the quality of the collected data, and in the third place, data is entered in the computer during the interview resulting in a clean record, so no more subsequent data entry and data editing is necessary.

Version 2.0 of Blaise was completed in 1988. It implemented just one form of computer assisted interviewing: CAPI. CAPI stands for Computer Assisted Personal Interviewing. It is a form of face-to-face interviewing in which interviewers use small laptop or notebook computer to ask the questions and to record the answers.

Dutch experiences with CAPI started in 1983. The first experiments showed that respondents had no objections against computer assisted interviewing, and the interviewers developed a positive attitude towards this way of working. Use of computers did not increase nonresponse.

Due to the success of these experiments, Statistics Netherlands started in 1987 with full scale use of CAPI in a regular survey: the continuous Labour Force Survey. More details about these CAPI surveys can be found in Van Bastelaer et al. (1987a, 1987b) and Bemelmans-Spork and Sikkel (1985a, 1985b). The interviewing programs in the early experiments were dedicated, tailor-made programs, and later based on a more generic system (Quest4). But from 1990 on all CAPI surveys were managed by the Blaise system. The most important one was the Labour Force Survey. Each month, about 400 interviewers equipped with laptops visited 12,000 addresses. The laptop was a Toshiba T1000, see figure 4.1.

Figure 4.1. A Toshiba T1000 laptop computer

After a day of interviewing, they returned home and connected their computers to the power supply to recharge the batteries. The laptop computer was also connected to a telephone and modem. At night, the collected data were automatically transmitted to the office, and in return, new addresses were sent to the interviewers. The next morning, they would find a recharged machine with a clean workspace, ready for a new day of interviewing.

Figure 4.2 shows an example of a screen of a CAPI program generated by Blaise 2.0. The screen is divided in two parts. The upper part contains the current question to be answered. As soon as the answer has been entered, this question is replaced by the next question on the route.

Figure 4.2. A Blaise 2.3 CAPI program

Just displaying one question at the time only gives the interviewers a very limited feed-back on where they are in the questionnaire. Therefore, the lower part of the screen displays (in a very compact way) the current page of the questionnaire.

The CAPI program applied dynamic routing. No questions could be skipped that had to be answered, and no questions could be answered that were not on the route. Thus the routing structure of the questionnaire could not be violated.

The CAPI program also applied dynamic error checking. As soon as a consistency error was encountered, an error message appeared on the screen, including a list of all questions involved. The interviewer could jump back to a question in order to correct its answer. It was not possible to continue to a new question until all errors has been corrected. Therefore, when the end of the interview was reached, no more errors remained in the form.

In 1990, another mode of computer assisted interviewing was included: CATI, or Computer Assisted Telephone Interviewing. Thus, version 2.3 of Blaise was born. In the case of CATI, the interviewing program is installed on a desktop computer. The interviewer called respondents from a central unit (call centre), and carried out the interview by telephone. The computer program selected the proper question to be answered, and displayed it on the computer screen. The interviewer read out the question, and entered the answer provided by the respondent. The program checked for all kinds of errors. If an error was detected, the computer warned the interviewer that something is wrong, and corrections could be made. See e.g. Nicholls and Groves (1986a, 1986b) for more details about CATI.

An important component of a CATI survey is the call scheduling system. This system must take care of proper handling of busy numbers, no-answers, appointments, etc. Such a scheduling system was automatically made available by the Blaise system when it created a CATI survey.

In the very early 1990's nearly all household surveys of Statistics Netherlands had become CAPI or CATI surveys. Surveys using paper forms had almost become extinct. Figure 4.1 lists all major and regular household surveys at that time together with their mode of interviewing.

Table 4.1. Household surveys carried out by the CBS

Survey	Mode	Interviews per year
Survey on Quality of Life	CAPI	7,500
Health Survey	CAPI	6,200
Day Recreation Survey	CAPI	36,000
Crime Victimisation Survey	CAPI	8,000
Labour Force Survey	CAPI	150,000
Car Use Panel	CATI	8,500
Consumer Sentiments Survey	CATI	24,000
Social-Economic Panel	CATI	5,500
School Career Survey	CATI	4,500
Mobility Survey	CATI / CADI	20,000
Budget Survey	CADI	2,000

An important new feature in version 2.3 of Blaise was the *Setup Generator*. This tool assisted in preparing data collected with the Blaise for subsequent processing, like tabulation and analysis. Statistics Netherlands had adopted the principle that whenever good software is available, wheels should not be reinvented by developing similar software itself. Particularly, this applied to software for statistical analysis. However, using available software has the drawback that it is usually necessary to put the data, and the description of the data (the metadata) into a different format. Version 2.3 of Blaise took care of that. The Setup Generator was able to use the information in the Blaise questionnaire specification to create setups for the tabulation package Abacus, the statistical packages SPSS, SAS and Stata, and to prepare data for downloading in database packages like Paradox and Oracle.

To prepare data for processing by other software, the Setup Generator used instruction files. Blaise contained instruction files for the software packages mentioned above. For other, not mentioned software, users could generate new instruction files for their own favourite software packages. Thus, Blaise offered a flexible approach to adapt the system for work in a specific environment.

Figure 4.2 shows an example of setup script for SPSS. Such a script partly consists of fixed text and partly of setup instructions. The fixed text (e.g. `VAR LABELS`) was just copied to the setup file. Instructions were placed between square brackets. They used information that was retrieved from the questionnaire specification file (e.g. `QuestionName`).

The ideas behind the Setup Generator language were similar to those behind XML / Sthyle sheets. However, at that time XML did not yet exist. The Setup Generator was an XML application 'avant la lettre'.

Figure 4.2. Setup generator script for SPSS

```
[SubFileLoop]
  [OutFile := SubFileBlockName + '.SPS']
  []TITLE           '[SubFileBlockName]'.
  []DATA LIST       FILE = '[FileName].[SubFileExtension]' /
  [QuestionLoop]
    [if not FirstQuestion then][/][endif]
    [:16][QuestionName 16:] [FirstPosition:4] -[LastPosition:4][&]
    [if QuestionType=Open or QuestionType=Code then] (A)[&]
      [elseif QuestionType=Subrange and NumberOfDecimals>0
      then] ([NumberOfDecimals])[&]
    [endif]
  [EndQuestionLoop]
  [].

  []VAR LABELS
  [QuestionLoop]
    [If QuestionLabel <> '' Then]
      [if not FirstQuestion then][/][endif]
      [:16][QuestionName 16:]'[QuestionLabel]'[&]
    [EndIf]
  [EndQuestionLoop]
  [].

  []VALUE LABELS[&]
  [TypesLoop]
    [if TypesIndex>1 then]/[endif][&]
    [QuestionLoop]
      [if QuestionTypeNr=TypesIndex
        then][/][:16][QuestionName 8:][&]
      [endif]
    [EndQuestionLoop]
    [AnswerLoop]
      [if AnswerIndex>1 then][/][:24][endif]
      [AnswerCode :8] '[AnswerLabel]'[&]
    [EndAnswerLoop]
  [EndTypesLoop]
  [].

  []LIST            /CASES TO 10.
  []SAVE            /OUTFILE '[SubFileBlockName].SYS'.
[EndSubFileLoop]
```

Finally, version 2.5 of Blaise emerged in 1992. One of the major enhancements included in this version, was *trigram coding*. One of the time-consuming activities of the traditional survey process is the coding of open answers. A typical example is the question about the occupation of the respondent. Questions are easiest to process if a respondent selects one possibility from a list of pre-coded answers. However, for a question like occupation, this set of pre-coded answers would be very long, and thus it would be very hard for the respondent to select the proper answer. This problem is avoided by letting the respondent formulate his own answer, and then literally copying the answer on the form. To enable analysis of this type of information, answers must be classified afterwards. This coding was usually carried out by experienced subject-matter specialists.

The Blaise System already contained a tool for *interactive coding*, thereby providing the possibility of integrating coding either in the data collection phase or in the data entry and data editing phase. The coding module could be used in two different ways. One way was *hierarchical coding*. Coding of an answer starts by entering the first digit of the code by selecting the proper category from a menu. After the user had entered a digit, the program presented a subsequent menu containing a refinement of the previously selected category. So the description became more and more detailed until the final digit was reached. Another way to

code a question was to apply the dictionary approach. This approach simply tried to locate an entered description in an alphabetically ordered list. New in version 2.5 of Blaise was trigram coding. A trigram is a sequence of three subsequent characters appearing in a word. For example, the word ‘coder’ can be decomposed into the trigrams 'cod', 'ode' and 'der'. An entered description is decomposed in the set of all of its trigrams. Also all descriptions in the dictionary are stored in the form of trigram sets. The system now attempts to find a description in the dictionary by locating trigram sets that have a high percentage of trigrams in common. Thus, a description will even be found if spelling errors are made, or if an alternate spelling is used. All these coding techniques provided a better treatment of open questions. They improved data quality, and speeded up the survey process, particularly if coding could be carried as part of the interview.

5. Other developments

During the development of Blaise 1 and Blaise 2 other projects were initiated that were loosely linked to Blaise and played a roll in the use of the data captured with Blaise. Worth mentioning are Manipula, Abacus and Bascula.

5.1. Manipula

The Manipula project started in 1989 during the development of Blaise CATI call management system. Users experienced problems in preparing data files required as input for this system. Programs were available on the mainframe computer to convert and reformat files, but on the PC they were lacking. The first version of Manipula was developed to do this job.

Manipula was designed for manipulating ASCII and Blaise files under MS-DOS. The program could be used to link files, to select fields and records, to do computations, to detect errors, to make print files and for sorting and summarising. Manipula uses a *setup*. In this setup the user gives a description of one or more input files, the output file and the manipulations to be carried out on the data. A setup is specified in the Manipula language. A simple example of a setup can be found in figure 5.1

Figure 5.1. A simple Manipula 1.0 setup

```
INPUTFILE "samplon.dat"
  city, 1, 1, STRING;
  province, 3, 1, STRING;
  sex, 5, 1, STRING;
  age, 7, 2, STRING;
  employed, 10, 1, STRING;
  income, 12, 4, STRING;
OUTPUTFILE "samplon.exl"
  sex, 1, 1, STRING;
  age, 3, 2, STRING;
  income, 6, 4, STRING;

MANIPULATE
  IF ((citys'1') OR (city='2') OR (city='3') AND
     (employed='1')
  THEN
    SELECT
  ENDIF;
```

The setup above describes a selection of fields for certain records in an ACII data set. Manipula was a tool developed independently of Blaise which also had its own way of describing the data.

5.2. Abacus

Blaise 2.0 also contained the tabulation package Abacus. Abacus had a very efficient counting algorithm. This made it possible to produces tables out of very large data files. And the tables produced, were camera-ready.

Abacus was a user-friendly package allowing tables to be specified in an interactive way. This was an innovation. Most tabulation packages at that time used, sometimes complex, specification scripts.

Three-dimensional tables could be specified (rows, columns and layers), allowing for concatenation and nesting of variables in each dimension. Also weight variables could be specified, so weighted totals could be computed (e.g. in order o correct for a nonresponse bias).

Figure 5.2 shows an example of a table specification screen of Abacus. The rows contain the variable *Town* nested in *Province*. The columns contain the two concatenated variables *Job* and *Sex*.

Figure 5.2. Specifying a table in Abacus

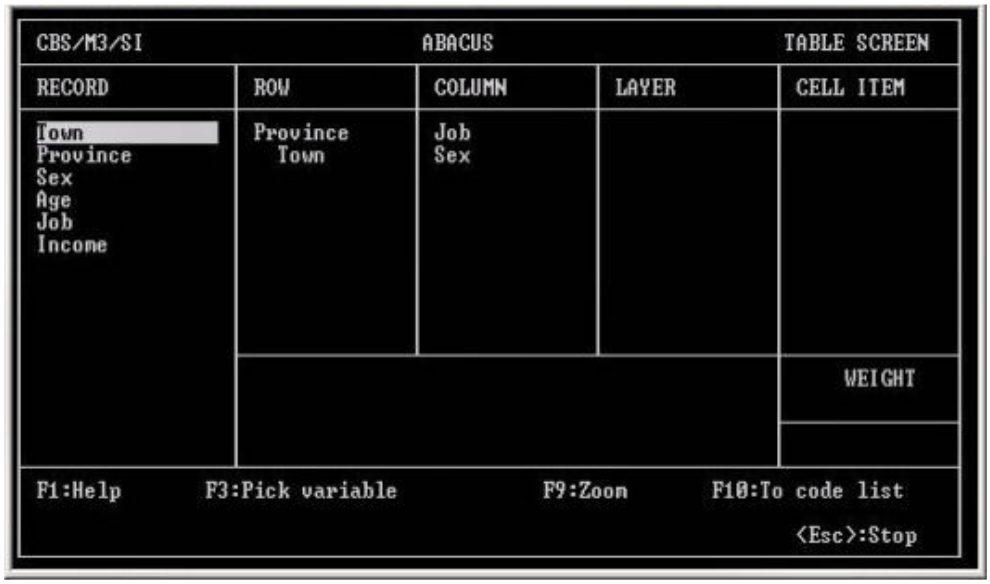
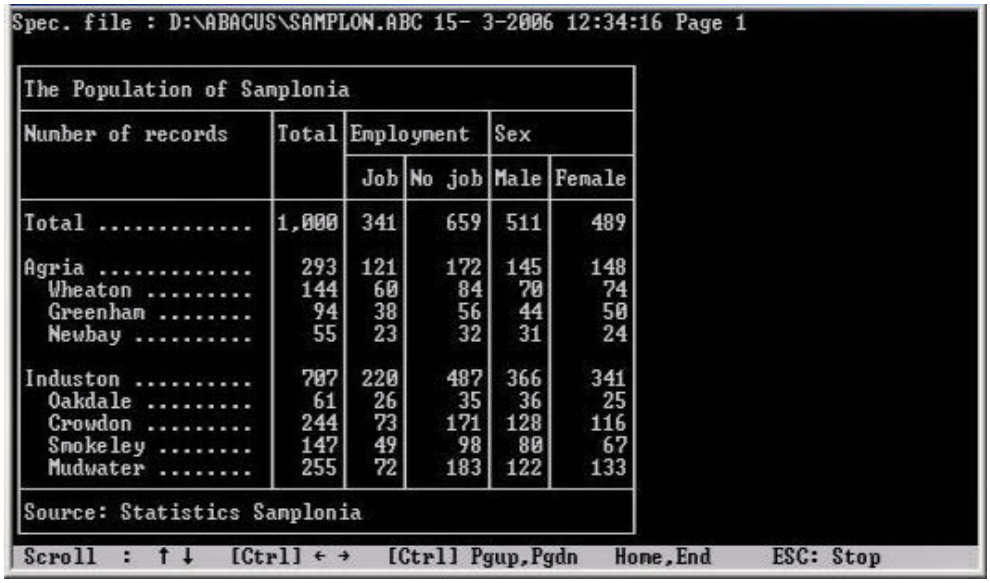


Figure 5.3 shows an example of the type of output that could be generated using Abacus.

Figure 5.3. A table generated by Abacus



Spec. file : D:\ABACUS\SAMPLON.ABC 15- 3-2006 12:34:16 Page 1

The Population of Samplonia

Number of records	Total	Employment		Sex	
		Job	No job	Male	Female
Total	1,000	341	659	511	489
Agria	293	121	172	145	148
Wheaton	144	60	84	70	74
Greenham	94	38	56	44	50
Newbay	55	23	32	31	24
Induston	707	220	487	366	341
Oakdale	61	26	35	36	25
Crowdon	244	73	171	128	116
Smokeley	147	49	98	80	67
Mudwater	255	72	183	122	133

Source: Statistics Samplonia

Scroll : ↑ ↓ [Ctrl] ← → [Ctrl] Pgup,Pgdn Home,End ESC: Stop

5.3. Bascula

To improve the quality of estimates in sample surveys, some kind of weighting adjustment procedure is often carried out. The reason is that the collected data can not be used for making inference about the population without having made corrections for unequal selection probabilities or selective non-response.

Post-stratification is a well-known correction technique. Every record is assigned some weight, and these weights are computed in such a way that the weighted sample distribution of characteristics like sex, age, marital status, and area reflects the known distribution of these characteristics in the population.

Lack of sufficient population information can make application of post-stratification difficult. Statistics Netherlands carried out research in order to improve weighting techniques. The result was a new general weighting technique called *linear weighting*. This technique computes weights from a linear model that relates the target variables of a survey to auxiliary variables.

Post-stratification is a special case of this method. Because of the generality of the method, different weighting modes can be applied that take advantage of the available population information as much as possible, and at the same time avoid the above mentioned problems.

Multiplicative weighting is another weighting technique that can compute weights for the case of insufficient population information. This technique is sometimes also called raking or iterative proportional fitting. Weights are obtained as the product of weight factors, which are computed in an iterative process.

In the late eighties, Bascula was developed as a tool implementing these weighting techniques. The first version of Bascula ran under MS-DOS, and later versions under Windows.

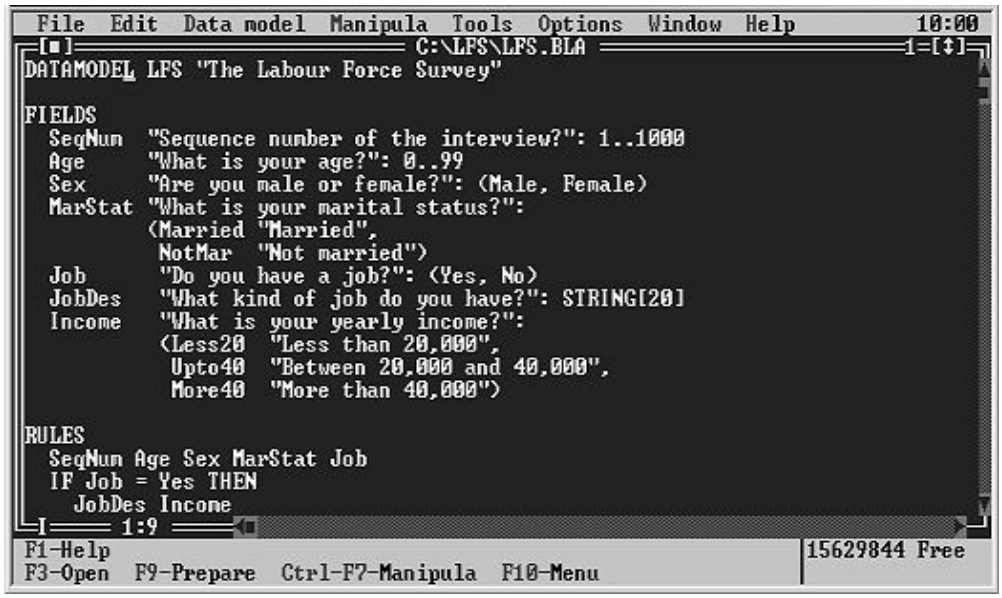
6. Blaise 3: A control centre for survey processing.

Version 2 of Blaise started a process of integrating the software needed to carry out all steps of the survey process. Many computer programs existed that took care of

one or more of these steps (data entry, data editing, tabulation, analysis, etc.), but the ideal package that does everything did not (yet) exist. This was not very surprising. Handling statistical data has so many diverse aspects that it is impossible to cover everything in one package. Moreover, since new ideas have evolved in quick succession both in statistics and informatics, they could never have been implemented at once in one system.

Having to work with all these different programs, each requiring its own data format, its own specification language (metadata format), and its own user interface, is not very efficient. Blaise 3 offered a solution to these problems. It was a system that could manage all software needed in the survey process, from questionnaire design, via data capture, to data editing and subsequent activities. It spoke the language of each program and thus was able to take care of the communication between them. Separate programs became modules in a *statistical control centre*. Using such a control centre was like being a pilot in the cockpit of an aeroplane: There are instruments to control all activities, and to monitor the situation. For more about the idea of a statistical control centre, see Bethlehem (1997).

Figure 6.1. The Blaise III Control Centre



Blaise 3 was a completely redesigned version of Blaise 2. It was finished in 1994. Object oriented programming was used to develop the system. To indicate that it was a completely new package, its name was changed into Blaise III. See figure 6.1 for a sample screen of the survey development environment if Blaise III.

Figure 6.2. A simple Blaise III data model

```
DATAMODEL LFS "The Labour Force Survey";

FIELDS
    SeqNum  "Sequence number of the interview?": 1..1000
    Age     "What is your age?": 0..99
    Sex     "Are you male or female?": (Male, Female)
    MarStat "What is your marital status?":
               (Married "Married",
                NotMar  "Not married")
    Job      "Do you have a job?": (Yes, No)
    JobDes   "What kind of job do you have?": STRING[20]
    Income   "What is your yearly income?":
               (Less20  "Less than 20,000",
                Upto40  "Between 20,000 and 40,000",
                More40  "More than 40,000")

 RULES
    SeqNum Age Sex MarStat Job
    IF Job = Yes THEN
       JobDes Income
    ENDIF

    IF Age < 15 "Respondent is younger than 15" THEN
       MarStat = NotMar "he/she is too young to be married !"
    ENDIF

 ENDMODEL
```

Blaise III was based on the idea that processing survey information is essentially a set of activities for manipulating data and meta-data. The data are the answers to the questions on the questionnaire form, and in the course of the process they are transformed into useful statistical information. This can only be achieved properly if relevant metadata is available: the information about the data. Blaise III implemented the principle that there could be no data without metadata. This principle was applied everywhere in the system.

Carrying out a survey with Blaise III started by specifying a *data model*. This was the new name for the questionnaire specification in the Blaise language. Blaise III introduced some basic changes in the language. Figure 6.2 shows the Blaise III version of the questionnaire in figure 3.1.

Some keywords have been replaced by new ones to reflect the more general nature of the metadata specification. For example, `QUESTIONNAIRE` has become `DATAMODEL`, and `QUEST` has become `FIELDS`. Blaise 1 had separate sections of route and check instructions, but in Blaise III they were combined in one `RULES` section. The semicolon was not necessary any more as a separator between commands.

The data model is a metadata specification. Blaise stored this metadata specification in machine-readable form, making it available to all parts of the system. After that, never again variables had to be specified. The metadata was always there, and it could automatically be transformed into many different formats, depending on the software that needs it.

Blaise III supported hierarchical data models. This is how many survey researchers see the world. There are households, households have members, members have health problems, etc. This is exactly what the language of the Blaise system offered. Blaise III promoted a modular approach to data model design in which hierarchies are implemented by means of nesting simple data structures.

Figure 6.3. Blaise specification of a hierarchical data model

```
DATAMODEL HouseHold "The Household Survey"

  BLOCK BPerson "Demographic data of respondent"
    FIELDS
      Name     "What is your name?": STRING[20]
      Sex      "Are you male of female?": (Male, Female)
      Married  "Are you married?": (Yes, No)
      Age      "What is your age?": 0..120
    RULES
      Name Sex Married Age
      IF (Age < 15) "If age of respondent is less than 15" THEN
        MarStat = NevMarr "you are too young to be married !"
      ENDIF
  ENDBLOCK

  LOCALS
    I: INTEGER

  FIELDS
    Address "What is the address of the household?": STRING[30]
    HHSize "Number of persons in the household?": 1..10
    Person: ARRAY[1..10] OF BPerson

  RULES
    Address HHSize
    FOR I:= 1 TO HHSize DO
      Person[I]
    ENDDO

 ENDMODEL
```

Data models could become large and complex, making it difficult to keep track of both the overall structure, and the details of several thousands of fields. Blaise offered the *block* concept to group metadata in a modular way. A block can be seen from the inside as a group of closely related fields that are independent from all other fields. From the outside a block can be seen as a unit of information that can be nested in other blocks, or repeated a number of times. Figure 6.3 contains an example of such a simple hierarchical data model.

This example contains the definition of one block: *BPerson*. The fields section at the highest level (the household level) introduces an array of 10 occurrences of this block (*Person[1]* to *Person[10]*). All these 10 fields represent a group of fields: the fields in the block BPerson (*Name*, *Sex*, *MarStat* and *Age*).

The Blaise Control Centre offered tools to handle data model specifications in a transparent and user-friendly way. It could visualise a data model by displaying it as a tree structure. Figure 6.4 shows how the data model of figure 6.3 is displayed.

Figure 6.4. A hierarchical data model

```
┬ HOUSEHOLD                          ┬ HOUSEHOLD
├── Address                          ├── Address
├── HHSize                           ├── HHSize
└+ ARRAY [1..10] OF BPERSON          └┬ ARRAY [1..10] OF BPERSON
                                      ├── PERSON
                                      ├── Name
                                      ├── Sex
                                      ├── Married
                                      └── Age
```

The data tree on the left-hand side described the data model at the highest level. It gave a good view of the general structure. There is one elementary field (*Address*) and one block field (*BPerson*). The plus sign (+) indicated a block of which the fields were not displayed. Unfolding the block, i.e. making visible all fields in the block, would result in the data tree on the right hand side with all the details.

Previous versions of Blaise could generate either a CADI, CAPI or CATI program. Each of these programs had its own characteristics and behaviour. And all programs had to be compiled using a Turbo Pascal compiler. This all changed in Blaise III. Blaise III had just one program, and it needed not be compiled. So Blaise users did not need purchase the Turbo Pascal compiler any more. This program was called the *Data Entry Program* or DEP for short. Its characteristics and behaviour could be changed in a dynamic way. There was no need any more to decide at the design stage what kind of program was required. Here are two of several aspects that could be controlled:

- *Static or Dynamic routing*. Static routing meant that route instructions were interpreted as checks. The Data Entry Program did not force any questions to be answered or skipped. In dynamic routing mode, the Data Entry Program took control of routing through the fields. Fields were processed in the order prescribed by the rules section.

- *Static or dynamic error handling.* Static error handling meant that a detected problem did not interrupt data entry or data editing activities. Errors were reported as error counts displayed adjacent to the input fields. If the Data Entry Program was in dynamic error handling mode, it would stop immediately if a problem was encountered. It was not possible to continue data entry until the problem has been solved.

By combining a routing mode and an error handling mode, different behaviour of the DEP could be invoked. Figure 6.5 shows the DEP in interviewing mode.

Figure 6.5. The DEP in interviewing mode

After data collection, data and meta-data had been brought together. Then, the set of information is ready for further analysis. What the next step would be depended on the survey at hand. It would often require some kind of manipulation of the data. Blaise III had a special tool for data manipulation. It was called *Manipula*. Manipula was a 'Swiss army knife' for data handling. It could combine or split data

files. It could create new data files by selecting variables and/or records. It could derive new variables by means of complex algorithms. It could re-arrange or sort data files, and it could convert data files from one format to another. It supported several data formats including ASCII, Oracle and Paradox. The principle that there should be no data without meta-data also applied to Manipula. In fact, an input data model and an output data model had to be specified, and Manipula took care of transforming the data in such a way that they fit the output model. Manipula required data and metadata as input, and produced data as output.

Blaise III 1.1 introduced in 1994 a new tool called Maniplus. As the name suggests the basis of this tool is Manipula, so all the functionality which was already present in Manipula was also available in Maniplus. Maniplus was described as 'a powerful tool to build applications involving Blaise'. It gave survey system designers the possibility to combine tools like the DEP, the Data Viewer, Manipula, and a DOS Shell into a user-friendly package controlled by user-defined menu systems and dialog boxes including integrated help.

It turned out that Maniplus filled an important gap in the Blaise system and probably because of that it usage quickly became widespread. Over the years major uses of Maniplus included interactive survey control systems such as a CAPI laptop management system (single-, or multi-survey) and in-office data flow control systems. Figure 6.6 show a screen of a Capi laptop management system designed in Blaise III using Maniplus.

Figure 6.6. Example of a Maniplus application screen

A closer look at traditional survey processing would make clear that metadata specifications can come in many forms. The questionnaire form is just one specification, and a setup for a statistical analysis package like SAS or SPSS is another. One design objective of Blaise III was to confront its users with only one type of metadata specification. This did not mean that he could do without other types of metadata specifications. Particularly, if software was used that was not an integral part of the Blaise system, a tool was needed to translate the metadata specification into a form suitable for processing it by the other software. Blaise III had a tool for manipulating metadata. It was called *Cameleon*. It was the successor of the Setup Generator. It supported many metadata formats. Examples are setups for the statistical packages SAS, SPSS and Stata, instructions for building databases in Paradox and Oracle, and specifications for the (built-in) tabulation package Abacus and the weighting package Bascula. For packages not supported

by Cameleon, users could easily design a metadata specification script themselves, and add it to the list.

Another new feature in Blaise III is the possibility to work with multi-lingual questionnaires. If question texts were specified in several languages, interviewers could change to a different language 'on the fly'. This turned out to be a very useful feature in countries where people speak different languages, like Finland (Finnish and Swedish), or the US (English and Spanish).

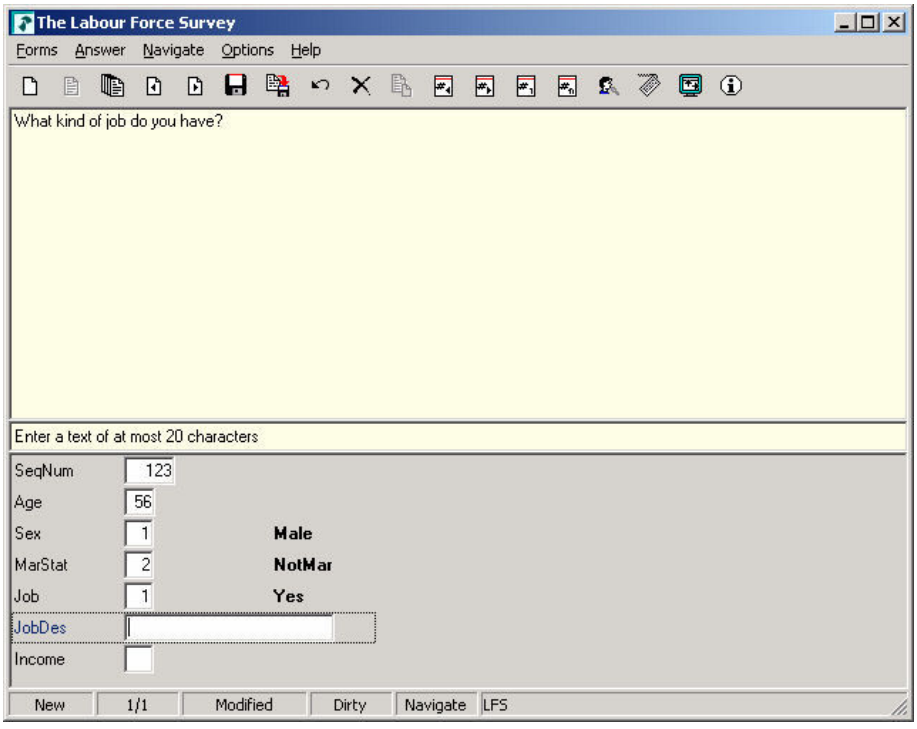
Finally, Blaise III had a tool for translating all texts used by the system into another language. The system was distributed in two languages (English and Dutch), but this option made it possible to make, for example, a French version of the Blaise system. In fact, this translation facility was a kind of version control system for all languages used. It not only helped translating texts, but also simplified the job of updating text files in case a new version of Blaise was distributed.

7. Blaise 4 Windows

Version 4 of Blaise marked the change form MS-DOS to Windows. The first version of Blaise 4 Windows was released in 1998. The functionality of Blaise 4 was basically the same as that of Blaise III. Of course, the graphical user interface offered much more possibilities for screen layout. Figure 7.1 gives an example of a screen of the Blaise 4 DEP in interviewing mode.

Since the introduction of Windows there are many more ways to communicate between applications. The Blaise developers recognised in the late 1990's that it was vital to open up the Blaise system in order to 'survive' in the ever more demanding world of official statistics. This new approach has two main advantages: it is easier to re-use parts of Blaise in other environments and it becomes easier for others to build extensions to the system.

Figure 7.1. The Blaise 4 DEP in interviewing mode.



It started with OBA, the Open Blaise Architecture. The main objective of OBA was to offer an API for all re-usable parts of the system. An *API* (Application Programming Interface) is a set of definitions of the ways one piece of computer software communicates with another. One of the primary purposes of an API is to provide a set of commonly-used functions. Programmers can then take advantage of the API by making use of its functionality, saving them the task of programming everything from scratch.

The most important API in Blaise is the 'Blaise API'. This API offers:

- Access to all metadata of a data model, like question texts, type definitions, rules definitions, etc.
- Read / write access to the data of one form (record).
- Access to the rules engine of Blaise. This makes it, for instance, possible to access all information on 'errors/edits' or to find out what is the next question on the route given a certain question in the data model.

To make something like the Blaise API possible a lot of technology in the background was needed. Blaise used Microsoft COM technology. This is a software architecture developed by Microsoft to build component-based applications.

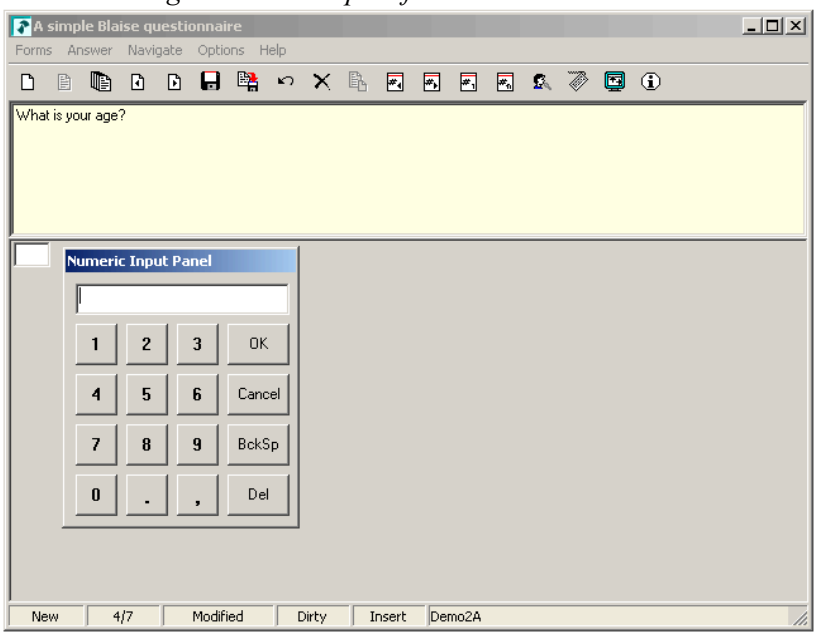
Two examples of the use of the Blaise API are:

- You can make your questionnaire documentation tool, by using the API to extract the relevant metadata from the data model.
- You can build your own Internet application based on Blaise by using the API.

One of the objectives of OBA also was to offer data storage in relational database packages. It meant the collected Blaise data could now directly be stored in many other data formats, for example, in databases systems like Oracle and SQL Server.

At many places the Blaise system offers hooks that can be used to extend its basic functionality. One way to use such hooks is to introduce alternative controls during interviewing. An example of such a control is displayed in figure 7.2. Here a small numeric input panel is displayed that can be used to supply an answer to a numerical question. This can for instance be handy when interviewing is performed using a Tablet PC.

Figure 7.2. Example of an alternative control



OBA was introduced in version 4.5 van Blaise. It was released in 2001. The new set of software tools was called the Blaise Component Pack (BCP).

Now that more and more people are connected to the Internet, web surveys are gaining in popularity amongst researchers as means of data collection. This data collection mode is called computer assisted web interviewing (CAWI) and has many advantages, which include a high response speed, the ability to interact so a respondent can get feedback about questions or possible errors, and leaving it to the respondent when to fill in the questionnaire.

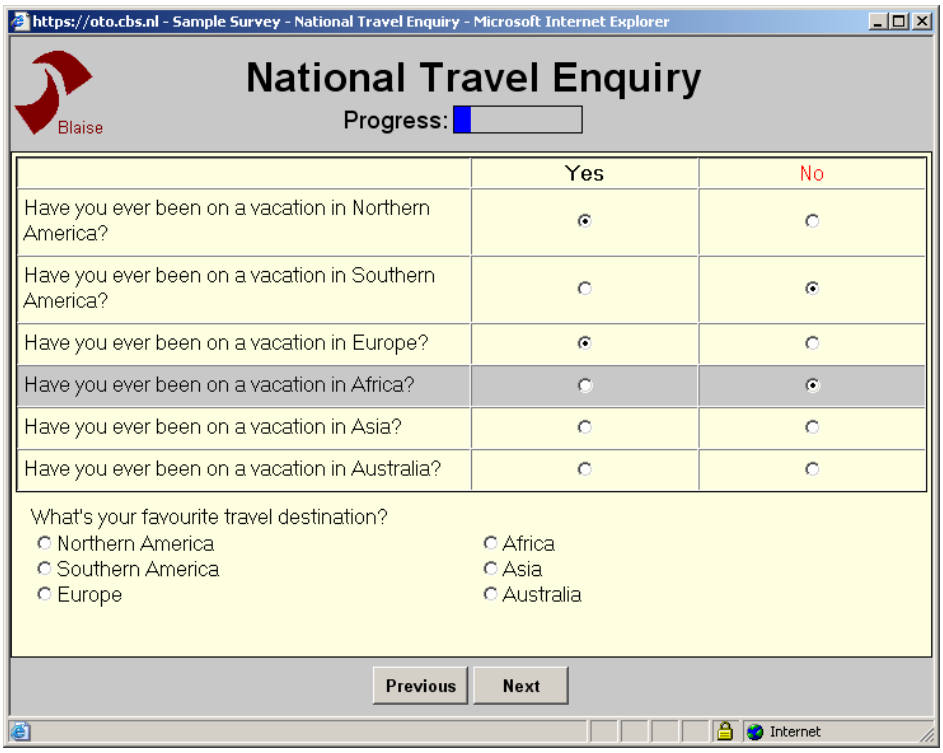
Blaise 4.6 was released in 2003. This version provided a powerful and flexible system for this data collection mode. Blaise Internet lets you choose between two possible approaches for web-based questionnaires. They are called the Interview approach and the Form approach.

The *Interview approach* is typically used for long and complex questionnaires that ask for conditional question routing and checking of questions. They are preferable if skipping or branching patterns are required in the questionnaire, or if controlling the respondent's flow through the questionnaire is important.

A survey using the Internet approach is sometimes called an 'on-line survey'. The reason is that an interview needs (and offers) continuous interaction between the respondent and the internet server. Therefore, it is unavoidable that the respondent is online during the entire interview.

Interviews are page-based, i.e. the questions are grouped into pages. Each page is sent to the respondent as a separate HTML page. After the respondent has answered the questions on the page, the answers are submitted to the internet server. The answers are checked and the applicable rules are interpreted on the server. Then, an adapted HTML page is returned to the respondent based on the answers to the previous questions. See figure 7.3. for an example.

Figure 7.3. A web questionnaire using the Interview approach



The *Form approach* is suitable for short and simple surveys with straightforward data input without question routing. Internet forms are actually paper and pen interviews turned into a version suitable for Internet. The questionnaire comprises one HTML page that can be scrolled up and down to answer the questions.

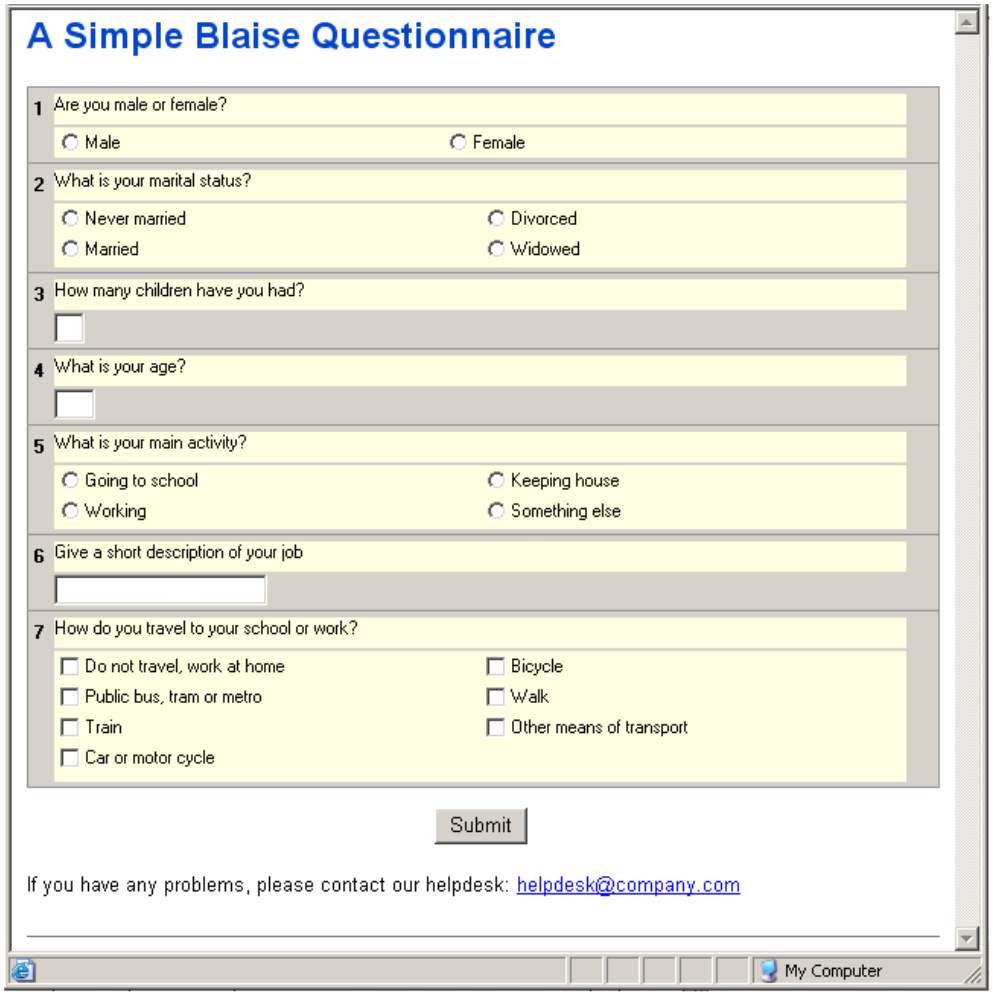
The HTML file may be delivered in any way (for example as an email attachment or as a URL on a website). All questions are presented in a fixed sequence. Respondents can browse through the form and answer questions in any order. They can fill out questions offline, meaning that there is no need for continuous contact between the respondent and the server while they answer questions. However, this doesn't mean the form can not be filled out on-line.

Because there is no contact between the respondent and the server, no routing or checking is possible during the interview. However, the client computer does execute range checks for entered answers.

When the respondent is done and submits his form to the server (after making an internet connection if none is present), the answers are sent to the Blaise Internet server. The server can execute the checking mechanism if desired and stores the data in a Blaise database. The respondent receives an answer from the server.

Figure 7.4. contains an example of a screen of a web survey using the Form approach.

Figure 7.4. A web questionnaire using the Form approach



The layout of an Internet questionnaire is very important. Since no interviewers are available to assist respondents in answering questions and navigating through the questionnaire, everything must be clear for every respondent. The standard Blaise layouts used by the Data Entry Program are not suitable for web surveys.

A web survey is set up with the *Internet Workshop*. This one of the options in the *Tool* menu of the Control Centre of Blaise 4.7. After a Blaise data model has been created, either the Interview approach or the Form approach is selected. The next step is to select an HTML page layout by selecting the proper style sheet. There are several pre-defined style sheets available.

After the survey specification has been completed, the necessary files must be installed on an Internet server. For this, the option *Internet Server Manager* in the Tools menu of the Control Centre can be used.

8. Conclusion

Over the years Blaise has enabled statisticians and others to streamline their statistical production process. Through the common metadata language, and the possibilities to transfer data and metadata from one program to the other, it became easier to process survey data.

Blaise has proved to provide a useful and efficient environment for survey processing. It is not only the standard package for Statistics Netherlands, but it is also in wide scale use at more than 120 other statistical agencies, universities, and research agencies elsewhere in the world. That Blaise is not a simple system for small scale surveys is proved by the fact the system was selected for processing the first agricultural census of the Peoples Republic of China, where approximately 3 million interviewers were to collect data on approximately 200 million farms. Unfortunately the project was stopped due to internal political struggle in China.

It is surprising to see that ideas behind Blaise have survived now for 20 years, even in this quickly changing world of processing statistical information. The first ideas emerged in an era of small, slow stand-alone computers running under MS-DOS, but these ideas still turn out to be valuable in our current Internet era. Blaise was alive and kicking for 20 years and holds a promise to remain doing so in the coming years.

9. References

Van Bastelaer, A.M.L. and Sikkel, D. (1987), "From three to three hundred hand-held computers," in *CBS select 4, Automation in Survey Processing*, Voorburg: Statistics Netherlands, pp. 13-26.

Van Bastelaer, A.M.L., Kerssemakers, F.A.M. and Sikkel, D. (1987), "A test of the Continuous Labour Force Survey with hand-held computers, interviewer behaviour and data quality," in *CBS select 4, Automation in Survey Processing*, Voorburg: Statistics Netherlands, pp. 37-54.

Bemelmans-Spork, E.J. and D. Sikkel (1985a): Observation of prices with hand-held computers, Statistical Journal of the United Nations Economic Commission for Europe, vol. 3, no.2.

Bemelmans-Spork, E.J. and D. Sikkel (1985b): Data Collection with hand-held computers, Proceedings of the 45th session, International Statistical Institute, Book III, topic 18.3.

Bethlehem, J.G. (1987), "The Data Editing Research Project of the Netherlands Central Bureau of Statistics," *Proceedings of the Third Annual Research Conference of the US Bureau of the Census*, Washington: Bureau of the Census, pp. 194-203.

Bethlehem, J.G. (1997): Integrated Control Systems for Survey Processing. In: Lyberg, L., Biemer, P., Collins, M., De Leeuw, E., Dippo, C. and Schwarz, N. (Eds), *Survey Measurement and Process Control*, Wiley, New York, pp. 371-392.

Deming, W.E. (1986), *Out of the Crisis*, Cambridge: Cambridge University Press.
Hammer, M. and Champy, J. (1993*), Reengineering the Corporation, A Manifesto for Business Revolution*. London: Nicholas Brealey Publishing.

Hammer, M. and Champy, J. (1993*), Reengineering the Corporation, A Manifesto for Business Revolution*. London: Nicholas Brealey Publishing.

Session 2.1: Standardising

In-house Support for Blaise Application Developers and Users
Janica Cajhen & Pavle Kozjek
(Statistical Office of the Republic of Slovenia, SORS)

Blaise Programming Techniques for Better Documentation
Peter Sparks & James Hagerman
(University of Michigan, USA)

Standardizing and Automating Blaise Output Test Data Delivery (TransSAS)
Latha Srinivasamohan
(U.S. Census Bureau)

Developing an Integrated Household Survey (IHS) Blaise Questionnaire from Four Major Social Surveys
Tim Burrell
(Office For National Statistics, UK)

In-house Support for Blaise Application Developers and Users

Janica Cajhen& Pavle Kozjek (Statistical Office of the Republic of Slovenia, SORS)

1. Introduction

The Blaise system is used at SORS since 1993. In this period the usage was extended from supporting single CADI and CATI surveys to a general data collection and editing tool, completely covering CATI and CAPI mode of data collection, as well as data entry and editing from paper forms. In some surveys, Blaise is also used in a process of editing of data from secondary sources or as a tool for different auxiliary processes. Manipula is successfully replacing old mainframe tools used in statistical processes.

On the other hand, some more recent Blaise methods and techniques are not yet implemented in production. In 2005, SORS began pilots on using BCP and Blaise IS. The pilots are under way and we expect to discuss and implement the results during this year.

To summarize, instead of trying to rapidly implement some new technical solutions we focused first on **setting a basic common standards on two main groups of survey applications**: CADI (including secondary sources) and CATI – CAPI. Common standards in this context mean that one survey can be added to a system, covered by generic user interface that supports all the steps in a data entry and editing process. The main ideas of this approach were presented in the SORS IBUC 2003 paper (Blaise for Post-collection Data Editing - Building general data editing system based on Blaise).

One of the key conditions to be fulfilled in order to implement the ideas of common standards was to give the necessary knowledge to developers and users of Blaise applications. Once the systems are implemented, the knowledge about using and maintaining them is inevitably. When building our own systems we can not expect the help and support from the outside.

The following article describes how the support to developers and users of Blaise applications at SORS is organized.

2. Development from the good practices

The idea on setting common processing standards on groups (or clusters) of surveys comes from the good experience with our in-house developed Gentry system for high speed data entry. In this case the target group of surveys was defined very clearly - all surveys reporting on paper forms, and using high speed data entry.

With CADI data editing it was already harder to find common standards for the group of surveys, since the activities and processes understood as data editing are already more complex and “colourful”. Common solution should cover various kinds of import into editing system, different modes of editing, various reference files and databases, different levels of users to work with the data, different ways of final data export etc. In a few words: generic GUI for data editing needs much more buttons.

The first step in preparing standard solutions for CADI surveys was to harmonize the needs of statisticians and data editing staff about the generic data editing application and belonging user interface. As soon as it was developed and implemented, the knowledge about adding new surveys into the system was spread to the larger group of application developers. At the same time, the group of application users was trained about using the new solution..

With CAPI – CATI surveys the things get more difficult and complex, since the processes are more different more from survey to survey. The system to cover and control the most important common processes is under development.

3. Help and support for developers of Blaise applications

Since Blaise was adopted at SORS as standard data editing tool, all developers were expected to be able to use it when preparing data editing applications. Most of developers from *Department for data entry and editing applications* had no previous experience with Blaise, and the **courses** were the first thing they needed. They were also not familiar with LAN environment. As the first step, the initial three-day course at a very basic level was organized. Content of the course was development of applications for the in-house standardized CADI mode of data editing. A short developer's manual was prepared, and the course was based on a practical work on examples.

With the help of the first course, the new data editing system was implemented in practice at the end of 2002. The second internal course (beginning of 2004) was mostly about different possibilities using Manipula and the third course (November 2005) was about standardized application development on LAN.

It was noticed that Blaise language was accepted very fast and successfully amongst former mainframe developers. Much bigger issue was migration to a new platform. For that reason a special **user interface for developers** was designed. Its main purpose was to enforce standard development and to bridge the gap between different environments. It supported:

- 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
- import and export data
- implementation - sending prepared application to production environment

The use of this user interface was not mandatory. It seems that with growing knowledge of developers this interface is becoming obsolete, and other modes of support are replacing it. But certainly it was a good start for beginners.

On-line manual for developers is at the moment probably main and most important support for developers. It is located on a server, accessible through the user interface, organized systematically (by contents) and regularly updated with the new contents. The manual gives a brief overview of all the steps necessary to prepare the survey application, it includes instructions, flowcharts, names and locations of standard datamodels, setups and templates etc. For the most important developing issues the translated terms and links to original Blaise Developers Guide are added.

Templates and standard solutions (data models, settings, Manipula setups) were designed to streamline the work of developers and to enforce in-house conventions

about developing applications. They are located in a read-only library on LAN server, with folders separated regarding to the contents.

The personal **help desk** for developers is not formally organised. But in practice it works well, and it is one of the fundaments of the support. Help for less experienced developers is always obtained by the colleagues from department: personally, by phone or by e-mail. Using the same approach to development makes things much less complicated and time consuming.

There are also ideas about establishing Blaise developer's **forum** on the SORS intranet site. At the moment it seems to be not necessary, since most needs are covered by above mentioned ways of support to developers.

4. Help and support for users of Blaise applications

When talking about users at SORS, we need to distinguish between two different groups:

- "Specialized" data entry and editing staff that takes care only about entry and editing, but for a large number of surveys
- Statisticians that take care about complete survey process (data entry-editing is just a part of it)

In general, user access to Blaise applications is enabled through **generic user interface**. This user interface (prepared in VB) gives as much as possible intuitive access and control to all processes of data entry and editing.

At SORS, users are involved in the process of development of applications and interfaces by checking and testing the prototypes until they are ready for the "production". Prototyping brings an important advantage: when the applications are implemented, some users already know all the functionality of the solutions, and they can immediately spread and disseminate the knowledge to their colleagues. Beside clear and intuitive interfaces, this is the second reason why the **courses** for the in-house Blaise users can be relatively short and to the point. Of course, the training for CAPI and CATI interviewers does not belong to this category and is not discussed in this paper.

Like for the group of developers, a **general manual** for users of Blaise applications was prepared. The first part explains user interface and all the steps supported in the process, and the second part gives a brief information on operational work - data entry and editing in Blaise. Explanation can be rather short, since most of the surveys in the system use standardized Blaise environment. When necessary (complex surveys, exceptions etc) the **content-based manual** can be prepared for the users. Both types of manuals are located on a LAN server (content-based in a survey working folder) and available through buttons on the generic user interface. With development of applications the manuals are updated and maintained.

When courses and manuals are not enough, the first level of personal **help desk** for users are usually their colleagues from the department, more experienced users of Blaise applications. If id doesn't work, a developer of application is contacted, and the problem is redirected to the development department.

Above mentioned methods of user support are used by both groups. There are also some specific needs of the second group – statisticians. They often ask developers for some tailor-made additions to the basic data entry-editing application, usually fast analytical Manipula setups, or special kind of data export. If it is not a part of

interface, a short instructions for the user should are added to the solution. But in general we try to include all the entry and editing proceses into a small number of generic user interfaces.

5. Conclusions

By generalizing Blaise as main data end editing tool, the need for standardization of solutions at SORS became crucial. By classifying survey processes in a few main clusters with regard to their entry and editing charactereistics we were able to set some standards in the development, as well as in the usage of applications. Cosecutively, it was possible to organize some basic modes of support to developers and users in a relatively clear, effective and controlled way. We believe this approach will be working with the new, more advanced Blaise applications as well.

6. References

De Bolster, G. Generating Blaise with Blaise, Proceedings of the 9th International Blaise Users' Conference, Ottawa, 2004

Hansen, S. E. Prpgramming Guidelines for Good Data Documentation, Proceedings of the 9th International Blaise Users' Conference, Ottawa, 2004

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

Sundgren, B., An Information Systems Architecture for National and International Statistical Organizations, CES/AC.71/1999/4

Ziesing, K. Generic System Interface Instrument, Proceedings of the 9th International Blaise Users' Conference, Ottawa, 2004

Blaise Programming Techniques for Better Documentation

Peter Sparks & James Hagerman (University of Michigan, USA)

1. Introduction

There has always been a divide between the finished software product and what really is going on internally. In the survey business, it is sometimes necessary for non-programmers to examine the actual source code. This is not always desired, either by the non-programmer or the programmer! However, a number of programming methodologies can be implemented to ease this pain and make the program more robust, easier to maintain, easier to transfer, and so on.

This paper examines different methods that can be used by the programmers (also known as “authors” or “developers”) to aid in providing clear documentation for a variety of users: other programmers (Blaise and non-Blaise), non-technical Survey Managers, and automated documentation systems (i.e., MQDS [Michigan Questionnaire Documentation System], Delta). Overly complex code typically yields poor or no documentation, reduced performance, and is not robust. Programming for good documentation improves portability and maintainability of the program code. It also provides better output from automated documentation systems and the ability to compare cross-wave instruments.

2. The Basics

First rule of thumb: Make the program usable by humans.

People are the single best method to actually understand the internals of a programmed survey. Even so, there are limits that are reached by anyone. Hence the following suggestions are given to make sure the program code is usable without the aid of automated documentation systems.

2.1. Comments

It is the bane of all programmers and desired by everyone. The best practice is "comment as you go." Write the comment as the code is being programmed and plan on being like a tourist - get the snapshot now and do not wait until another time. You will find that your own comments will help as the code is being developed. It is even more important when working with a team of programmers.

It is difficult to over-comment program code. You will know when you have reached that point because you cannot find the actual program code. This is the point in time that a separate document needs to be written.

2.1.1. Header Comments

Place a comment header at the start of each file. Include things such as the study, date, modification history, authors, copyright and use notice, and what the section is about. Keep this up-to-date! *Note:* some version control software (see 0) can automatically insert the date and time of the last modification.

```
{*********************************************************}
{                        STUDY NAME                       }
{                        Module A                         }
{                  Main Application Control File          }
{PROGRAMMER: Jim Hagerman                                 }
{CREATED:    04/21/2004                                   }
{MODIFIED:   04/05/2005                                   }
{COPYRIGHT 2005 The Regents of The University of Michigan  }
{*********************************************************}

{Production Release Information:

      Rel 1 03/04/2005 (JRH)
      Rel 2 04/05/2005 (JRH)

      Changes Implemented in 04/05/2005 version:
       03/18/2005 A21 - Deleted "(contribuciones)" and replaced "las
                        contribuciones" with "los impuestos"
}
```

2.1.2. Block/Procedure Comments

Before each block and procedure put a description of the block (input, output, purpose ...)

2.1.3. End of Structure Comments

If the end of a structure (ENDIF, ENDDO, ENDBLOCK ...) is far away from the start of the structure (IF, FOR, BLOCK ...) it is helpful to create a comment following the end structure statement duplicating the condition for the start of the structure. This is more important where a large number of structures are all ending together.

2.1.4. Rules Comments

There always will be instances of complex program code. Document these as with a plain description of what is being accomplished before using technical details. Your goal is to write a comment so that you could pick up the same program code years later and know what it's doing.

Rule of thumb: could a new programmer understand the code by reading it without comments? If "No," then comment and simplify!

2.1.5. Revision Comments

After the survey has been released and amendments have to be made, revision comments are needed. There should be a revision history maintained (usually at the start of each file) and a revision comment before each change. In the revision comment put the following:

Date, Programmer's initials, Description of the change.

This will make it useful for re-examining the history of changes when it's time for data analysis.

2.1.6. Well-named variables

Because the Blaise fieldname is used for routing in the Rules section and it is the default identifier for the data, you may not have the flexibility of using descriptive field names. This is a situation where good programming methods conflict with the intended use of a datamodel: collection and export of the data. If you are in an organization that uses the same datamodel for both purposes then flexibility in

choosing field names will be done according to their data use. For example, the actual field name could be SpBirthDate, the field tag labeled A15, and the field description would be "Spouse Birthdate." This way, the field name would be used in the rules, while the field tag displayed on the status bar, and the field description is displayed instead of the field name in the field pane and could also be used in the final dataset as a meaningful variable label.

2.2. Well-named Response Codes

When possible use meaningful descriptive response code names. For example:

```
 YesNo =     (C01         "Yes",
                           C05  (5)  "No")
```

Instead of "C01" the word "Yes" could instead be used:

```
tYesNo =     (Yes (1) "Yes",
                        No   (5) "No")
```

Hence reading the program code will be made much easier.

2.3. Multiple languages

When programming multiple languages be sure to explicitly define each language at the top of the program. This is not only the language code (i.e., Eng for English) but it is the text of the language as well.

```
LANGUAGES =
      Eng "English",
      Spa "Spanish"
```

Throughout the program, use these language tags to explicitly mark the text even though the default will be the next language. For example:

```
TdDate (D154_) ENG "(Please tell me today's date.)

                   @/@/THE DATE ^AFillMonth/^AFillDay/^AFillYear
                   @/THE DAY IS ^AFillDayName

                   @/@/DAY OF WEEK:"

               SPA "(Por favor, dígame la fecha de hoy.)

                   @/@/LA FECHA ES ^AFillMonth/^AFillDay/^AfillYear
                   @/EL DIA ES ^AFillDayName

                   @/@/DIA DE LA SEMANA:" /
                   ENG "TODAY'S DAY OF WEEK"
                   SPA "LA FECHA DE HOY" :

                   (DAYOK     (1)   ENG "Day ok"
                                    SPA "Día ok",
                    DAYNOTOK  (5)   ENG "Day not ok"
                                    SPA "Día no ok")
```

2.4. Standard case conventions

As suggested by the Blaise Reference Manual all keywords should appear in caps, i.e. DATAMODEL, IF, THEN, DO, TABLE, BLOCK, FIELDS, RULES, … Do not use mixed/lower case for keywords whether they are bolded or not in the program editor.

If you can use descriptive names, use "Camel" case, avoid confusing letters and numbers look-alikes (the lowercase letter “l” and the number “1,” letter “O” and number zero “0,” the number “2” and the letter “Z”). Hence a name will look like SpouseBirthDate where each word starts with a capital letter followed by lowercase letters and no underscores.

Tip: The F12 key in the Blaise editor will change the text under the cursor into uppercase and a second press will change it to lowercase.

Tip: The list of keywords is in the C:\Program Files\StatNeth\Blaise 4.7 Enterprise\Bin\Blaise.SHT file that determines which keywords are highlighted in the editor. If there's a new keyword that you want highlighted add it to this list.

2.5. Consistent indenting

Spacing is very important for readability and maintainability of the program. It is relatively easy to spot different programmers in code that does not follow rules for indents, variable naming, layout, and so forth.

2.5.1. Matching Control Structures

Make sure that all your structures that have starting & ending parts are matched-aligned. These include:

IF/ELSEIF/ELSE/ENDIF
BLOCK/ENDBLOCK
TABLE/ENDTABLE
DATAMODEL/ENDMODEL
LIBRARY/ENDLIBRARY
FOR/ENDDO
PROCEDURE/ENDPROCEDURE

Some structures are division markers in the program and do not have matching end members:

FIELDS
RULES
PARAMETERS
TYPE
CHECK
SIGNAL
LAYOUT

For all structures that are inside other structures (i.e., embedded IF statements), indent the next level so that each structure is easy to match the beginning and ending points.

```
PD16_HadEpisode
IF PD16_HadEpisode = Yes THEN

     PD16a_HowManyEpisodes
     SIGNAL
          PD16a_HowManyEpisodes < 90
          Eng "Iwer: please verify number of episodes"

     PD16b_TypesEpisodes
     IF PD16b_TypesEpisodes IN [Mild..Severe] THEN
          PD16b_ExampleEpisode
     ENDIF

ELSE

     PD17_OtherExperience

ENDIF
```

2.6. Well-defined Storage Structure

Place your source code in the same directory structure from one project to another. Place your datamodel release and all associated files, likewise, in another structure. Name all your files consistently so they can be easily found. For example, the main source file with the uppermost Rules section could be named with an extension .BLA, while all additional sections which are included could be named .INC. A quick search (*.bla) will then locate the main file no matter what it is named.

Use the project manager to your advantage (and everyone else's). Absolutely use a project file and maintain the list of all files that belong with the project. Be sure to include your Modelib (.BML), configuration file(s), menu file, and anything else that is required for deploying your project.

Both of these standards will be aids for archiving and recovery.

2.7. Program Defined Types

Use precompiled type libraries by placing all common types into one that will be used throughout your project. Make use of the type file via the LIBRARY command. Since the .lib is compiled it then helps make sure common types are used within your project and across projects in your organization. When archiving a project it is important to put a copy of the source type with the code.

2.7.1. Type Use

Types should be used when feasible to ensure consistency throughout the program. Although many things can be assumed for working with questions it is better to document more than less. For example,

```
MyYesNo    "Answer Yes or No": (Yes, No)
```

Would be better programmed as:

```
TYPE
  tYesNo = (Yes         (1) "Yes",
            No          (5) "No")

FIELDS
  MyYesNo   (A1)  "Answer Yes or No" /
                  "Sample Yes/No Question": tYesNo
```

The field tag and field description have been added, a reusable type has been added, and the answer categories are defined with explicit values instead of i mplied. The defined type also allows the use of various input masks as well as making all tYesNo questions comparable across the survey.

2.8.2. Blocks and Tables as Types

Both Blocks and Tables can be considered complex types consisting of a small datamodel within their structure. Therefore, they should also be preceded by a type designation. For Blocks, the standard has been to use a preceding "B" and for Tables it is with a "T."

```
BLOCK BPerson
           ...
ENDBLOCK {BPerson}

TABLE TPersonGrid
      FIELDS
            Person : ARRAY[1..10] OF BPerson
           ...
ENDTABLE {TPersonGrid}

FIELDS
      PersonGrid : TPersonGrid
```

2.8.3. Input Masks & Types

There is an added benefit to using types: input masks. Since each type can have a custom mask, there is a benefit of having a few types used consistently throughout the program. For those questions that have unique response codes or ranges it is actually better to put that actual type on the field itself rather than creating a unique type. There is less overhead in maintenance, compiling, and so forth if there are only types that are needed stored in the datamodel properties file (.bxi).

2.9. Create Formatting Standards

These are highly attuned to an organization and its own standards, so all fonts and styles, placement, and so forth are not part of this paper. Make use of the customization possible but endeavor to be consistent.

Custom fonts

Blaise allows multiple attributes to be assigned to each custom font letter. In general it is easier to "read" the question text if the formatting tags are not embedded (e.g., @A@BSome interviewing instruction here.@B@A) but are instead defined with a defined composite meaning per tag; for example, interviewing instructions in Arial 10 blue and bold could be set with one attribute (e.g., @I; *I = Interviewer*). Close each section of formatted text with its matching formatting tag (e.g., @ISome interviewing instruction here.@I).

A testing feature useful for checking fills is to define a font tag (e.g., @T; *T = Testing*) that turns on the red color. These tags are used to surround fills in the question text and make the fills easier to spot during testing/debugging. When the testing is done the @T is set back to default to hide the red highlight.

If possible put the formatting tags in the question text rather than embedding them within fills. This will make it easier to spot and also reduce the risk of having unbalanced enhancements because a fill was active one time and not another.

2.9.1. Info & Field Panes

Create standard named layouts in the Modelib, and use standard font styles as well. Thus, if programmers or project managers switch between projects they understand what is meant by Size18Col1Height0 Infopane and what characteristics the @B formatting tag will have. Make use of these named layouts in the LAYOUT section of the program code.

Avoid scrolling in the info pane because it is known that text not shown beyond the current pane is likely to be missed when being read.

2.9.2. LAYOUT instructions

Layout instructions embedded at the end of blocks and datamodels tend not to be very descriptive as to what the final presentation will look like to the interviewer (or field representative). All that information is buried in the Modelib file.

In order to see what the interviewer sees, the Modelib and DEP configuration files must be read and interpreted – a daunting task since it would essentially duplicate the work of the DEP using the layout, font, grid, and other formatting instructions.

If the purpose of the documentation is simply to retrieve the question text and some basic formatting (bold, underline, italics, colors, font and size) then the layout instructions can be ignored. If the effort is to truly duplicate the DEP screens, then much more work is necessary.

2.10. Program Structure

2.10.1. Blocks

In general, each section should be programmed as a block by itself, and, according to Blaise reference manual, the more a program is partitioned into blocks, the better. The smaller units have several advantages; faster execution speed, easier to maintain, better data manipulation, and so forth. The trade-off is that the more a program is subdivided then the more difficult it is to read the source code.

Remember that blocks (and tables) are essentially complex types. In terms of the Meta information the reference to a block is the same whether one block definition is embedded within another block or that they are all at the same level. If the goal of the documentation is to reconstruct the exact block structure of a datamodel then more information than just the name will be needed.

For example, both block structures below yield exactly the same field name with block notation: A.B.C.SomeField.

```
BLOCK BA
      BLOCK BB
            BLOCK BC
                  FIELDS
                        SomeField : TYesNo
            ENDBLOCK

            FIELDS
                  C : BC
      ENDBLOCK

FIELDS
      B : BB
ENDBLOCK

FIELDS
      A : BA
```

or

```
BLOCK BC
      FIELDS
            SomeField : TYesNo
ENDBLOCK

BLOCK BB
      FIELDS
            C : BC
ENDBLOCK

BLOCK BA
      FIELDS
            B : BB
ENDBLOCK

FIELDS
      A : BA
```

Block Parameters

For program efficiency and lower overhead pass parameters into and out of blocks. Do not reference the block itself when using variables. Remember that generated parameters have the highest overhead, and large numbers of generated parameters will easily bog the execution speed of an interview.

Explicitly state each parameter. By default Blaise treats them as IMPORT and it will be confusing to have the following:

```
BLOCK BA
      PARAMETERS
            IMPORT    piA : INTEGER
                      piB : INTEGER
            EXPORT    peA : INTEGER
                      peB : INTEGER
```

In this example the peB is really an IMPORT parameter because the EXPORT command only is for the field (list) that immediately follows. That is, IMPORT, EXPORT, and TRANSIT are not status change flags like CHECK or SIGNAL. So attempting to assign a value to peB will result in an error by the Blaise compiler.

The form that will work for EXPORT on multiple items is:

```
EXPORT      peA,
            peB : INTEGER
```

Pass parameters whenever possible. Otherwise the variables references that require a parameter-like treatment will be "generated parameters" which are the same as TRANSIT at a high overhead.

2.11. Parallel Blocks

Parallel blocks present an interesting problem for documentation. They can be a view of the same fields and rules that will be eventually collected but presented on demand, or they can be a review of information, or even collecting information that's outside the normal survey (i.e., language switching and how often it occurs).

The names of the parallel blocks are defined at the top of the datamodel. For automated documentation systems these blocks could be presented as unique views.

3. Related assignments and checks closely grouped

When programming edit checks in the rules of the instrument it is good practice to place the checks as close to the source questions as possible. This has the benefit of triggering potential checks quickly rather than searching through all the rules and the person reading the code will also be able to find the relevant questions quickly.

```
A1_Day
IF A1_Month = September OR A1_Month = April OR
   A1_Month = June OR A1_Month = November THEN

      CHECK
           A1_Day < 31 OR A1_Day = NONRESPONSE
           "@D@Rw@R@D @B@RApril, June, September, and
           Novemberonly have 30 days@/@/@|···Please verify
           the day of birth and re-enter@R@B"

ELSEIF A1_Month = February THEN

      CHECK
           A1_Day < 30 OR A1_Day = NONRESPONSE
          "@D@Rw@R@D @B@RFebruary only has 28 or 29 days
          @/@/@|···Please verify the date of birth and
          re-enter@R@B"
ENDIF

A1_Year
IF A1_Year <> NONRESPONSE THEN
      IF A1_Month = February AND (A1_Year MOD 4 <> 0) THEN

            CHECK
                 A1_Day < 29 OR A1_Day = NONRESPONSE
                 INVOLVING (A1_Day)
                 "@D@Rw@R@D @B@RFebruary has only 28 days
                 for the given year@/@/···Please verify the
                 date of birth and re-enter@R@B"
      ENDIF
ENDIF
```

4. Tips

4.1. String vs. Open

Use string types whenever you want the data stored in your dataset and there is a fixed upper size to the information. Use open types whenever the data could be large and probably should be recoded at a later date, and that the data itself is not going to be exported to a data set. You can freely copy information back & forth between these forms but realize the default string size is currently 255 characters (up to 32767 if specified) and the default open size is 32767 characters variable capacity.

4.2. Passing Parameters

If you have an array of information that needs to be passed to a sub block then you should arrange your programming to pass one member of that array to the sub block at a time (similar to working with tables and grids).

4.3. When to use Procedures

This item is certainly open to debate, but a good rule of thumb is to use a procedure when a repeated result (twice or more) is desired that does not involve the interaction of the interviewer. For example, string comparisons, conversion of dates into a formatted date string, complex sorts and searches, complex counts, and the like. It is wise to avoid collecting data within procedures for the following reason:

- The audit trails will contain the keystrokes for the values gathered in a procedure, but the data in a procedure will be lost the same way auxfields are lost, and any remarks gathered in a procedure will be lost in the same way.
- Upon recovery of a suspended case, the DEP will stop at the procedures to re-gather the data and in order to avoid re-asking the procedure, questions flags have to be set, a decision made to ask procedural questions only once or to store the data elsewhere (in which case, why not collect it in a normal block – very messy).

Procedures in Blaise do make logical sense and should be used for routine tasks that will otherwise lengthen code and make maintainability difficult.

In terms of documentation, procedures should be commented as for blocks (inputs, outputs, expected results). They are like blocks in an instrument and so should be placed at the nearest level where they are used or in a separate procedure file (*.prc). Please remember that like blocks, the more levels that are embedded, the more difficult it becomes to decipher the instrument. That is, procedure A calls procedure B calls procedure C calls ... Done correctly, it can be a concise method of performing very complex operations. Done incorrectly it can become a nightmare to read and troubleshoot, decrease performance, and make it difficult to document code.

In short, keep procedures short and to the point and avoid making calls from one to the next (go "wide" instead of "deep").

4.4. Paging in Tables

Break Table pages so no scrolling occurs during entry

```
      LAYOUT
            BEFORE C2a_Names[1]   NEWPAGE
            BEFORE C2a_Names[6]   NEWPAGE
            BEFORE C2a_Names[11]  NEWPAGE
            BEFORE C2a_Names[16]  NEWPAGE
ENDTABLE {TC2a}
```

5. Simplified code approaches

Below, are three examples of writing an IF statement and all reach the same outcome. The point here is that two of these examples are probably better suited to more efficient programming style whereas the remaining example is best suited for documentation purposes. Thus, the question becomes, "Where is the middle ground on this issue?"

Example 1: This could be typical code that a programmer would write. All it tells you is to select people who are not dead and no DKs or RFs. People who read this code are generally more interested who is actually passing this IF statement as opposed to those who are not.

Example 1

```
IF Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu <> DIED AND
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = RESPONSE THEN
```

Example 2: Another example of code a programmer would write but with the inclusion of who passes this IF statement instead of who doesn't. Anyone who reads this code would need to understand the use of the "IN" operator in the IF statement.

Example 2

```
IF Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu IN [FU, HUCS, HU,
   FUMI, M_O, Jail, Mil, Educ, Health, Other] THEN
```

Example 3: This is the type of code would be most easily read by a non-programmer. It is not the most efficient method programming (it works though), but it does provide valuable information regarding the IF statement and is very useful in automatic documentation systems.

Example 3

```
IF Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = FU     OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = HUCS   OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = HU     OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = FUMI   OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = M_O    OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = Jail   OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = Mil    OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = Educ   OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = Health OR
   Coverscreen.CYPersInfo.CYPInfo[I].CYFuHu = Other THEN
```

6. Datamodel Properties

The datamodel properties are ways to enhance the basic functioning of the Blaise program while running in the DEP. These are: masks to the types defined in the datamodel, both general and custom; and additional information set aside for status bars, names and methods for parallel blocks, languages, alien routers, and so forth.

In complex documentation systems, it is possible to extract the datamodel properties such as masks to give an even more accurate representation of the datamodel as it would appear in the DEP. These are important since there is no other place in the source code that gives this indication. However, the masks tend to be interactive (inserting the thousands marker and the currency symbol at the correct places) and not really "seen" until data is entered.

Storing this information from automated systems makes sense because different users of the information may or may not want to see this supplemental formatting.

6.1. Global DK/RF

The program should always contain a global DK/RF for every question. This is accomplished by putting the following code in:

```
DATAMODEL FirstNew "First questionaire"

      ATTRIBUTES = DK, RF
```

6.2. Discontinuous Ranges

If a question requires multiple ranges (i.e., ages 1..120, 995..997), then the range should be put in full (1..997) and a hard check be put in to exclude invalid values.

6.3. Calculations/Constants

All global constants should be declared at the very start of the program to make changes easy to do. Modular (“include” files) constants should be at the very start of module.

For example, get rid of the constant (1980, 1, 1)

```
IF (BirthDate > (1980,1,1)) THEN
      CHECK
           Married = No "^Name is too young to be married"
ENDIF
```

And instead declare it early in the program by using a calculation:

```
cAge18YearsAgo : 0..120
           ...
cAge18YearsAgo := STARTDATE - (YEAR(STARTDATE)-18,
                  MONTH(STARTDATE), DAY(STARTDATE))
```

6.4. Locals

Variables such as reused loop indexes and calculations should be stored as locals. A rule of thumb to consider is whether this temporary information is of value to the project staff. If so, then ask the project staff to verify.

```
LOCALS
      LLoopIndex : INTEGER
```

Calculated ages could be stored as either AUXFIELD or stored in the data (using the KEEP method). Generally, any calculations based upon a current date are evaluated one time at that date, and then are not re-evaluated. If the calculation was always redone every time the form was entered, then there is a potential for wiping out data, or causing questions with missing data to appear. So locals may be used for calculations based on dates, but the source of those calculations should not vary.

7. External Databases

Blaise provides powerful methods of retrieving extra information that is easily retrieved from within the interview. This external data can be used for fills, look-up lists, classifications, searches, and the like. As such, the data within an external database can change through the life of a survey without any change to the datamodel.

There is little that a programmer can do to help in this documentation other than provide reasonable comments at the USES statement (describe what the database is about, primary keys, fields accessed, etc...). More than one EXTERNAL statement can be used to retrieve from the same database. There doesn't appear to be a performance issue from using multiple EXTERNAL statements within a program to the same database though in theory it might have an effect.

8. Other Useful Tidbits

8.1. Project Files

Project files are needed to not only provide a workspace for the development of a datamodel, but they keep the Modelib, run parameters, datamodel properties and other important details together. Without a project file the programmers run the risk of forgetting something when the datamodel is prepared.

8.2. Source Control

It should be an absolute necessity that version control of the source code as well as releases of a datamodel be maintained (i.e., SourceSafe, QWin...). This is not only a safeguard for the normal development of the source code (ability to recover from mistakes, compare prior versions) but it makes it much easier to develop code as part of a team. When it comes time for releases, the datamodel can be recreated and/or archived with the matching code. Although not frequently required, having past versions available to answer questions is important.

9. Electronic documentation methods for better documentation:

9.1. How do you improve your documentation?

Add more to the code rather than less. It's impossible for a program to add more detail that isn't there to begin with. This means adding:

- description element for blocks
- question tags
- language markers per text
- well-defined code values (don't assume a value) and code texts

```
BLOCK BSection_A  "Social Relations II Core Network - Section A"
```

9.2. How do you improve the management and efficiency of your code?

Try to keep logic and coding in small units that are simple and build the program from those units. In other words "block" your code.

9.3. How do you simplify the description of universes and the generation of logic for automatic documentation systems?

Use the IF/ELSEIF construction when possible instead of repeated IF constructions

The IF/ELSEIF structure will check only as many conditions as necessary to find a true statement before skipping the following statements. This generally is greater program efficiency since on average most logic checks will never reach the final condition.

However, when the statement is broken into separated IF parts then each of the IF conditions have to be checked thereby resulting in a slower execution because each has to be checked each time. But, sometimes this is necessary if, as in the example below, the asking of fields (B1, B2, and B3) is not mutually exclusive.

```
A1
IF A1 = Yes THEN
      B1
ELSEIF A1 = No THEN
      B2
ELSE
      B3
ENDIF
```

versus:

```
A1
IF A1 = Yes THEN
      B1
ENDIF

IF A1 = No THEN
      B2
ENDIF

IF A1 = NONRESPONSE OR A1 = EMPTY THEN
      B3
ENDIF
```

9.4. How do you improve the use of text enhancements and avoid redundancies?

Avoid embedding text enhancements that do the same job; for example:

let @B be defined as toggling Bold on

let @S be defined as Arial 11 Bold on Blue color

Then some question text such as:

```
"@BA11. @SSome interviewer instruction@S and @B no more bold here."
```

Results in the DEP as:

A11. Some interviewer instruction **and** no more bold here.

The inner text formatting of @S overrides the prior @B text formatting. The confusion may arise from the bold being turned on a second time and yet not turned off from the second @S since the @B is still in effect. With slightly more text the same result is given but it is clear that the first and third parts of the sentence are bold while the middle part is an interviewer instruction.

9.5. How do you program readable question text while using text enhancements and fills?

Avoid embedding text enhancements inside fills. Let @B be defined as toggling Bold on.

If your fill assignment is:

```
IF ChildGender = Male THEN
      xFillHeSheUndertake := 'he @Bundertake@B a job'
ELSE
      xFillHeSheUndertake := 'she @Bundertake@B a different job'
ENDIF
```

Then the question text might read "When did ^xFillHeSheUndertake?"

Whereas if the assignments were:

```
IF ChildGender = Male THEN
      xFillHeShe       := 'he'
      xFillDifferentJob:= 'job'
ELSE
      xFillHeShe       := 'she'
      xFillDifferentJob:= 'different job'
ENDIF
```

Then the question text might read "When did ^xFillHeShe @Bundertake@B a ^xFillDifferentJob?"

Ultimately it is easier to read text that has fewer small piece changes than it is to derive the text from one fill.

9.6. How do you document explicitly in your logic without formal comments?

Adding text to logic conditions:

```
IF A1 = C05 "Live at home = No" THEN
```

This is another method for documenting program logic without explicitly commenting in the code. Also, this extra information can be pulled out, via the Blaise Component Pack, and processed by documentation systems in addition to any "normal" information. This feature is undocumented but is still supported in Blaise 4.7.

9.7. How do you define descriptive fills for multiple languages that make it easier for automatic documentation systems to interpret?

If memory is not an issue then create a unique fill per situation instead of reusing the same fill with different contents (e.g., two fills for [he/she] and [is/was]). Create unique fills per language (i.e., FillHeShe_Eng and FillHeShe_Sp).

Use descriptive names that describe the fill itself, such as:

```
AUXFIELDS

      xyou_youanyfamliv: STRING[50]
      {you/you or anyone else in your family living there/
       they/they or anyone else in their family living there}

      xyou_youanyfamliv_sp : STRING[50]
      {Es Ud. dueñ(o/a)/Son Uds. dueños}

      xyou_yourfamliv  : STRING[40]
      {you/you and your family living there/they/
       they and their family living there}

      xyou_yourfamliv_sp : STRING[40]
      {Ud./su familia}
```

9.8. What type of difficulties do arrayed fills cause for automatic documentation systems?

Arrays of fills make documentation difficult, but sometimes they are very useful in the code. These types of fills are good for creating restricted enumerated lists, displaying household members, and creating lists of selected items.

This means any array element could hold a fill for any assignment to the array. For example, John Doe could appear in any of one of ten places for Sibling Name in the example below. Fills that are arrayed are always difficult to document and ideally each element of the array should be documented separately.

```
Sibling_Name            : ARRAY[1..10] OF STRING[50]
```

```
BC17  (BC17)
      "You mentioned ^BCMostRecCount jobs that ended at the same
      time.··Which one of these jobs ^xdoyou_doeshe_doesshe
      consider to have been ^xyour_his_her main job?"

      "Mencionó empleos que terminaron al mismo tiempo.··¿Cuál de
      estos empleos considera ^xUd_el_ella haber sido su trabajo
      principal?" /
      "Most Recent Main Job" :

      TMRJobList, NODK, NORF


TYPE
      TMRJobList =      (Job1       "^xMRJob[1]",
                         Job2       "^xMRJob[2]",
                         Job3       "^xMRJob[3]",
                         Job4       "^xMRJob[4]",
                         Job5       "^xMRJob[5]",
                         Job6       "^xMRJob[6]",
                         Job7       "^xMRJob[7]",
                         Job8       "^xMRJob[8]",
                         Job9       "^xMRJob[9]",
                         Job10      "^xMRJob[10]")
```

9.9. How do Parameters affect automatic documentation systems?

Although the use of parameters in blocks and procedures may be confusing and more difficult to work with than just a straight reference, the programmer should use clearly defined parameters whenever possible. This avoids generated parameters which carry a much higher overhead than passed parameters; passed parameters clearly document all the information a block/procedure needs.

Although they improve execution efficiency greatly, parameters being passed back and forth in an application can cause a great deal of pain for automatic documentation systems due to a significant increase in the complexity of the algorithms. This is primarily due to the automatic documentation systems' need to match up the internal parameter names with the original call to the block or procedure.

This affects not only program logic but the use of fills too; due to data being passed in and out to construct various fills.

9.10. How does implicit definition of field widths affect the programming?

Avoid using straight basic types (i.e., INTEGER, REAL, STRING) for fields and auxfields. These are stored by default values and generally give grief to the data analysts because the ranges are not defined. They also contribute to "loose" programming by allowing values out of an implied range. For example LCount below should have values 0 through a maximum of 5.

However the actual value for LCount is -99,999,999,999,999,999 to 999,999,999,999,999,999 which gives plenty of leeway to make mistakes. This is a much greater concern if LCount is later used as an index (which means Blaise will stop preparation due to an invalid range) or was stored as a field for later data analysis.

```
LOCAL
     LCount : INTEGER
RULES
     LCount := 0
     IF A1 = Yes THEN
           LCount := LCount + 1
     ENDIF
     ...
     IF A5 = Yes THEN
           LCount := LCount + 1
     ENDIF
```

9.11. How do you handle using one instrument for either multiple languages or modes?

Use a flag indicating the mode or language of the interview throughout the instrument. This mode variable can be established at the start of the interview so that appropriate logic and fills can be assigned (e.g., between CAWI and CATI interviews). Such a variable could be filled from the CAPI/CATI/CADI/CAWI functions available in the RULES section along with other variables.

The ACTIVELANGUAGE command provides a handy way to retrieve the current interviewing language that can be used in conjunction with a mode variable. If you keep all fills separate by language by naming them uniquely then you will not need extra programming to make assignments in the RULES section.

In this example the same fill is used for both English and Spanish, and so the ACTIVELANGUAGE command needs to be used in the construction of the fill.

```
IF ACTIVELANGUAGE = SPAN THEN
      IF ChildGender = Male THEN
            xFillHeShe := 'el'
      ELSE
            xFillHeShe := 'ella'
      ENDIF
ELSE
      IF ChildGender = Male THEN
            xFillHeShe := 'he'
      ELSE
           xFillHeShe := 'she'
      ENDIF
ENDIF
```

However, if separate fills were created per language then this could be programmed more simply as

```
IF ChildGender = Male THEN
            xFillElElla := 'él'
            xFillHeShe := 'he'
ELSE
            xFillElElla := 'ella'
            xFillHeShe := 'she'
ENDIF
```

This has the advantage of requiring less work during the rules scan after each field is entered. It is also easier to maintain, reduces the numbers of lines of code, and keeps fills confined to their appropriate language.

Use the ACTIVELANGUAGE command if there are appropriate language-specific questions:

```
IF ACTIVELANGUAGE = SPA THEN
      YearEnterCountry
      EntryConcerns
ENDIF
```

10. Conclusion

Blaise is a powerful programming language that gives the programmer a great deal of versatility. It also then becomes the programmer and the organization's responsibility to establish and follow programming standards. These allow for better resource management between programmers and projects for maintainable code; as well as open the door for automated documentation systems to use the consistency of the code. Standards in programming also lead to other individuals making use of resources that may otherwise be cryptic.

11. References

Altvater, D., Stanford, V., Ziesing, C. (2001). Programming Techniques for Complex Surveys in Blaise. Paper presented at the 7th International Blaise Users Conference, Washington, D.C., September.

Blaise OnLine Assistant (release 4.7.1000). (2005). "Good Programming Practices," and "Examples, Tips, Tricks." Retrieved February 20, 2006 from World Wide Web: http://www.blaise.com/onlinehelp/

Hansen, S.E. (2004). Programming Guidelines for Good Data Documentation. Paper presented at the 9th International Blaise Users Conference, Gatineau, Québec, September.

Kelly, M. (2000). What users want from a tool for analysing and documenting electronic questionnaires: the user requirements for the TADEQ project. Paper presented at the 6th International Blaise Users Conference, Kinsale, Ireland, May.

Pierzchala, M. & Farrant, G. (2000). Helping non-programmers to specify a Blaise Questionnaire. Paper presented at the 6th International Blaise Users Conference, Kinsale, Ireland, May.

Standardizing and Automating Blaise Output Test Data Delivery (TransSAS)

Latha Srinivasamohan (U.S. Census Bureau)

1. Background

This paper will discuss an automated Blaise output delivery tool known as TransSAS. This tool runs on a multi-user instrument testing environment developed and supported by the Technologies Management Office (TMO) of the U.S. Census Bureau.

2. Overview

As Blaise instruments at the Census Bureau became complex, the need for a tool that facilitated testing of the instrument data as well as the instrument itself became apparent. There are several tools to help in the software development process in Blaise, but we realized that a utility tool for testing the data that is output from Blaise would be useful. For the CASES instruments we provided data output from the testing environment using an application developed by the Demographic Surveys Division (DSD) data processors, called TransCASES. The TransSAS utility has been developed recently to provide similar output from our Blaise instruments. It is a tool, which allows testers and data processors a way to retrieve and review SAS output from a Blaise instrument from the TMO testing environment.

This paper will briefly describe the testing environment (known as TMOusers Menu), discuss the importance of data verification by sponsors and data processors during the early stages of instrument development, and point out some of the issues we have encountered while running TransSAS in a multi-user environment.

3. Overview of the Multi-User Testing Environment at the Census Bureau

The TMOusers Menu is a WinBatch application, which has a survey-specific Manipula interface to access the Blaise instruments. All current surveys are listed here, and access is given to appropriate sponsors/testers for testing of their surveys. Developers have read/write access to the shared drive on which the TMOusers Menu runs to allow for submitting and testing of the instruments. The figures below show the three screens that come up before arriving at the Maniplus interface.

Figure 3.1

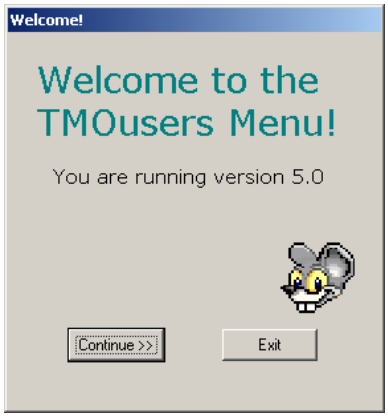


Figure 3.2

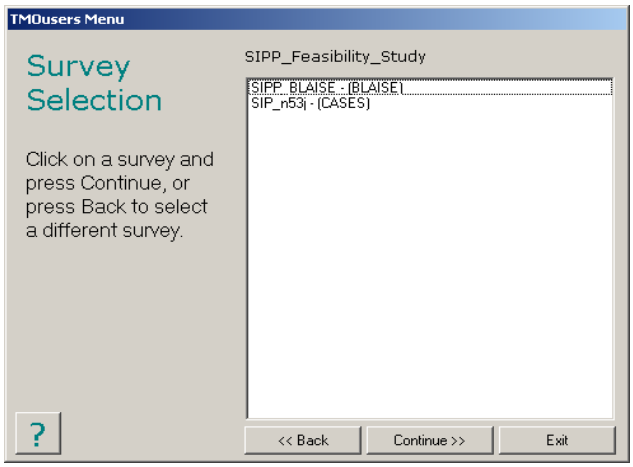
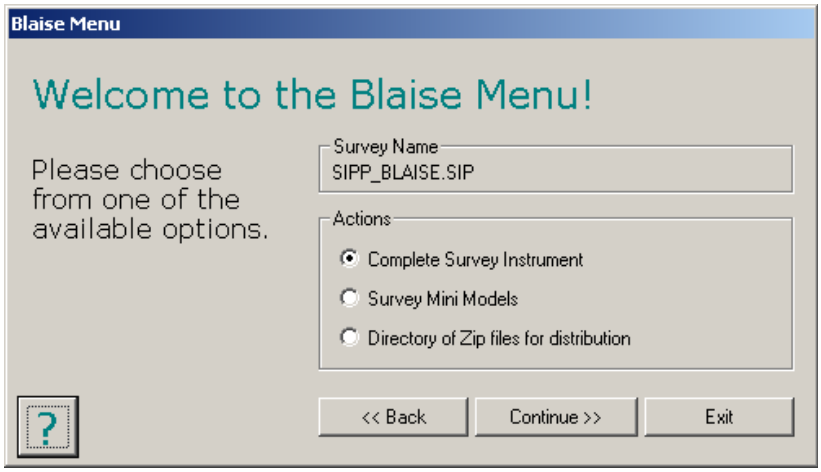


Figure 3.3



4. Why TransSAS?

The need for a centralized automated testing tool became apparent as more of our surveys were being converted to Blaise from CASES. The data processors already had an output processing system based on the data received from the CASES instruments. The output processing teams needed to modify their system for Blaise instruments. There was no easy way to look at data during instrument development without a tool to export Blaise data into a readable format.

5. Importance of Identifying Data Issues in Early Stages

5.1 Data Integrity

Identifying data processing requirements early in development helps maintain data integrity. Subsequently, data retention issues can be reduced if sponsors and data processors are able to verify data during unit testing. Our instrument development process consists of several rounds of unit or module (mini-model) testing, in which each module is placed on the TMOusers Menu for sponsors to test. Extracting data from mini-models is possible using the TransSAS application. Some critical issues can be identified during this process and can be brought to the attention of developers. It is possible at this stage to reorganize the blocks before integrating the instrument. This process is like normalizing a relational database to create data output in a format that the processors expect. Not only is the data dependable, but this process of re-organizing the blocks helps create more compact instruments, which enhances performance.

5.2 Time Efficiency for Developers, Sponsors and Processors

One of the main goals for development teams is maintaining timely releases of the deliverables to the client. Maintaining the quality of our products is another important goal. Prior to the development of TransSAS, there was no standard method of providing output to the data processors until the systems test. A systems test is conducted to verify that the instrument is able to interface with our control systems. At this stage of development the instrument is almost ready for production. The complexity of our surveys makes it difficult to quickly debug and fix a problem if it is identified at this stage. Although it can still be fixed, it can be time- consuming for developers and might create a setback in our development process causing a chain of delays for other areas as well. Therefore getting a timely feedback from the sponsors regarding data issues help in fixing major problems before instruments go to systems test. Sometimes data that is output from one interview serves as input for other follow-on interviews with the same respondent. TransSAS can be used to help create test input data for follow-on instruments on the testing system.

6. Common Tools Used to build TransSAS

The Technologies Management Office provides their sponsors with three output options to receive production data from Blaise instruments:

1. Blaise database
2. ASCII files
3. ASCII Relational files.

[1]Xiaodong Guan, at U.S. Census Bureau, has discussed in detail these three output options in his paper “Output Processing for Consumer Expenditure Survey “ and how his output data from a highly complex instrument. His paper concluded that using ASCII relational output was the best method for their office.

Consequently, the Demographic Surveys Division requested that TMO use the ASCII Relational option when implementing TransSAS. We create a Blaise-to-ASCII relational export script using Manipula, which generates the ASCII Relational files, and subsequently use Cameleon scripts to generate setup files for running SAS. The Cameleon Translator files (.CIF), can be used to generate SAS

[1] Output Processing for Consumer Expenditure Survey Xiaodong Guan, U.S. Census Bureau Howard R McGowan, U.S. Census Bureau

code for selected blocks in the instrument. We used a Cameleon script that was developed at Census to generate SAS macros, which can be easily run by data processors.

The Cameleon scripts can also generate setup files to the Oracle client server environment, Paradox, or any other database. The WinBatch programming tools are used to link all the programs to create a standalone application for distribution in a multi-user testing environment. TransSAS helps to achieve our goal of exporting data from Blaise and to generate SAS code for analysts.

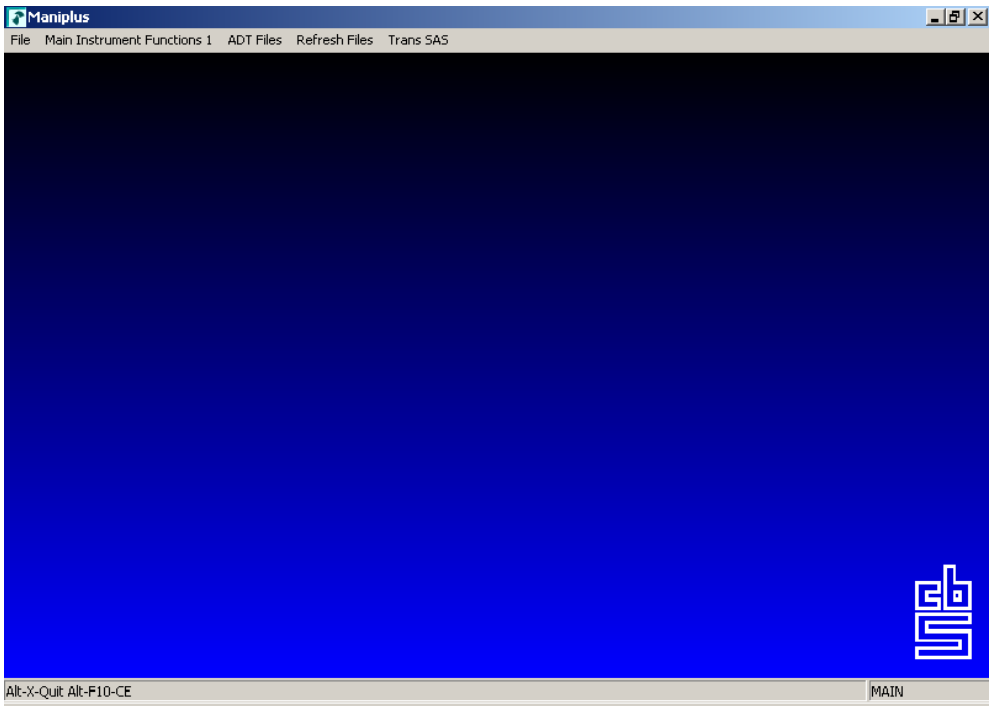
To sum up, TransSAS was built using the following software and tools.

1. Blaise
2. Manipula
3. Cameleon
4. WinBatch

7. How TransSAS works

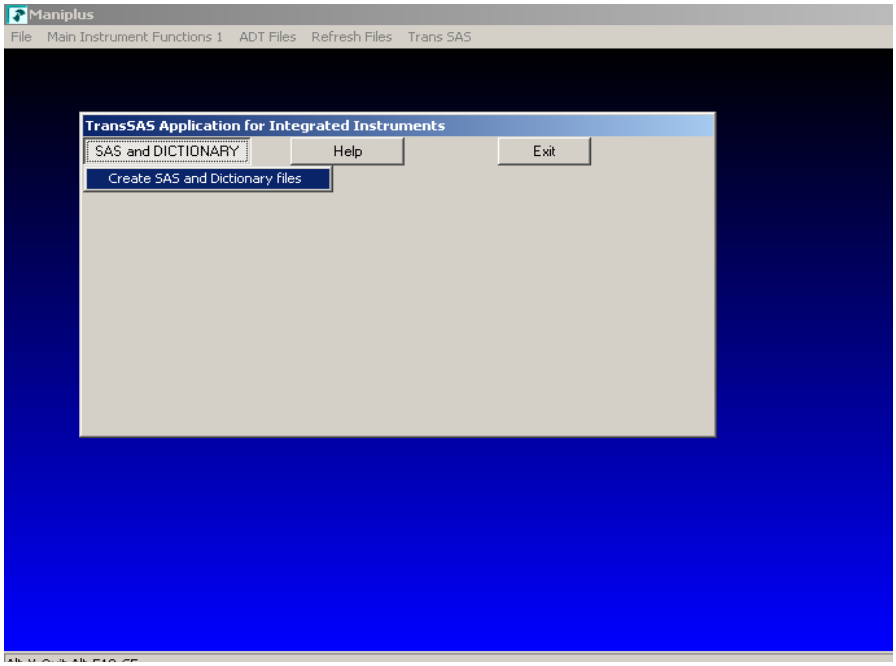
We have seen some of the common tools that have been used to create TransSAS. Now we will see how it works. The TransSAS executable is located at the survey level folder of our instruments on the TMOusers Menu. The testers and sponsors are given rights to access their respective surveys to test the instruments. They go through the preliminary screens on the TMOusers Menu (see Figures 3.1-3.3) to get to this Maniplus interface. The Maniplus interface shows several drop-down menus on the menu bar, which are based on survey specific requirements. They include options to run the instrument by selecting a case id, view audit trails, and refresh the databases. TransSAS is seen as the last option (see Figure 7.1).

Figure 7.1. Maniplus interface to test the instrument



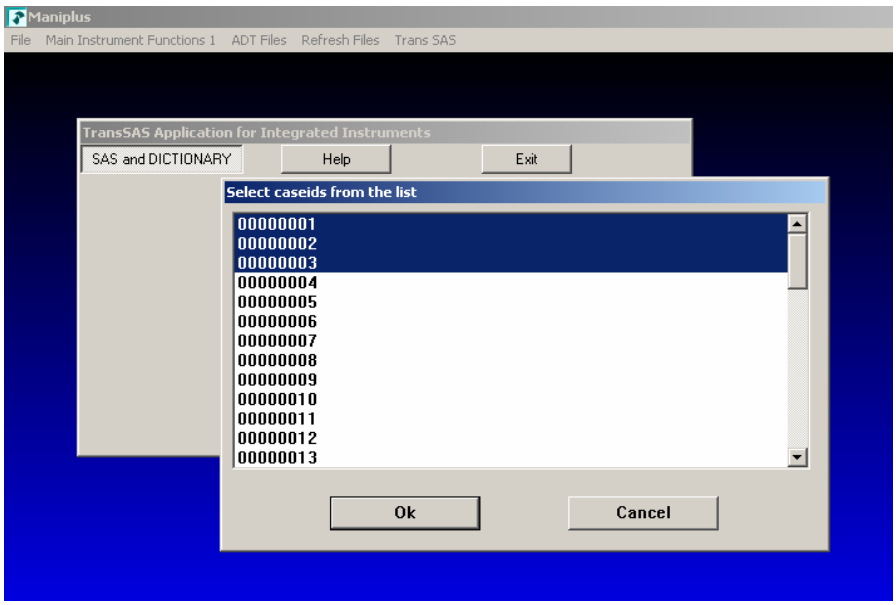
When users are at this screen they can perform several functions, such as picking a case and entering data, looking at the audit key files and so on. Now with TransSAS being an option, they can choose to output the data as well. When the TransSAS option is picked, the following screen is displayed (see Figure 7.2).

Figure 7.2



From this point on the control is turned over to WinBatch. When users pick the first option to create the SAS and dictionary files, a screen with a list of caseids is displayed (see Figure 7.3). These are cases that are available to test and therefore the cases that could contain data for output. The program allows users to select multiple cases from the list by holding down the shift key. It is important that the user selects specific cases to output data. Otherwise, if no cases are identified, the program will produce output for all cases.

Figure 7.3



Once the cases are identified, the program creates a consolidated database by combining the selected cases into one database. The ASCII relational files are output for the selected cases based on the meta data.

The Manipula script generates the ASCII Relational data files for each block. It connects the block name, block number, data model name, and ASCII data set name.

The Cameleon script has been customized to create the SAS files for these cases. It also produces a dictionary file, which contains a list of variables that are present in the instrument.

Since all the scripts are run from the WinBatch application, control does not pass back over to Maniplus until WinBatch terminates.

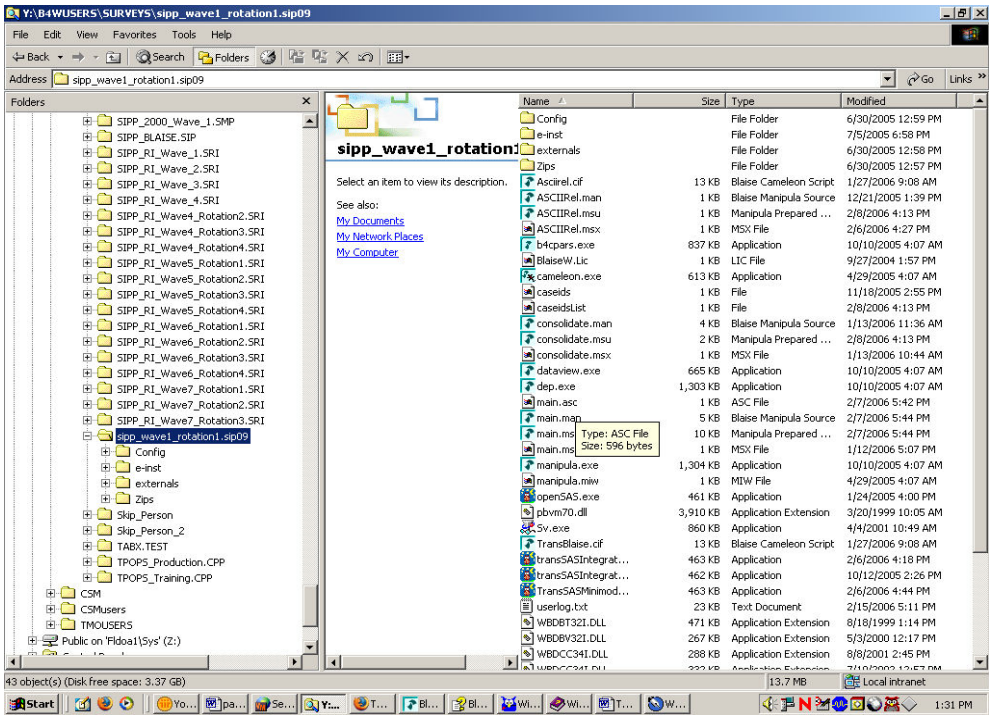
We have integrated all the scripts into one application and used the Blaise parser and executables so it can run in any environment (e.g., desktop, laptop, network etc.). We have seen the functionality of how TransSAS works. Next we will see how TransSAS is packaged and deployed on our network.

8. Packaging and Deployment

The TransSAS application was created using WinBatch, which is a very powerful general-purpose programming language. Its main purpose was to interact with Manipula and Cameleon. The WinBatch application was used to call the Manipula and Cameleon scripts using command line parameters and to allow user input. The WinBatch script is compiled to generate the TransSASIntegrated.exe. The command line parameter used is the name of the folder at the survey level for a project on the TMOusers Menu. Since we also conduct module testing, a similar executable is available for testing mini model output on the TMOusers Menu. The access to the TransSAS application for mini-models is the same as it is for integrated instruments. Testing output from mini models is important because it is one way data inconsistencies can be identified during early stages of development.

Here is some background information about the TMO standardized file names and directory structure. TMO uses 'inst' for its main project files, for instance the inst.bdb, inst.bmi, inst.bxi etc. These files are stored under the e-inst directory for the integrated instruments, and for the mini-models they are stored under the name of the mini-model. Figure 8.1 shows an example of our directory structure on the TMOusers Menu.

Figure 8.1



8.1 Scripts and Other Files Required for Integrated Instruments

As already mentioned, the required files will have to reside at the execution level of each survey. Table 8.1.1 shows the name of each program used and a detailed explanation for each. Mini-models have a similar structure except for consolidate.man file.

Table 8.1.1

Program name	Purpose	Output
AsciiRel.man	Creates the ASCII relational files The ASCIIrel.man is compiled and run inside the WinBatch script using the parser.	One data set per block in the instrument. E.g. A0, A02, A03.
ASCIIRel.cif	Creates the SAS code and dictionary files. The ASCIIRel.cif is compiled and run from inside the WinBatch script using the parser.	One data set per block is created.
Consolidate. man	At present we generate one record per caseid, (caseid.bdb). The consolidate.man combines the data for these caseids into one consolidated database.	Output is Inst.bdb
Main.man	Main.man is the script that launches the instruments on the TMOusers Menu. This is the screen where control is turned over to Manipula from the WinBatch interface. See Figure 7.1.	
Main.asc	Menu options are defined in this ASCII file.	Main.man uses this ASCII file as input.
Caseid	A text file with a list of caseids It displays what cases are available to pick.	

8.2 Executables Required for Running TransSAS on Integrated Instruments

Table 8.2.1

Programs	Purpose
B4cpars.Exe	Parser to compile the scripts
Cameleon.exe	Needed for running Cameleon
Manipula.exe	Needed for running Manipula
Sv.exe (Optional)	Needed to open SAS viewer
Wordpad.exe (Optional)	Needed to open SAS and dictionary files.
TransSASIntegrated.exe	Executable used by integrated instruments.

It is important to note that the scripts must be in sync with the Blaise data model. Hence, there is a need to compile them for every release of the instrument where the data structure has changed. This process can be time-consuming for developers. Taking some of these issues into consideration, we have automated these processes in the TransSAS application. Every time we run the TransSAS application, it will compile the scripts.

Note: The Manipula and Cameleon scripts will have to be compiled in the version of Blaise that is being used by the survey instrument. It is important for programmers to place the correct parser and executables for that particular version at the execution level or the root directory.

9. Code Examples

The TransSAS executable is called from within Main.man. This is our intermediate step before turning control over to WinBatch. A call to the WinBatch executable is placed inside the Main.man, which launches the Maniplus interface (see Figure 7.1).

- **Main.man**
  Add the MenuID to the main.asc file to select the TranSAS program.
  Use the same MenuID number to condition inside the code. Change the survey name without adding any spaces.

```
IF MenuID = '3.01' THEN
 Reslt := run('TransSASIntegrated.exe Maricopa_County')
ENDIF
```

Figure 9.1

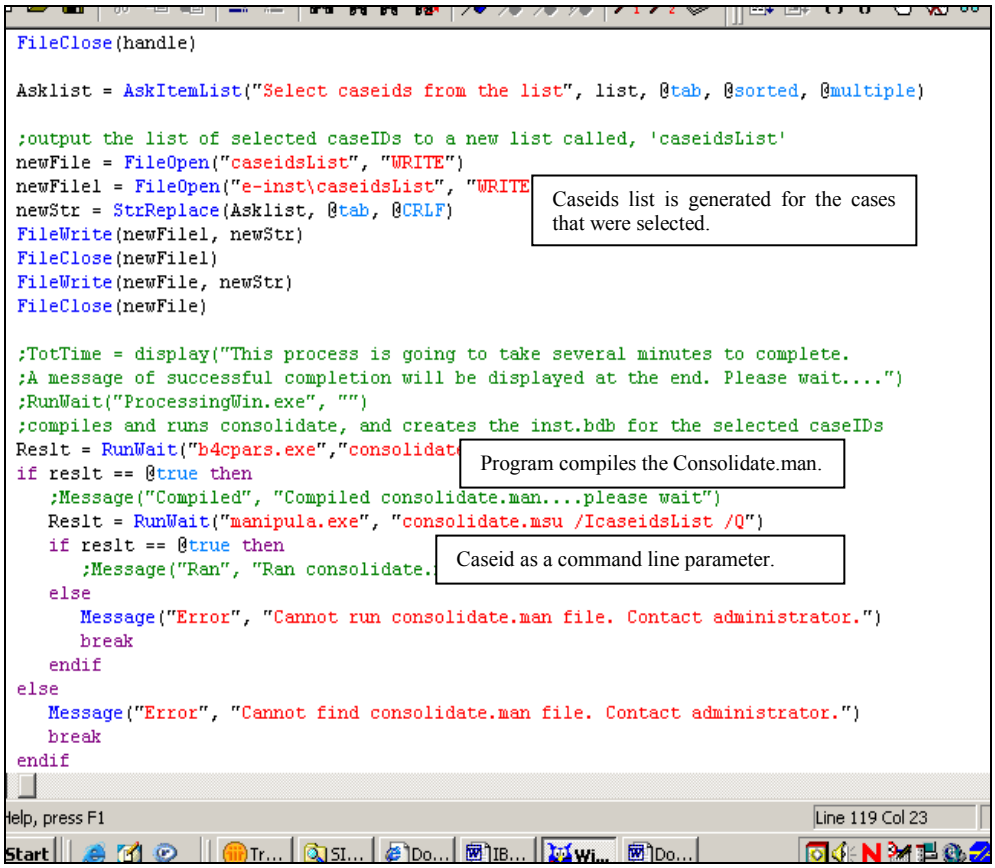


Figure 9.1 shows the WinBatch code, which was used to build TransSAS. The list of caseids is the first display (see Figure 7.3). We have also implemented functionality in the program, where a new list of caseids is stored in as ASCII file called caseidlist. The caseidlist file helps testers identify cases for which data has been extracted.

10. Snippet of the SAS code (Blaiseoutput.SAS)

- Below is an example of the SAS macro inside the Blaiseoutput.SAS file that was created by Cameleon.

```
%MACRO A76(dataname); /* corresponds to the data file number generated by
                        the Manipula script.*/
DATA &dataname.;      /* SAS dataset name  */
    Input
        FPRIMARY  $ 8.
        INSTANCENUMBER  $ char5
```

- The example below shows the output for a subroutine in the instrument, which is not part of the database. It is useful to identify and remove subroutines.

```
/*FileNumber:= 76   Block Name = RoundToNearest*/
```

- The location of the ASCII Relational files for the INFILE statement inside the SAS macros has been parameterized. See sample code from Cameleon below.

```
[Spaces:= 2]
[/]%MACRO[:3][FileNumber](dataname,InputPath,InputFile,ext=[FileNumber],
                                                    FILESTAMP=0);
[/]DATA &dataname.; /*FileNumber:= [BLOCKNUMBER] [Block Name=blockname]*/
[Spaces:= Spaces + 2]
[:Spaces]INFILE "&InputPath.&InputFile..&ext" MISSOVER PAD
                                                    LRECL=RecordLength];
```

11. Current Limitations

It is time-consuming to run TransSAS on the network. Things that affect the performance of TransSAS may be the following:

- Number of cases selected.
- Size of the instrument
- The network speed.
- Consolidating the individual databases.
- Compiling the scripts
- Cases locked by another user

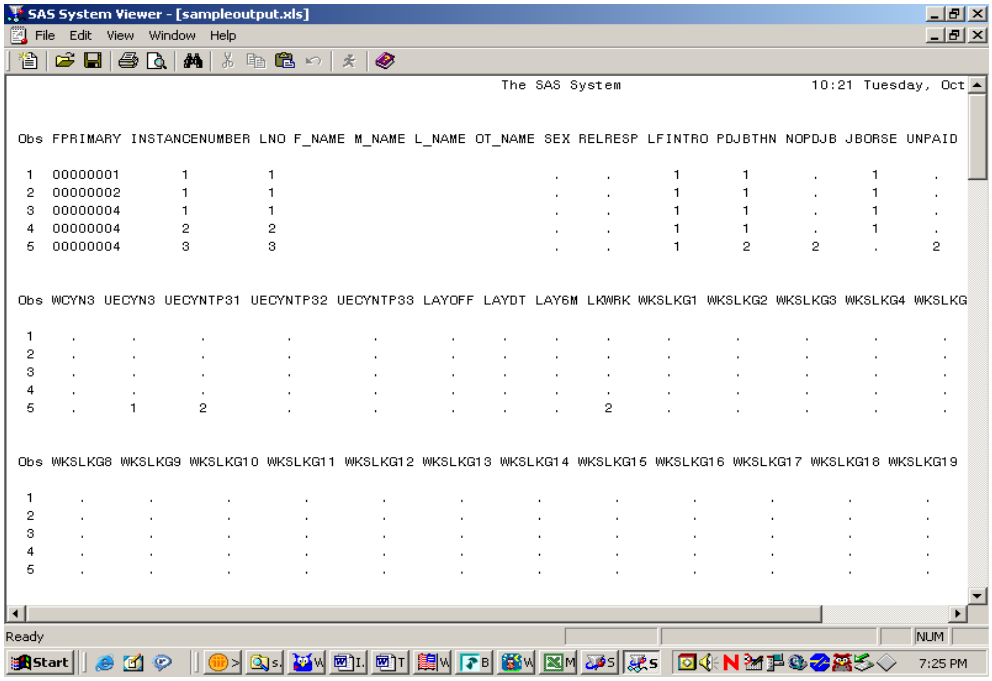
12. Benefits

- The program is run in "quiet" mode. Therefore, it does not interfere with other work. The users will see a message box that asks them to choose a destination folder for the output once the program is completed.
- Compiling scripts is beneficial as it saves on setup time for authors: They do not have to compile to keep the scripts in sync. TMO has instruments in several different versions of Blaise. Therefore, compiling the scripts is highly recommended.
- The users are given the ability to choose their output destination folder.
- Reduces user errors: The program automatically deletes the SAS, dictionary and data files from the network drive. By doing this, we can avoid users from accidentally deleting any other important file from the network.

13. Sample SAS Output

Figure 13.1 shows a sample of SAS output ".lst" files which was generated by running one of the SAS datasets which was output from Cameleon.

Figure 13.1



Some sponsors have tested their instruments using TransSAS and have also run the SAS macros to verify the data. So far we have got positive feedback from them. Table 13.1 is an example of the SAS output created by the sponsors using SAS code that was generated by Cameleon.

Table 13.1

b06_varname	n05_varname	b06_length	n05_length	b06_type	n05_type	b06_blockname	b06_filename	Outcome
AFUEL	AFUEL	1	1	C	C	BlkBAC	b12	OK
AFUELQ	AFUELQ	8	1	N	C	BlkBAC	b12	OK
AFUELQSP	AFUELQSP	30	30	C	C	BlkBAC	b12	OK
AFUELV	AFUELV	1	1	N	C	BlkBAC	b12	OK
	AFUR		1		C			Drop
AGE	AGE	8	3	N	N	BRELATION	a62	OK

14. Conclusion

The TransSAS application can be deployed in a multi-user testing environment, but it is not mandatory for everyone to set up their environment in this manner. Since it is an executable, it can be deployed on a client-server environment, packaged on a laptop, or placed on a FTP site for external sponsors to download. Other languages such as VB, Delphi, C, and C++ can be substituted for WinBatch to run TransSAS.

To summarize, implementing TransSAS for instrument testers and data processors to retrieve test output from Blaise instruments provides the following benefits:

1. The application provides an easy way for sponsors to access their output data to ensure better data quality.
2. The application improves time efficiency for programmers and data processors.
3. The application is low maintenance and cost effective

15. Future Plans

There is room to improve the TransSAS application. We are looking at making some enhancements in the future. We can build an application similar to TransSAS using different object-oriented languages where we can explore the following possibilities:

1. Deploying it in client-server environment.
2. Develop the application using the Blaise Component Pack along with any language that supports the Microsoft COM (Common Object Model) specification like Visual Basic, Visual Basic for Applications, Delphi, C++, et cetera.
3. To improve the speed of the application we have considered the following methods:
   a) Selectively compile the scripts by incorporating the version info and the date changed into the instrument so that it can be part of the meta data. The WinBatch program can then be modified to read this and selectively compile the scripts by comparing the dates.
   b) Explore the possibility of modifying the Cameleon/Manipula scripts, such that it eliminates creating empty data files.
   c) Lastly, create a batch program to run this application behind the scenes

References

Guan, Xiaodong & McGowan, Howard R. (2001) : Output Processing for Consumer Expenditure Survey. Presentation for the International Blaise Users Conference.

Jellema, T.R. (2001) : Blaise to OLE-DB-Relational Data Migration? Yes! Presentation for the International Blaise Users Conference.

Schou, Roger. (2003) : Creating Code to Convert Blaise to ASCII for SAS Input (Using the Blaise API within Visual Basic). Presentation for the International Blaise Users Conference.

Wensing, Fred. (2004) : Instrument assembly, documentation and release. Presentation for the International Blaise Users Conference.

Acknowledgements

I would like to acknowledge the following people from our TMO Authoring staff. I would like to thank Karen, Rob, Tom, and Ed for reviewing and editing this 10th International BLAISE Users Conference presentation paper. I also would like to thank Yogeeta who has jointly programmed the TransSAS application and has provided valuable input into this paper. I would like to thank Emily for providing valuable input about our TMO testing environment.

Karen Bagwell, U.S. Census Bureau, [Assistant Division Chief]
Rob Wallace U.S. Census Bureau, [Team Leader]
Thomas Spaulding, U.S. Census Bureau, [Team Leader]
William Edward Dyer, U.S. Census Bureau [Section Chief]
Yogeeta Purohit, Contractor at U.S. Census Bureau [Programmer/Analyst]
Emily Johnson, U.S. Census Bureau [Computer Specialist]

Developing an Integrated Household Survey (IHS) Blaise Questionnaire from Four Major Social Surveys

Tim Burrell (Office For National Statistics, UK)

1. Introduction

The Integrated Household Survey (IHS – formerly known as the Continuous Population Survey is a project to integrate four government surveys on which the UK Office for National Statistics (ONS) leads, and replace them with a single modular survey. A comprehensive integration of the entire survey process is proposed: 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.

This paper first details the IHS project, the paper then focuses on one aspect of the survey's development: developing an IHS Blaise questionnaire from four major social surveys. The IHS is due to begin in January 2008. The development of this has involved two field trials during 2005, a larger scale pilot in January 2006 and there will be a further pilot in 2007. This paper will describe the first attempt at programming the Blaise questionnaire for the field trials, lessons learnt, and the new design for the pilot in 2006. The paper will discuss the issues involved in designing and programming a Blaise questionnaire to handle the different topics and rules of the four surveys with a view to producing coherent and consistent outputs.

2. Background

The aim of the IHS is the need for UK National Statistics to produce better information on key social and economic variables between decennial censuses. These demands cannot be met within current survey arrangements.

The central objectives of the CPS are to:

- develop a world class modular survey system, 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 decisions;
- increase the precision of most existing 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.

There are four surveys planned for integration. These are:

- the Labour Force Survey (LFS);
- the General Household Survey (GHS);
- the Expenditure and Food Survey (EFS);
- the National Statistics Omnibus Survey (OMN).

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). More information on these surveys is in Appendix A.

3. Design of the IHS questionnaire

The proposed IHS involves a single modular Blaise questionnaire instrument which meets the information needs of the individual surveys. This questionnaire will comprise of a core set of questions, which will be administered to the whole sample and topic modules. The topic modules will meet the needs of the four surveys and will only be administered to a proportion of the sample. The limited number of different interview combinations will be composed of both the core and selected topic modules. Interviewers in an integrated field force will administer all interview combinations.

Designing a suitable core module is essential to the success of the IHS. The core module is intended to collect information about key Census type variables and other socio-demographic and labour market indicators. Because the core module is administered first, and to all respondents in the survey, it needs to be concise and relatively straightforward. This core module will be used as an aid to weight topic data as it will have a very large base of responding adults.

This core module will be followed by survey specific questions from the original four surveys. In some cases, topics covered in the different surveys are common. For example, detailed questions on employment and education are required on the GHS and LFS type interview combinations but not necessary on the EFS and Omnibus.

There are a number of other procedural differences between the surveys involved in the IHS. LFS interviewers readily accept proxy interviews as the interview is short and the information collected mainly factual. Proxy interviews are discouraged on the GHS and EFS as a lot of the information collected would only be known by the individual. The Omnibus does not accept proxy interviews due to the high volume of opinion questions.

The EFS has a diary questionnaire. Each individual within a sampled household must complete a two week diary of expenditure. Interviewers who have only ever worked on the LFS will not be experienced how to ‘sell’ a much longer survey involving a diary. Other differences between the LFS and GFF surveys include rules on whether to interview students living away from home and how to deal with multiple households.

One of the main challenges is the development of a questionnaire which can route effectively through the relevant topics and guide the interviewer to use practices that they may not be familiar with.

4. Questionnaire design for field trials

The first IHS field trial took place in late 2005. The purpose of the early field trials was to both test the questionnaire but also to test working patterns and most importantly a new training scheme to educate all interviewers in the different practices on each of the surveys. The introduction of the IHS will involve a number of changes to interviewer working practices as well as a new questionnaire. At present, ONS has a field force dedicated to the LFS. Another field force deals with all other ONS surveys (the exception being the International Passenger Survey

which also has a unique field force based at ports and airports). This field force is named the General Field Force (GFF). The IHS involves combining these two large field forces. All interviewers from these field forces will need to be able to work on all the surveys involved in the IHS.

Field force integration presents more problems than just that of interviewers needing to be taught new surveys with different subject matter. There are many practical differences between the two types of surveys (LFS and others). The main difference is in field pattern and the amount of time each interviewer has to complete their cases. LFS interviewers receive a weekly batch of cases. At the end of each week all cases must be returned. There is some scope to re-issue these cases but the hope is that all cases are dealt with in one week. In the following week, a new batch of cases is issued. On GFF surveys, cases are drawn in monthly batches. Cases are drawn and sent to interviewers on a monthly cycle. Interviewers have one month to work through all the cases. This allows far more flexibility in working patterns. One of the goals of the IHS is to test how these working practices can be integrated.

Although the trial was not too concerned with outputs and consistency with current estimates (due to the small sample size) it was still important to have a functional integrated single questionnaire based on current ONS standards and practices. The next section describes the development of the Blaise questionnaire for these field trials.

Since 1990, there has been an ongoing programme of harmonisation of questions, concepts and classifications. Household surveys at ONS and across the wider UK Government Statistical Service (GSS) have adopted many common standards. At ONS we have a bank of standard Blaise question blocks based on these harmonised topics as well as other ONS standard topics. For example, household composition, ethnicity, country of birth and standard questions on employment, health and qualifications. Where these standard questions are available and if they are appropriate, a survey can take the block of questions and insert them into a questionnaire. Therefore the same questions are used across all surveys. In an integrated survey they can be administered across all module types as a 'core' set of questions, if appropriate. The IHS aims to build on and improve standardisation.

Before any cases are sent to interviewers they are assigned a code depending on what the type of interview the case is. This is essential for the interviewer to know what to expect and also for routing through the questionnaire.

4.1. Core interview

The first stage in designing the questionnaire was to establish which topics should be included in the core interview. The core includes information about key Census type variables and other socio-demographic and labour market indicators.

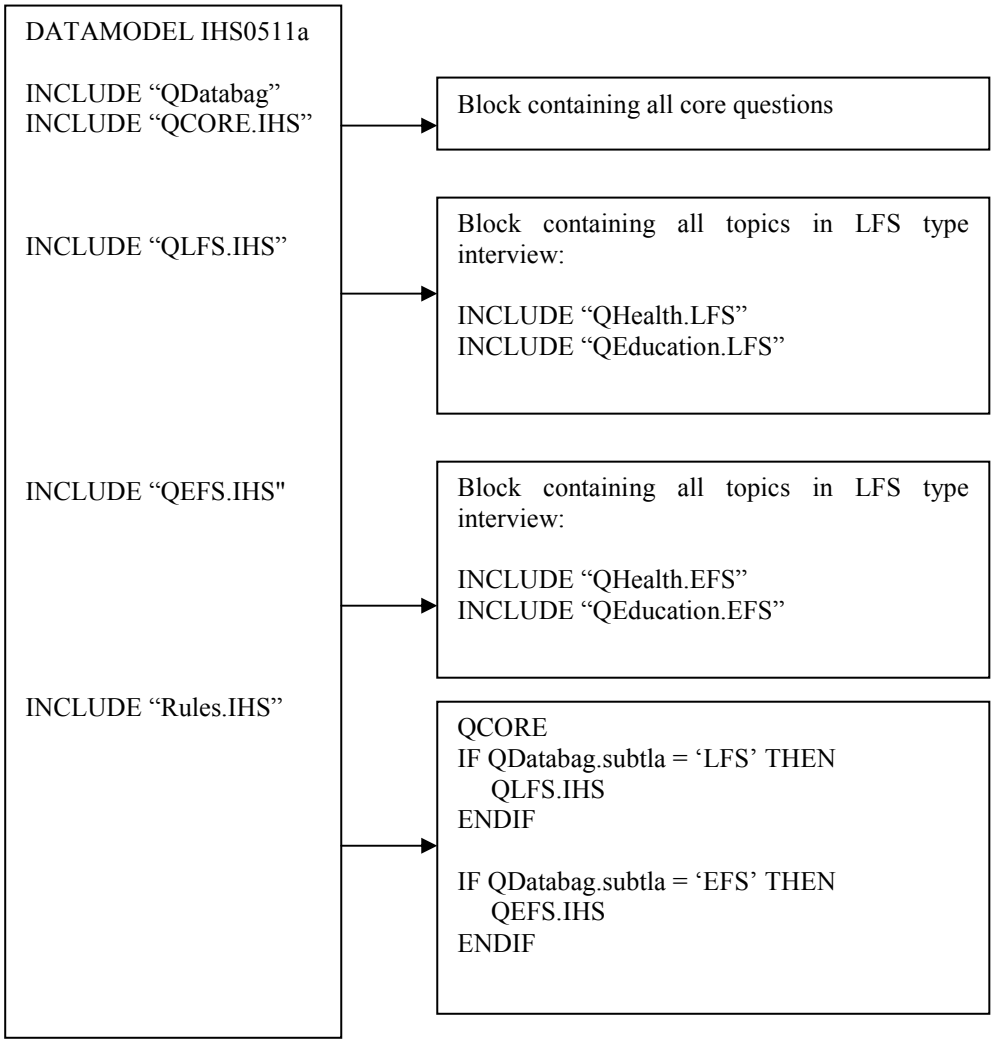
Some surveys collect more detailed information on certain topic areas. For example, an Omnibus type interview only asks a few education related questions to code the respondent's highest qualification, the LFS has a far longer set of questions to establish the same derived variable as well as many other education related variables. The shorter set of questions were included in the core and where necessary, supplementary education questions were asked later in the interview.

4.2. Programming the topic modules

The four original survey questionnaires are already programmed in Blaise. For simplicity, in terms of programming and time, once the core questionnaire had been devised and programmed, it was possible to remove these core questions from

individual survey questionnaires and add the survey specific questions into the IHS questionnaire. This was fairly straight forward, due to way we programme Blaise questionnaires at ONS. Figure 1 shows the simplified version of the Datamodel.

Figure 1. Datamodel structure for field trial questionnaire

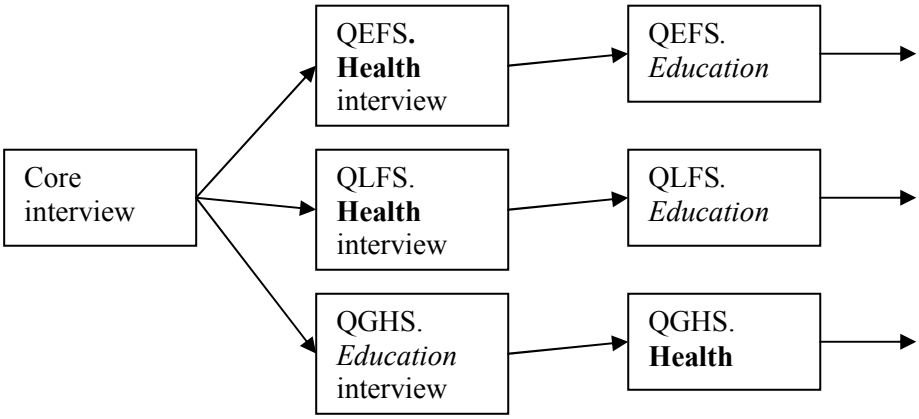


The diagram above shows a simplified version of the Datamodel. The right side of the diagram shows an example of the contents of the INCLUDE files. The first block, *'QDatabag'* is where information is stored about the type of interview. All core questions are contained in the block *'QCore.IHS'*. The survey specific topic blocks follow and are contained within an include file for each of the different survey types. *QLFS.IHS* contains the survey specific questions for the LFS which are not already in the core. Likewise, *QEFS.IHS* contains the survey specific questions for the EFS.

The questionnaire, programmed using this method, worked well in the field. The interviewer could complete the interview and all the correct information was asked depending on the type of interview. There were however some problems. Firstly, repetition of questions, while the core records basic health and education, some modules follow these up in great detail. This interview involved moving away from these topics and having to go back to them at a later stage. For example, key education questions are asked in the core. Then more detailed education questions are required at a later stage in the interview. This resulted in the interview being disjointed and respondents asking 'haven't I already answered that question?' In

this model, there is no consistency in subject order. For one interview combination health was followed by education. For another combination education was followed by health. Interviewers found this confusing as interview did not flow very well. An example is provided in figure 2.

Figure 2. Questionnaire structure for field trials:



One of the goals of the IHS is to have an integrated processing system where data checks can are performed on the data as a whole. It is desired that following the interview a standard set of processes using standard tools can be carried out on the whole dataset before being delivered to researchers for in depth analysis. At present, the same checks take place on the four surveys but are using different tools. An integrated processing system will remove this duplication of effort and cut down the number of analysis tools used.

The structure of the Blaise questionnaire used in the field trial led to problems at the data processing stage. This is because the same variables appear in the datafile more than once but with different names. For example, *QEFS.Health* and *QLFS.Health* contain the same variables but they are listed separately in the datafile. (Figure 3 provides an example)

Figure 3. Data structure during field trials:

Type of interview	Case number	Core info	QEFS. Health	QLFS. Health	QEFS. Education	QLFS. Education	QGHS. Education
EFS	001	☑	☑	●	☑	●	●
EFS	002	☑	☑	●	☑	●	●
LFS	003	☑	●	☑	●	☑	●
LFS	004	☑	●	☑	●	☑	●
GHS	005	☑	●	●	●	●	☑
GHS	006	☑	●	●	●	●	☑

The datafile was very large due to the duplication of fields. This meant that it was not possible to process the whole datafile using standard ONS systems. The datafile had to be split into topic type before it was processed.

On the whole the questionnaire worked in the field. However, problems with repetition within the questionnaire and interview flow, and subsequent problems with processing data meant we had to restructure the questionnaire for the pilot.

5. The pilot questionnaire

The next stage in the IHS development was a larger pilot in starting in February 2006 Previous field trials had established that interviewers could adapt to working on the IHS. This pilot gave us the first opportunity to compare outputs and response rates with the existing surveys.

5.1. Design of the pilot questionnaire

For the pilot, we needed to ensure no variables were duplicated. We hoped this would help with both the flow of the interview. This would also provide a platform from which to develop an integrated approach to processing the data.

The core section contains questions on basic employment details. The full LFS style interview contains far more detailed questions on employment. It was decided that for the pilot questionnaire we would ask whole topics together and route to the appropriate questions based on interview type. The example shown in figure 4 illustrates the interview structure for the employment section. The first three blocks listed (*QTHComp, QEthnic* and *QCob)* are blocks which, amongst others, make up the core interview. *QWrk1* and *QWrk2* make up the core information on employment. *QWrkdet* contains all the detailed work information and only applies to the LFS interview type.

Figure 4. Questionnaire structure for pilot

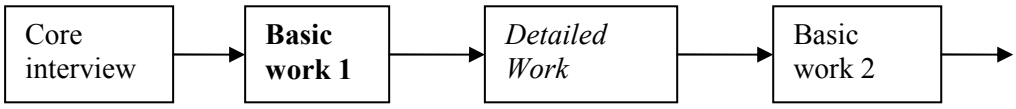
```
DATAMODEL IHS0602a

INCLUDE “QTHCOMP.IHS”
INCLUDE “QEThnic.IHS”
INCLUDE “QCOB.IHS”




INCLUDE “QWrk1.IHS”
INCLUDE “QWrkdet.IHS”
INCLUDE “QWrk2.IHS




INCLUDE “RULES.IHS”
```

```
RULES

QTHCOMP
QEthnic
QCOB




QWrk1

IF QDatabag.subtla = ‘LFS’   THEN
    QWrkdet
ENDIF

QWrk2
```

The diagram above shows, on the left, a simplified example of the Datamodel with each block referenced. On the right the diagram shows the rules section with the detailed work questions only being asked on LFS type interviews. The same practice occurs in other topic areas during the interview, for example, health, education and benefits. The result was that all variables are referenced just once in the questionnaire. This helps with the flow of the interview as questions will always appear to the interviewer in the same order. It also cuts down the size of the datafile produced.

Figure 5 shows the new questionnaire structure. After the first core section the work section is asked. Basic work 1 and basic work 2 (*QWrk1* and *QWrk2* in figure 4) are asked to everyone. Detailed work is asked in LFS type interviews. After the work section, the interview returns to the core, if appropriate.

Figure 5. Questionnaire structure for pilot



With the new structure in place, the datafile is much shorter as all the duplication has been removed. The data can be processed as a whole using ONS standard processing systems. Figure 6 shows an example of the structure of the datafile

Figure 6. Data structure for pilot

Type of interview	Case number	Basic work 1	Detailed work	Basic work 2
EFS	001	☑	●	☑
EFS	002	☑	●	☑
LFS	003	☑	☑	☑
LFS	004	☑	☑	☑
GHS	005	☑	●	☑
GHS	006	☑	●	☑

All basic work information for all interview types is held in the same place and all checks on work information or derived variables can now be run in one go.

6. Other developments

The IHS provides an opportunity for ONS to review current practices across the four current surveys. Although we have standards for Blaise programming and screen design and layouts, there are still some differences between surveys. The development f the IHS questionnaire provides an opportunity to remove these differences and standardise the whole questionnaire. A great deal of change is involved for interviewers, one of the goals of the questionnaire is that it should be easy for the interviewers to navigate and use.

6.1. Concurrent interviewing

At the time of the IHS questionnaire development, the LFS was the only survey involved which had a panel element. Five interviews are conducted with each selected household on a quarterly basis. All first wave interviews are conducted as face-to-face interviews with follow up interviews by. Due to the high volume of telephone interviews, respondents are interviewed consecutively.

The primary interviewing mode for all the other surveys is face-to-face. These interviews use concurrent interviewing. The different questionnaire topics are broken up and asked in turn to each respondent in turn. This helps keep all respondents interested and involved in the survey.

It was decided that, for the IHS questionnaire, concurrent interviewing would be used for all interview types. This meant a change for LFS telephone interviews. The main criticism from telephone interviewers was: if a case is partially completed, for example, in a four person household, the interview of only two people is completed. On re-entering the case to interview the final two people the responses from the original interview are present in the questionnaire and there is a danger they can be overwritten. This is also a problem of face-to-face interviews but occurs far less.

To avoid this problem, a category was added to an existing field to allow the interviewer to code whether an individual's interview had been completed. If this is the case all the responses from the completed interview were set to 'KEEP' fields and were 'greyed out' in the data entry programme. This avoids any overwriting and was very well received by the interviewers.

6.2. Q by Q Help

For both the field trial and the pilot, we enhanced the online help available to interviewers. The Q by Q online help facility contains procedural and question specific help, this is described in a further paper to be written and presented at IBUC 2006 called 'Training and Learning using Blaise'(Setchfield 2006):.

6.3. Additional Blaise features

We have introduced the following features on the IHS questionnaire which were not standard on ONS surveys before 2006:

- Q by Q help facility, as mentioned above.
- A calculator is now available for interviewers to use at anytime during the interview.
- For certain date fields within the questionnaire, a date picker is available.
- All parallel blocks are now available through using tabs or through using the keyboard.

7. Next steps

The IHS questionnaire is far more advanced now. The process of preparing for the his, which began in 2004, has illustrated the importance of a coherent approach to survey design, each part of the survey process has to work together to achieve the best results.For example, it is important to consider the outputs when writing the Blaise questionnaire..

ONS is currently developing an integrated processing system for the IHS. The next pilot, to be carried out in 2007, will also test this system and attempt to reproduce current outputs.

8. References

Rob Bumpstead, (2004): Integration of UK government household surveys: Development of a modular Blaise instrument for a proposed Continuous Population Survey. International Blaise Users Conference 2004

Colin Setchfield, (2006): Training and Learning using Blaise. International Blaise Users Conference 2006

Appendix A

Survey for integration:

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.

The General Household Survey (GHS), the first 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.

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.

Session 2.2: Cati Considerations

Can We Reach You by Telephone?
Fannie Cobben
(Statistics Netherlands)

Experiences of Usage of Blaise in CATI Surveys
Stefania Macchia & Manuela Murgia
(ISTAT, Italy)

NASS CASIC Survey Administration System
Everett Olbert & Roger Schou
(National Agricultural Statistics Service, USA)

A Multi-mode CATI-Web Survey Experience with Blaise
Craig Ray
(Westat, USA)

Can We Reach You by Telephone?

Fannie Cobben (Statistics Netherlands)

1. Introduction

Blaise supports various data collection modes. Statistics Netherlands has always favoured computer assisted face-to-face interviewing (CAPI) for its social and demographic surveys. Due to the persuasive power and assistance of interviewers visiting selected individuals or households, nonresponse is relatively low and data quality is high. However, the costs of this mode of interviewing are relatively high. A large group of trained interviewers is required that is distributed all over the country. To reduce costs, Statistics Netherlands is now considering changing some of its CAPI surveys into computer assisted telephone surveys (CATI). By concentrating interviewers in one call centre, a smaller group is sufficient. No more time is spent on travel, and this also means no more travel costs are involved.

The sampling frame for a telephone survey is obtained by linking telephone numbers to the names/addresses from the Municipal Base Admininstration. This is achieved by handing over the names/addresses to the Dutch telephone company KPN. However, such links will only be established for individuals with a listed, fixed-line number. As a consequence, individuals with an unlisted fixed-line number, indviduals with only a mobile phone, and individuals without a phone, will never be selected for a telephone survey. Currently, the percentage of individuals with a listed, fixed-line number is estimated to be between 60% and 70%. This means that there is a substantial undercoverage of 30% to 40%.

Although a possible change from face-to-face interviewing to telephone interviewing may substantially reduce the costs of the surveys, there is also a potential drawback: it may reduce the quality of the produced statistics. In this paper we explore the possible effects of changing the mode of data collection in one of the Statistics Netherlands surveys: the Integrated Survey on Living Conditions, denoted by its Dutch acronym POLS; 'Permanent Onderzoek LeefSituatie'.

2. Survey errors

Usually, one of the main objectives of a sample survey is to compute estimates of population characteristics. Such estimates will never be exactly equal to the population characteristics; there will always be some error. This error can have many causes; see e.g. Bethlehem (1999). The ultimate result of all these errors is a discrepancy between the survey estimate and the population characteristic to be estimated. This discrepancy is called the *total error*. Two broad categories can be distinguished contributing to this total error: sampling errors and non-sampling errors.

Sampling errors are introduced by the sampling design. They arise due to the fact that estimates are based on a sample and not on a complete enumeration of the population. Sampling errors vanish if the complete population is observed. Since only a sample is available for computing population characteristics, and not the complete data set, one has to rely on estimates.
Non-sampling errors may even occur if the whole population is investigated. They denote errors made during the process of recording answers to the questions. Non-sampling errors can be divided into various types of errors:

- An *overcoverage error* means that elements are included in the survey that do not belong to the target population.
- A *measurement error* occurs if a respondent does not understand a question, or does not want to give the true answer, or if the interviewer makes an error in recording the answer. Also, interview effects, question wording effects, and memory effects belong to this group of errors. A measurement error causes a difference between the true value and the value processed in the survey.
- *Undercoverage* occurs when elements of the target population do not have a corresponding entry in the sampling frame. So, these elements can never be contacted.
- Another important non-sampling error is *nonresponse*. It is the phenomenon that individuals selected in the sample do not provide the required information because they have not been contacted, are unable to co-operate or refuse to.

In the ideal situation, the individuals participating in a survey are selected by means of a probability sample. If selection probabilities of all elements in the population are known and strictly positive, unbiased estimates can always be computed, see Horvitz and Thompson (1952). However, due to non-sampling errors like undercoverage or nonresponse, actual selection probabilities can be zero (e.g. if an element is not included in the sampling frame, it can never be selected) or unknown (e.g. if selected individuals refuse to co-operate).

3. Analysis

3.1. Introduction

The aim of our research is to answer the following question:

What is the effect on the quality of estimates of population characteristics, when going from a face-to-face survey to a telephone survey and, consequently, c*an we adjust for this effect?*

We thereby focus on undercoverage and nonresponse bias. In section 3.2 we introduce the datasets that we use for our analysis. The methods that we apply to adjust for undercoverage- and nonresponse bias are explained in section 3.3. How we apply these methods to our data in order to answer our questions is explicated in section 3.4.

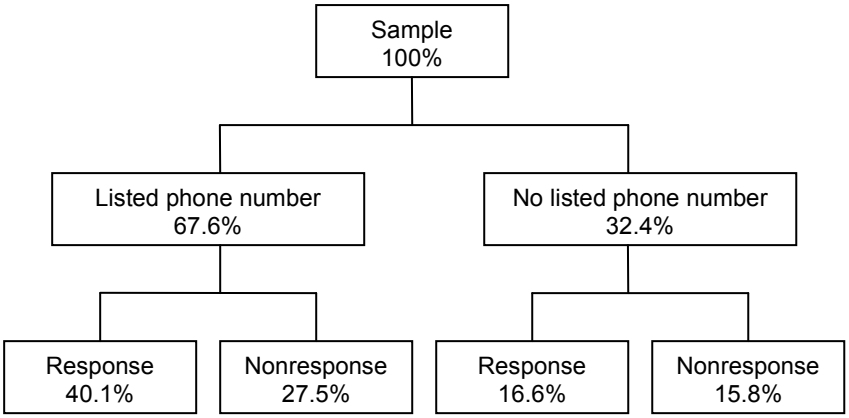
3.2. The data

POLS is a continuous face-to-face survey. Every month a sample of 3.000 persons is selected and interviewed face-to-face. The survey has a modular structure; there is a base module with questions for all sampled persons and in addition there are a number of modules about specific themes (such as employment situation, health and justice). The sampled persons are selected for one of the thematic modules; the base module is answered by everyone.

The target population is not the same for every module. However, all target populations at least consist of persons of age 12 and older. Persons are selected by means of a stratified two-stage sample. In the first stage, municipalities are selected within regional strata with probabilities proportional to the number of inhabitants. In the second stage, an equal probability sample is drawn in each selected municipality. In this paper, only persons of 12 years and older are regarded. These persons all have the same first-order inclusion probability. The focus of this research lies on the questions in the base module.

It is difficult to distinguish the contributions of both types of non-sampling errors in the total bias. Figure 3.2.1. describes the situation graphically for the POLS 2002 survey.

Figure 3.2.1: Graphical representation of undercoverage and nonresponse



In the ideal situation where the sampling frame exactly covers the population, the bias will only be caused by nonresponse of both individuals with and without a listed phone number. Face-to-face interviewing is close to this situation. The percentage response is 40.1% + 16.6% = 56.7%.
In case of a telephone survey, the bias is caused both by undercoverage and nonresponse. Only 40.1% of the original sample will respond.
Note that the percentage response among the listed phone numbers is much higher (59.4%) than for no listed numbers (51.1%). Apparently, individuals without a listed phone number behave differently from individuals with a listed phone number.

As we already said, POLS is a face-to-face survey. For our research we need both a face-to-face survey data file and a telephone survey data file. We construct the telephone survey data file from the face-to-face survey data file. We can do so by matching the records in the face-to-face data file to the telephone directory provided by the Dutch telephone company KPN. Deleting records without a listed, fixed-line telephone number provides us with the data file that would have been used had the survey been performed by telephone interviewing.

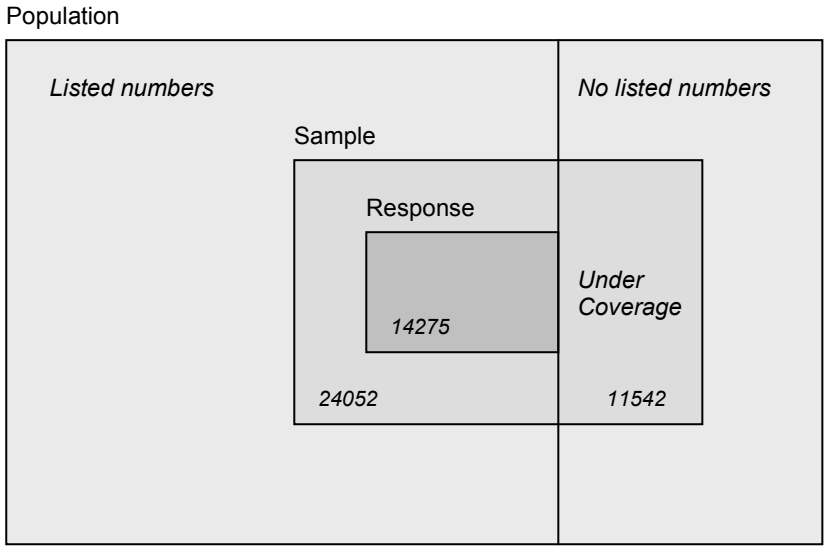
An advantage of this artificial way of generating a telephone survey data file is that possible mode effects caused by differences in face-to-face and telephone interviewing are avoided. The face-to-face sample consists of 35594 individuals, 24052 of which have a listed, fixed-line telephone (67,6%). In Table 3.2.1, the number of available records is displayed.

Table 3.2.1 Available data for both surveys

	Face-to-face survey	Telephone survey
Sample size	35 594	24 052
Number of respondents	20 168	14 275
Percentage of respondents	56.7%	59.4%

Figure 3.2.1 graphically displays the population and the two datasets.

Figure 3.2.1 Graphical representation of the division of the population with respect to telephone ownership



There are two kinds of variables that we use in the analysis: *background variables* and *target variables*. Background variables are available for both respondents and nonrespondents. These variables come from registers like the Municipal Basic Administration (GBA) and the Centre for Work and Income (CWI). The target variables are the answers to the survey questions; these are only available for respondents. The variables that we use in our analysis are displayed in Table 3.2.2.

Table 3.2.2. Background- and target variables used in the analysis

Background variable	***Categories***
Gender	Male, Female
Age in 3 classes	0 – 34, 35 – 54, 55 +
Age in 15 classes	12 – 14, 15 – 17, …, 70 – 74, 74 +
Marital status in 2 classes	Married, Not married
Marital status in 4 classes	Married, Not married, Divorced, Widowed
Ethnic group in 8 classes	Native; Moroccan; Turkish; Surinam; Netherlands Antilles/Aruba; other non-Western non-native; other Western non-native
Province of residence and three largest cities in 15 classes[1]	
Region in 4 classes	North, East, South, West
Degree of urbanization in 5 classes	Very low, Low, Average, High, Very high
Household size in 5 classes	1, 2, 3, 4, >4
Household type in 5 classes	Single, Couple, Couple with children, Single parent, Other
Interviewer month in 12 classes	January, February, …, December
Listed fixed-line telephone in 2 classes	Yes, No

[1] The Netherlands is divided in 12 provinces. The three largest cities are Rotterdam, The Hague and Utrecht city. Amsterdam is excluded from the analysis because no listed numbers were matched.

Background variable	*Categories*
Average house value in 14 classes	Missing; 0; 0 – 50000; 50000 – 75000; …; 275000 – 300000; 300000 – 350000; > 350000 (euro)
% non-natives in 6-digit postcode area in 8 classes	0 – 5%, 5 – 10%, …, 40 – 50%, 50% and more
Target variable	***Categories***
Employment	12 hours or more, less than 12 hours, unemployed
Educational level	Primary, Junior general secondary, Pre-vocational, Senior general secondary, Secondary vocational, Higher professional, University, Other
Religion	None, Roman-Catholic, Protestant, Islamic, Other

3.3. The methods

Linear weighting

The linear weighting technique is based on the generalized regression estimator; see also Bethlehem (1988).

Let the target population of a sample survey consist of N individuals 1, 2,…, N. Let Y denote a target variable of the survey. Associated with each individual k is a value Y_k of this target variable. Assume that the aim of the sample survey is to estimate the population mean of the target variable

$$\bar{Y} = \frac{1}{N}\sum_{k=1}^{N} Y_k \ . \qquad (3.3.1)$$

Furthermore, let X be a vector of auxiliary variables or covariates, with values X_k, for $k = 1, 2, \ldots, N$.
A sample selected without replacement from the population can be represented by an N-vector $s = (s_1, s_2, \ldots, s_N)$' of indicators, where $s_k = 1$ if individual k is selected in the sample, and where $s_k = 0$ otherwise. The expected value of s is equal to $E(s) = \pi$, where $\pi = (\pi_1, \pi_2, \ldots, \pi_N)'$ is the N-vector of first order inclusion probabilities. The sample size is denoted by n. Horvitz and Thompson (1952) show that

$$\bar{y}_{HT} = \frac{1}{N}\sum_{k=1}^{N} \frac{s_k Y_k}{\pi_k} \qquad (3.3.2)$$

is an unbiased estimator of the population mean (3.3.1).
Linear weighting amounts to applying the generalized regression estimator. This estimator is defined by

$$\bar{y}_{GR} = \bar{y}_{HT} + (\bar{X} - \bar{x}_{HT})'b, \qquad (3.3.3)$$

where $\bar{X}$ is the vector of population means of a set of auxiliary variables, $\bar{x}_{HT}$ is the vector of Horvitz-Thompson estimators for the auxiliary variables, and b is a vector of regression coefficients defined by

$$b = \left(\sum_{k=1}^{N} \frac{s_k X_k X_k'}{\pi_k} \right)^{-1} \left(\sum_{k=1}^{N} \frac{s_k X_k Y_k'}{\pi_k} \right). \qquad (3.3.4)$$

In the case of nonresponse, the Horvitz-Thompson estimators $\bar{y}_{HT}$ and $\bar{x}_{HT}$ cannot be used. Furthermore, estimation of b will have to be based on available observations only. We then use the modified generalized regression estimator, see Bethlehem (1988).

The assumption underlying this estimator is that the data are Missing At Random (MAR). This is the case if target variable and response behaviour are not independent, but they are independent given the values of the auxiliary variables. Without use of auxiliary information, estimates of the population characteristics would be biased. For qualitative auxiliary variables, MAR means that observed values of the target variable not necessarily constitute a random subsample of the sampled values, but they are a random subsample of the sampled values within subclasses defined by the values of the auxiliary variables. This implies that there are auxiliary variables conditionally on which the nonresponse is not selective for the target variable.

The propensity score

Before we come to the actual propensity scores, first some context. Again, it is assumed that the aim of the sample survey is to estimate the population mean of the target variable *Y,* see (3.3.1). In case of a telephone survey, only people with a listed number can be contacted. We assume that whether or not an individual has a listed number is the result of some random process, where each individual k has a certain, unknown probability τ_k of having a listed number, for $k = 1, 2, \ldots, N$.
Let T denote an indicator variable, where $T_k = 1$ if individual k has a listed, fixed-line telephone number, and where $T_k = 0$ otherwise. Then $P(T_k = 1) = \tau_k$..

For a telephone survey, only those values Y_k become available for which individual k is selected in the sample ($s_k = 1$) and has a listed phone number ($T_k = 1$). Therefore, the adapted Horvitz-Thompson estimator becomes

$$\bar{y}_{HT}' = \frac{1}{N} \sum_{k=1}^{N} \frac{s_k T_k Y_k}{\pi_k \tau_k}. \qquad (3.3.5)$$

Unfortunately, the listed phone probabilities are unknown. Therefore, we estimate them using the available auxiliary information. We use the method of propensity scores to achieve this. See Rosenbaum and Rubin (1983).

Translated in the current context, the propensity score $\tau(X)$ is the conditional probability that an individual with observed characteristics X has a listed telephone number ($T = 1$):

$$\tau(X) = P(T = 1 \mid X) \qquad (3.3.6)$$

It is assumed that within subpopulations defined by values of the observed characteristics X, all individuals have the same probability of having a listed number. In the context of nonresponse in survey sampling, this assumption is also referred to as Missing At Random (MAR). The propensity score method thus relies on the same assumption as linear weighting.

Often, the propensity score is modelled by means of a logit model:

$$\log\left(\frac{\tau(X_k)}{1-\tau(X_k)}\right) = \alpha_k + \beta_k' X_k + \varepsilon_k \qquad (3.3.7)$$

Other models can be used too, but e.g. Dehija and Wahba (1999) conclude that different models often produce similar results.

The propensity score can be used in two different ways: propensity score weighting and propensity score stratification. With *propensity score weighting*, the estimated propensity scores $\hat{\tau}(X_k)$ are directly used in the adapted Horvitz-Thompson estimator (3.3.5). *Propensity score stratification* is a form of post-stratification where strata are being formed on the basis of the propensity scores.
Suppose the sample is stratified into L strata by means of the estimated propensity score. The poststratification estimator is defined by

$$\bar{y}_{PS} = \sum_{h=1}^{L} \frac{N_h}{N} \bar{y}_h \qquad (3.3.8)$$

where N_h is the number of elements in stratum h and $\bar{y}_h$ is the mean of the available observations in stratum h. Bethlehem (1988) shows that the bias of the post-stratified Horvitz-Thompson estimator can be written as

$$B(\bar{y}_{PS}) = \sum_{h=1}^{L} W_h \frac{C_h(\tau, Y)}{\bar{\tau}_h}, \qquad (3.3.9)$$

where summation takes place of the L strata. The quantity $W_h = N_h / N$ is the relative size of stratum h, $C_h(\tau,Y)$ is the covariance between the values of target variable and listed number probabilities within stratum h, and $\bar{\tau}_h$ is the average of the listed number probabilities in stratum h.

This bias is small if the covariance is small in each stratum and the covariance is small when the variation in listed number probabilities is small. So it makes sense to construct strata in such a way that most variation of these probabilities is between strata and not within strata. Cochran (1968) suggests that as much as five strata may be sufficient to remove a large part of the bias.

Linear weighting combined with the propensity score

There are two ways to combine linear weighting and the propensity score method. The first approach is based on the idea of propensity score weighting and comes down to linear weighting with adjusted inclusion probabilities.
See that linear weighting as described by expression (3.3.3) only produces consistent estimates if the proper inclusion probabilities are used. Availability of data in a telephone survey is determined by both the sampling mechanism and the probability of having a listed number. Therefore, the in π_k in (3.3.4) should be replaced by $\pi_k \tau_k$. Unfortunately, the τ_k are unknown, so they are replaced by their estimates $\hat{\tau}(X_k)$.

The second approach is based on propensity score stratification. The idea is to include an additional variable in the weighting model that accounts for the probability of having a listed number. This additional variable is a categorical variable with each category representing a stratum based on the estimated propensity scores.

3.4. The data and the methods

So, we have a face-to-face survey that, in the nearby future, could become a telephone survey. Besides the missing data due to nonresponse, we would then also have missing data because of the undercoverage of non-listed, fixed line numbers. Like with nonresponse, undercoverage is only a threath to survey statistics if it is selective. Cobben and Bethlehem (2005) show that this is indeed so for POLS 2002. The characteristics of owners of a listed, fixed line number differ from those without such a number, especially with respect to the variables *ethnic group, % of foreigners* and *household type.*

To obtain insight into the consequences of this additional missing data, we apply the methods discussed in section 3.3 to our data sets. First, the response means of bot the face-to-face survey and the telephone survey are compared. The method that is commonly applied to adjust for nonresponse is linear weighting. For the POLS survey, we apply a weighting model that is proposed by Schouten (2004). This model is[2]

$$Age_{15} + House\ value_{14} + Percentage\ foreign_{8} + Ethnic\ group_{7} + Region_{15} + Household\ type_{4} \quad (1)$$

The subscripts denote the number of categories. When applying this model to both surveys, we do not take into account the selectiveness of the telephone survey yet. Therefore, we apply the propensity score method to the telephone survey to adjust for the propbability of telephone ownership. The combination of linear weighting with the propensity score methods is then applied to adjust for nonresponse and undercoverage simultaneaously. The results are discussed in section 4.

4. Results

To analyse the implications of changing a face-to-face survey into a telephone survey, in Table 4.1 we compare the response means and the adjusted estimates when applying the nonresponse adjustment; weighting model (1). A great degree of similarity of telephone estimates and face-to-face survey estimates means that going from a face-to-face survey to a telephone survey does not introduce an additional bias.

The second and third column in Table 4.1 display the unweighted response means for both surveys. There are quite some differences. Respondents that own a listed, fixed-line telephone tend to work less, have a higher education and are more often non-religious than respondents that do not own a listed, fixed-line telephone. However, it maybe that adjusting the for nonresponse handles the selectivity of telephone ownership and reduces these differences. Therefore, we apply weighting model (1) to both surveys. The results can be found Table 4.1 in columns four and five.

We regard the results from weighting the face-to-face survey to be the most unbiased, i.e. the most close to the true population characteristics. When we compare the estimates from the weighted telephone survey to these final estimates, we see that some bias still remains. The weighted estimates tend to adjust in the right direction, but then over- or underestimate the characteristic. Take for instance the percentage of persons that work 12 hours or more. The response mean for the

[2] Originally, the variable *telephone* was included in the model. However, when applying the model to the telephone survey, the variable *telephone* is redundant and thus removed. For reasons of comparison, we decide to also remove this variable when applying the model to the face-to-face survey.

telephone survey is 54.5%. The adjusted face-to-face survey estimate is 56.1%. The adjusted telephone estimate indeed shows an increase but overestimates the face-to-face survey estimate by 0.6%. The differences may seem minor, but 0.6% of 16 million persons still are approximately 100.000 persons.

Table 4.1. Comparison of unweighted and weighted response means for the face-to-face survey and the telephone survey (in %)

Variable	Response mean telephone survey	Response mean face-to-face survey	Weighted estimates telephone survey	Weighted estimates face-to-face survey
Employment				
12 hours or more	54.5	55.3	56.7	56.1
Less than 12 hours	7.4	7.1	6.9	6.7
Unemployed	38.1	37.6	36.4	37.3
Educational level				
Primary	6.7	7.2	5.8	6.1
Junior general secondary	12.2	12.0	11.9	11.9
Pre-vocational	19.5	19.7	19.3	19.9
Senior general secondary	6.9	7.2	7.2	7.4
Secondary vocational	31.0	30.7	31.4	31.1
Higher professional	17.2	16.6	17.6	16.9
University	6.5	6.4	6.8	6.6
Other	0.2	0.2	0.2	0.2
Religion				
None	36.2	37.7	37.8	38.4
Roman-Catholic	35.4	35.6	32.8	32.5
Protestant	22.7	21.4	21.0	20.5
Islamic	1.2	2.6	3.2	3.3
Other	4.5	5.2	5.3	5.4

These differences are remarkable since model (1) incorporates the variables that cause the largest selectivity (*ethnic group*$_7$, *percentage foreign*$_8$ and *region*$_{15}$). The results suggest that answers to the survey questions for persons with a telephone on average are different than those for persons without a telephone, even after taking into account the variables that we dispose of that are correlated with ownership of a telephone.

These results show that we can not just ignore the change of data collection mode from CAPI to CATI. Therefore, we tried to adjust for the selectiveness of the persons that own a listed, fixed-line number by using the propensity score method discussed in section 3.3.

We model the propensity score by means of a logit model:

$$\log\left(\frac{\tau(X_k)}{1-\tau(X_k)}\right) = \alpha_k + \beta_k' X_k + \varepsilon_k \qquad (4.1)$$

Other models can be used too, but e.g. Dehija and Wahba (1999) conclude that different models often produce similar results.

Using the complete POLS 2002 data set, the propensity scores are modelled with the software package Stata. By stepwise excluding insignificant variables, the final model turns out to be

$$\alpha_k + \beta_1 * Percentage\ foreigners_{8,k} + \beta_2 * \text{Region}_{15,k} + \beta_3 * Ethnic\ group_{7,k} +$$
$$+ \beta_4 * Urbanization_{5,k} + \beta_5 * Marital\ \ status_{2,k} + \beta_6 * Household\ \ type_{4,k} +$$
$$+ \beta_7 * House\,value_{14,k} + \beta_8 * Age_{3,k} + \beta_9 * Disability\,insurance_{2,\text{k}} +$$
$$+ \beta_{10} * \text{Social security}_{2,\text{k}} + \varepsilon_k$$

The first subscript of each variable denotes the number of categories. We estimate the model parameters by Maximum Likelihood Estimation. The value of the pseudo R^2 for this model turns out to be 9.1%. This is rather low, which is an indication that there still is a lot of unexplained variance in this model. Based on this model, the propensity scores can be predicted. These predicted scores can be used in various ways. The following techniques are explored:

- *Propensity score weighting*: the listed number probabilities τ_k in the adapted Horvitz-Thompson estimator (3.3.5) are replaced by their estimates $\tau(X_k)$ from the logit model.
- *Propensity score stratification*: This is a form of post-stratification where strata are being formed on the basis of the propensity scores. Cochran (1968) suggests that as much as five strata may be sufficient to remove a large part of the bias.

The results from these two techniques are displayed in Table 4.2.

Table 4.2. Estimates for the telephone survey data based on propensity score weighting and –stratification.

Variable	Response mean telephone survey	Response mean full sample	Propensity score weighting	Propensity score stratification
Employment				
12 hours or more	54.5	55.3	53.7	54.9
Less than 12 hours	7.4	7.1	7.4	7.4
Unemployed	38.1	37.6	38.8	37.7
Educational level				
Primary	6.7	7.2	6.5	7.1
Junior general secondary	12.2	12.0	12.2	12.2
Pre-vocational	19.5	19.7	19.8	19.1
Senior general secondary	6.9	7.2	6.8	6.9
Secondary vocational	31.0	30.7	31.1	30.9
Higher professional	17.2	16.6	17.2	17.1
University	6.5	6.4	6.4	6.5
Other	0.2	0.2	0.2	0.2
Religion				
None	36.2	37.7	35.0	36.9
Roman-Catholic	35.4	35.6	36.6	33.8
Protestant	22.7	21.4	23.5	21.7
Islamic	1.2	2.6	0.6	2.5
Other	4.5	5.2	4.2	5.1

To see how the techniques perform, the results from propensity score weighting and –stratification are compared to the response mean from the full sample. This technique is, after all, meant to adjust for the undercoverage and not for nonresponse. Column two and three display the response means for the telephone resp. face-to-face survey. In column four and five, the results from propensity score weighting resp. –stratifciation are shown.

Propensity score weighting does not seem to perform very well. Many values of estimates shift in opposite directions when compared to the response mean of the full sample. The results for the variable *Religion* are remarkably bad, adjusting most categories in the wrong direction. For instance, the percentage of non-religious persons in the response to the full sample is 37.7%, compared to a

response mean of 36.2% for the telephone survey. In stead of adjusting the underestimation, propensity score weighting increases the difference between the two response means with an additional 1.2%. This might be caused by the fact that estimates become highly dependent on the model used for the propensity scores.

Propensity score stratification leads to estimates that are shifted in the right direction. However, estimates are often still far away from the full sample response mean. So, stratification based on just propensity scores is not able to completely correct for the undercoverage bias.

So far, we established that propensity score stratification succeeds best in adjusting the response mean for the telephone survey to the response means for the full sample. Now we can proceed with adjusting the final estimates, i.e. including an adjustment for the nonresponse bias as well. We do so by a mixture of linear weighting (nonresponse adjustment) and the propensity method (adjustment for undercoverage). In section 3.3 we discuss two ways to combine these two methods. The results are displayed in Table 4.3.

Table 4.3. Results for the adjustment of the final estimates for both undercoverage and nonresponse.

Variable	Linear weighting the face-to-face sample	Linear weighting with 5 propensity score strata	Linear weighting with adjusted inclusion probabilities
Employment			
12 hours or more	56.1	56.3	56.2
Less than 12 hours	6.7	7.0	7.0
Unemployed	37.3	36.7	36.8
Educational level			
Primary	6.1	5.8	5.8
Junior general secondary	11.9	11.9	11.9
Pre-vocational	19.9	19.2	19.2
Senior general secondary	7.4	7.2	7.2
Secondary vocational	31.1	31.3	31.2
Higher professional	16.9	17.6	17.5
University	6.6	6.8	6.8
Other	0.2	0.2	0.2
Religion			
None	38.4	37.8	37.8
Roman-Catholic	32.5	32.7	32.7
Protestant	20.5	20.9	21.0
Islamic	3.3	3.2	3.3
Other	5.4	5.3	5.4

The results are compared to the estimates from linear weighting the full sample, in the second column, since these estimates should be closest to the true value.
The third and fourth column display the results from adjusting the telephone sample for nonresponse and undercoverage simultaneously with resp. linear weighting with an additional propensity score stratification and linear weighting with adjusted inclusion probabilities.

Including a categorical propensity score variable in the model seems to pay. The best adjustment technique to reduce the bias caused by telephone interviewing for this case, appears to be a combination of linear weighting in which the true inclusion probabilities are estimated by means of logit model for listed number propensity scores.

5. Conclusion

In this paper we consider the influence of the data collection mode on errors related to undercoverage and nonresponse, and compare various techniques that aim at adjusting for the bias caused by these errors. These techniques are partly based on linear weighting, and partly on using propensity scores.

The aim of our research is to answer the following question:

What is the effect on the quality of estimates of population characteristics, when going from a face-to-face survey to a telephone survey and, consequently, c*an we adjust for this effect?*

We explore to what extent adjustment techniques can reduce the bias caused by telephone interviewing. First, the telephone sample is adjusted for undercoverage of persons without a listed number. Two methods are used: Propensity score weighting and propensity score stratification. No nonresponse bias is considered yet and the results are compared to the unweighted response mean of the full sample. Second, the nonresponse bias is taken into account as well. Two combinations of linear weighting and the propensity score method are considered.

The best adjustment technique to reduce the bias caused by telephone interviewing for this case, appears to be a combination of linear weighting in which the true inclusion probabilities are estimated by means of logit model for listed number propensity scores. However, the ultimate estimates are still biased. There seem tp be a relationship between telephone ownership and the questions in the survey that we cannot explain with the auxiliary variables that we dispose of. We lack variables that are sufficiently informative to explain telephone ownership. This leads to biased estimates, especially for those survey questions that are related to education and income. It is rather ironic; the variables that we try to estimate are actually the variables that we would like to use in our model.

6. References

Bethlehem, J. (1988) *'Reduction of nonresponse bias through regression estimation'* Journal of Official Statistics, Vol. 4, No. 3, p. 251-260.

Bethlehem, J. (1999) '*Cross-sectional Research'* In: H.J. Adèr and G.J. Mellenbergh, Research Methodology in the Social, Behaviuoural & Life Science, Sage Publications, London, p.110-142.

Bethlehem, J. and Schouten, B. (2003) *'Nonresponse analysis of the integrated survey on living conditions (POLS)*', Discussion paper 04004, Statistics Netherlands

.

Cobben, F. and Bethlehem, J. (2005) '*Adjusting undercoverage and nonresponse bias in telephone surveys'* Discussion paper 05006, Statistics Netherlands.

Cochran, W. (1968) '*The effectiveness of adjustment by subclassification in removing bias in observational studies*' Biometrics, Vol. 24, p. 205 – 213.

Dehjia, R. and Wahba, S. (1999) *'Causal effects in non-experimental studies: re-evluating the evaluation of training programs'* JASA, p. 1053 – 1062.

Horvitz, D.G and Thompson, D.J. (1952) *'A generalization of sampling without replacement from a finite universe'* Journal of the American Statistical Association, 47, p. 663-685.

Rosenbaum, P.R. and Rubin, D.B. (1983) 'The central role of the propensity score in observational studies for causal effects' Biometrika, 70, p. 41-50.

Rosenbaum, P.R. and Rubin, D.B. (1984) *'Reducing bias in observational studies using subclassification on the propensity score'* Journal of the American Statistical Association, 79 (387), p. 516-524.

Schouten, B. (2004) '*Adjustment bias in the integrated survey on living conditions (POLS) 1998*' Discussion paper 04001, Statistics Netherlands

Experiences of usage of Blaise in CATI surveys

Stefania Macchia & Manuela Murgia (ISTAT, Italy)

1. Introduction

The paper shortly describes the positive results obtained through the use of Blaise in Istat for CATI surveys, which lead to the definition of a "standard" strategy that has already been largely adopted for three years. This strategy is based on the use of Blaise and on its interaction with other software packages, in order to provide solutions for those requirements common to all CATI surveys. Both standardised and not standardised solutions implemented in the software procedure will be described, although the last ones would more efficiently meet the CATI surveys' needs if they were directly provided by Blaise.

2. The use of Blaise in Istat for CATI surveys

A lot of surveys in Istat are carried out with CATI technique and for a great number of them a strategy has been defined aimed at improving the quality of data. This strategy consists in relying on private companies only for the call centre and the selection of interviewers and in providing them with the entire software procedure to be used for the data capturing phase, which is based on the Blaise system.

The majority of surveys carried out with this strategy regards households and individuals, but this is only due to the fact that they are often characterised by very complex questionnaires in terms of number of questions and consistency rules, so there is a greater interest in developing in house the software procedure.

As it can be seen in the following table, a survey on enterprises (*Water system survey*) has been treated too, which has a very huge number of variables (30,000), even if this number is only a maximum value defined considering the maximum numbers of different entities which could belong to each respondent/form (any respondent/form reaches this maximum values for all the entities).

Table 1- Surveys carried out with the strategy based on Blaise

Surveys	*Number of variables of the electronic questionnaire*		*Number of interviews*
Sample births survey *Year 2001*	*Long*	677	16,597
	Short	170	33,838
Sample births survey *Year 2004*	*Long*	707	15,642
	Short	199	33,515
University-to-work transition survey and perspectives *Year 2004*		218	25,510
Upper secondary school graduates survey *Year 2004*		315	20,408
Water System Surveys (preliminary survey) *Year 2006*		30,000	1,320
Violence against women survey *(in progress)*		2,774	25,000 (to be concluded)

This strategy turned out to be successful as it can be deduced by different aspects which have been evaluated. One of these is the response rate, which is surely high, considering the length of the interviews (table 2).

Table 2- Response rates and interviews' lengths of the surveys carried out with the strategy based on Blaise

Surveys	*Interviews' length*		*Response rates*
Sample births survey Year 2001	*Long*	12'00''	92.6%
	Short	5'00''	93.2%
Sample births survey Year 2004	*Long*	13'48''	94.7%
	Short	5'43''	96.8%
University-to-work transition survey and perspectives Year 2004		10' 56''	95.8%
Upper secondary school graduates survey Year 2004		13' 20''	94.7%
Water System Surveys (preliminary survey) Year 2006		9'03''	99.8%
Violence against women survey (in progress)		26'54''	72.4%

Having adopted this strategy with success for more than three years, made us cope with different requirements expressed by each survey that we solved either exploiting the potentialities of Blaise or integrating the software procedure with other software packages.

Some of these solutions turned out to constitute standard tools in our procedures, some others needed to be used only for particular purposes, so we did not develop standard functions, but we identified the software packages to be used for them, while for other requirements, we would like to have at disposal some specific functions included in the Blaise system. The last ones are those requirements which, even if are not common to all the surveys, are typical of a well identified subset of them and would require a big programming effort to be integrated in a software procedure.

3. Standard tools included in the used software procedure

The standard tools we implemented in the software procedure of the described strategy are those which fulfil the requirements which are common to all the CATI surveys.

They mainly concern:

- the production of a certain number of reports to monitor daily the interviewing phase both from the qualitative and the quantitative point of view,
- a standard procedure for the electronic transmission of data (interviews and reports) from the external companies to Istat, which guarantees all the security requirements,
- a function which allows the display on the screen, during all the interview, of the called telephone number and name of the person.

Considering the first item of the list, the set of reports have been developed using **Visual Basic**, based on an Access database, which produces Excel files, that can be easily managed by statisticians.

A lot of these reports contain indicators which allow to examine more in details some elements regarding the CATI contact results already provided by Blaise through some video screens displayed by the BTMana. Some of them are aimed at monitoring the productivity of interviewers, so they are calculated per interviewer (i.e. refusals, definitive interruptions, etc), others reports are aimed at detecting possible problems of the survey questionnaires (i.e. definitive interruptions per questionnaire section) and other are aimed at optimising the proceedings of the interviewing phase (i.e. future appointments per time slices, future appointments per time slices per interviewers when the "route back to interviewer" criterion has been selected).

Other reports regard information which are stored in Blaise, but not displayed and are aimed at the already mentioned scopes. The most important are listed:
- refusal causes and definitive interruptions causes (total and per interviewer)
- interview's length (total and per interviewer)
- last contact results distribution per type and time slice
- last contact results distribution per type and per day of week
- still available sample units classified per stratum
- matrix of movements of interviewed units from a stratum to another

Then, there are other sets of reports, regarding indicators produced through some algorithms calculated on data stored in Blaise:
- rates in total and per interviewer (response rate, refusal rate, definitive interruption rate)
- no more available sample units (because the maximum number of calls has been reached).

Some examples of reports are shown in the Appendix.

The second standard tool included in the software procedure for CATI surveys concerns the electronic transmission of data from private companies, in charge of interviewing, to Istat. The transmission is made daily, so that the Istat survey experts can monitor the interviewing activity and immediately put in practice possible correctives if something does not work well.

Three datasets are managed by this procedure:
- an ASCII file containing the Blaise "*.bdb*"
- the Blaise *".bth*."
- the complete set of reports "*.xls"*.

From the technical point of view, each day, when the interviewing activity is over, an automatic procedure, installed on a pc of the external company, downloads the Blaise *".bdb"* in ASCII and includes all the above mentioned files in a *".zip"* file.

Then, the private company connects to an Istat web site and, by an user_id, accedes to the *"send"* function. The transmission is based on a *"secure"* protocol (HTTPS) and puts data on an Istat server (outside the firewall) devoted to data collection. An automatic procedure transfers data on another server inside Istat firewall, where the responsible of the survey can accede (with his proper user_id) to get data.

When data are transmitted, two e-mails are automatically sent, to the company and to the Istat survey experts, so that each of them is informed of the result of this activity.

The last tool implemented for all the CATI surveys carried out using this strategy is a function which allows to show on the PC screen of the interviewers the called telephone number and the name of the respondent, during all the interview. As a

matter of fact, Blaise automatically displays these data only in the *"dial screen"*, so that, if there is a problem during the interview (i.e. if the telephone line goes off or if the interviewer wants to take note of the telephone number for any reason), the interviewer could not read this information. Technically speaking, for this purpose, a *.dll* was created with *DELPHI.*

4. Tools implemented for requirements typical of certain surveys

Other requirements are proper of certain surveys. For this kind of subjects, as already said, we did not develop standard procedures, but we identified the software tools to be used.

For instance, for two of the surveys on individuals we carried out with the above mentioned strategy ("University-to-work transition survey and perspectives" and "Upper secondary school graduates survey"), we performed the coding of Occupation during the interview. It was a particularly delicate activity, since the Occupation was the central variable of the survey. We used the Blaise assisted coding function, building a very rich *coding dictionary* (list of descriptions with correspondent codes) and we monitored the performances of interviewers on this subject through the use of control charts developed with the SAS QC software. In more details:

- for the assisted coding phase:
  - we built a dictionary of 16,300 records, each containing the Occupation description and the associated code;
  - the coding during the interview was done using the trigram search method;
- for the monitoring phase:
  - we built an application with the QC module of the SAS software;
  - the application read the database and the history file;
  - every week it produced graphical trend - on average and for each interviewer- of three main indicators – : Quality of Coding (complete codes assigned over the total number of codes to be assigned); Time for Coding; Response rate;
  - interviewers that showed systematic out of control values for one or more indicators were re-trained and monitored till their trend was close to the average one.

The entire procedure can be easily adapted to each survey that requires the assisted coding of Occupation.

5. Non standardised solutions for recurrent requirements in many surveys and that could be provided by Blaise

In this paragraph we will describe the solutions we have adopted to meet a certain number of requirements that, although are not common to all the CATI surveys carried out in Istat with the above mentioned strategy, are recurrent in a lot them. The implementation of some of these solutions required us a big effort in terms of time and resources and the solutions we found were, generally, not immediately portable to other surveys. This big effort as well as the non portability of the results might be overcome if these requirements were met directly by functions integrated in the Blaise system.

5.1. How to assign a contact result to household's members

In household surveys, it is very frequent to extract a sample of individuals from a sample of households. The sample of individuals is determined during the interviewing phase according to eligibility criteria. For example, in the "Labour

Force" survey for each household, the eligible members are those aged more than 14; in the "Violence against women" survey only female individuals aged from 16 to 70 can be interviewed.

In these cases, to obtain a complete interview, it is necessary to contact more than one member of the household and to assign a contact result both to those who are eligible and have completed the interview and to those who are not eligible or do not wish or cannot co-operate (example: the extracted eligible member has a serious illness or refuses to provide answers or he/she is absent for all the survey period, etc.)

To meet this requirement we have developed a function that, in the course of the interview, without abandoning the form, has to assign a contact result to the extracted member and then to select the next eligible one. The implementation of this solution was very difficult because quite often its rules were in contrast with the logic of the Blaise scheduler. This happened each time the treatment associated with the contact result of the individual was different from the one associated with the contact result of the household/form. The following example tries to clarify this point.

Example: Let us suppose the household has two eligible members. The first refuses to cooperate and the next completes the interview. The treatment that Blaise would associate to the first individual (and therefore to the form) would be of "No Answer" and, therefore, the household would not be contacted anymore; what we need instead was to make the household available for the next contact in order to get the interview.

It would be therefore quite important that the Blaise scheduler could manage two kinds of dial results: one for the household/form and one for each member of it. The solution we implemented was not only very time consuming, as previously said, but also not so ease to adapt to the next survey that will have the same requirement.

5.2. How to make the structure of the datamodel of a complex survey unit more flexible

The already mentioned structure of the household (composed by a certain number of individuals to be interviewed) implies another problem: as Blaise organises the household datamodel on a single record and the number of individuals of the households is not a constant, it is necessary to define a maximum value for this number. As a consequence of this, we have very long records with a high number of empty variables.

This problem is even greater for surveys on enterprises, when it is necessary to interview the "mother enterprise" and its local units (or function units), because the maximum number of these units of lower hierarchical level is even greater than the maximum number of individuals of a household.

A possible solution could be of defining a main datamodel corresponding to the unit of the superior hierarchical level (i.e. the household) which calls other datamodels corresponding to the lower hierarchical units (individuals), so as not only to assign contact results to each of them (see point 4.1), but also to write a single record for each of them.
We are trying to implement such a structure for the second part of the "Waters system survey", managing the interaction between different datamodels with **Visual Basic**.

5.3. How to make the managing of parallel blocks more flexible

The great majority of the CATI surveys uses parallel blocks to assign a dial result and, to this aim, an easier Blaise management of them would have better satisfied some surveys' needs; the main of them are described in the following.

Some surveys use more types of dial results than the eight ones provided by Blaise and for this reason different dial results can be associated to same treatment. Besides, it is quite common to associate the same parallel block/treatment to a dial result that can be keyed both from the dial screen and from the datamodel. This normal situation becomes difficult to manage under these circumstances:
the parallel block has to perform different actions according to where the dial result is assigned (from the dial screen or from the datamodel);
the data model includes more than one parallel blocks associated to the same treatment.

In this situation, once a dial result is keyed from the dial screen it can potentially be linked to any of the many parallel blocks that have the same treatment.

In our experience, we have to face this for the appointment treatment that was linked to three dial results. One of them was "answering machine with message" - the message said the time or period the family was at home and the operator has to fix an appointment according to it-; this dial result was present in the dial screen. At the same time, the datamodel allowed also to take other dial results associated to the appointment treatment: "appointment with the family" and "appointment with the extracted member". Besides, the parallel blocks associated to the three cases had to perform different sets of rules.

To overcome the problem that it is not possible to link the "answering machine with message" result to its specific parallel block (because all the parallel blocks have the same treatment), we had to use only one parallel block that contained all the rules relative to all the three cases and put all these actions under a very complex system of "*if-and-then"* conditions that allowed to perform the right rules according to the selected dial result.

Probably this could be avoided if the Blaise system would have allowed to link each dial result not to a treatment but to the a customised name of the parallel block associated to the treatment.

Another very frequent requirement we have to meet is to assign different dial results according to when the phone contact is closed. For example: for all surveys, the dial result "refuse" can be selected only until the very beginning of the interview (when administering some "screening" questions), while, when questions concerning the proper survey subject start, the dial result is "definitive interruption" in case the respondent does not wish to co-operate anymore,. In the first case we link the dial result to the treatment "No Answer" while in the second case to the treatment "Other". As we do not want to make the operator choose between them (as we noticed there are frequently misunderstandings) we make possible to key the right result making appear only one of the two parallel blocks a time, depending on which part of the questionnaire is administered.

To do this we have to put under "*if-and-then"* conditions not only the two parallel blocks involved but also the entire set of parallel blocks used throughout the questionnaire. This implies that we have to find a suitable condition for the calling of each parallel block, also for those that would normally not need it, like the "No answer".

Clearly, in both the situations described, the complex system of "*if-and-then*" conditions is peculiar to each CATI survey because it is generally based on values of the survey questions. It is therefore almost impossible to create standardised set of rules in that they have to be customised to the survey questionnaire.

A parametric management of the parallel blocks in Blaise could probably represent an easier way to meet this requirements.

5.4. How to modify the Blaise database

The need of a change in the variables' values on the Blaise database while the interviewing phase is in progress is very common to all the CATI surveys we manage.

These changes generally concern the input variables like the telephone number, the household address, etc. and, in general, all variables whose values are known in advance, but have changed in the period among the survey waves. The telephone number is the variable that more frequently than the others changes. This is because, Istat is used to send an announcement letter before the beginning of the survey, through which it informs the sample unit about a future phone call regarding the survey and asks for its co-operation. The letter also contains a toll free number the sample units can call to ask for information or to communicate possible changes; the most frequent change is about the units' phone number. In general, the information about this change is got before the survey starts, but quite often this is not possible and the communication arrives when the survey is in progress. To make this change a long procedure is required (the same is valid for any variables) :

a) download the database;
b) make this changes with a data management program (SAS or Manipula);
c) reload the database.

Obviously, to do all these operations it is necessary to stop the fieldwork, especially for those surveys whose crew size is from morning to evening. The consequences of this are quite easily understandable.

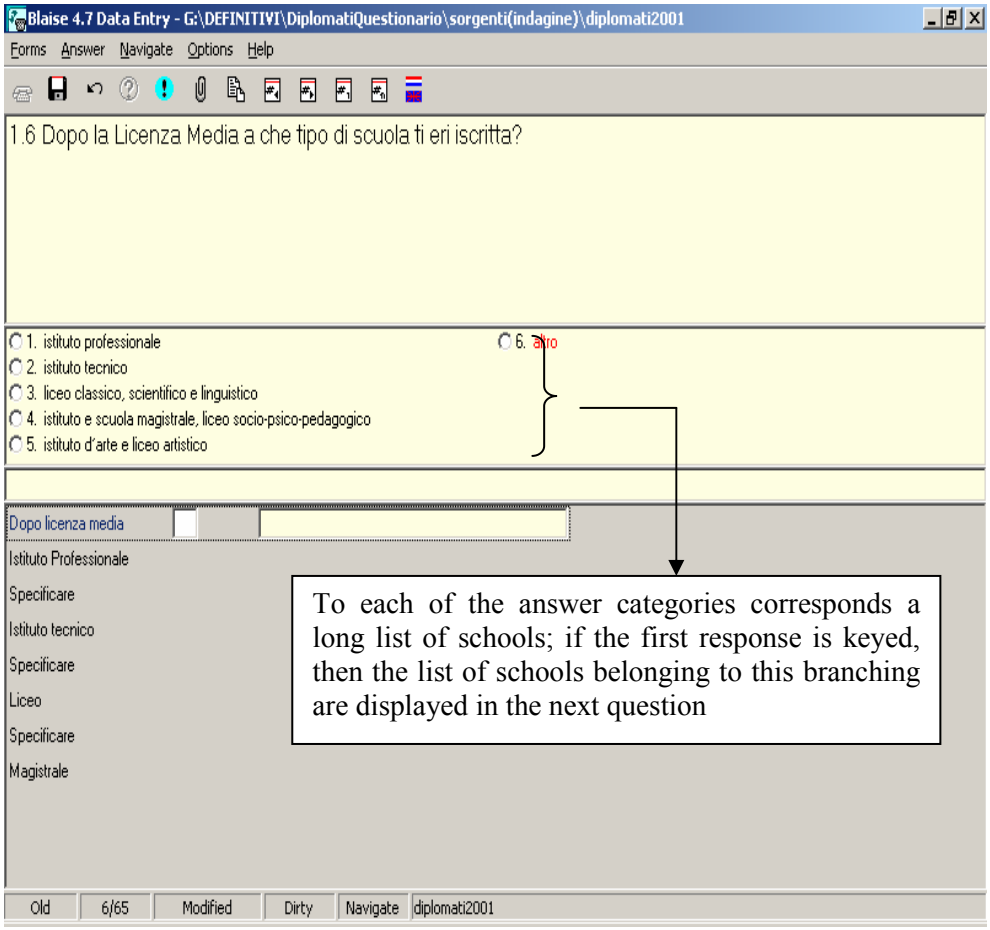
It could be hypothesised that a "browse-editing" function for the Blaise database would definitely speed up and simplify these operations.

5.5. The implementation of the item rotation

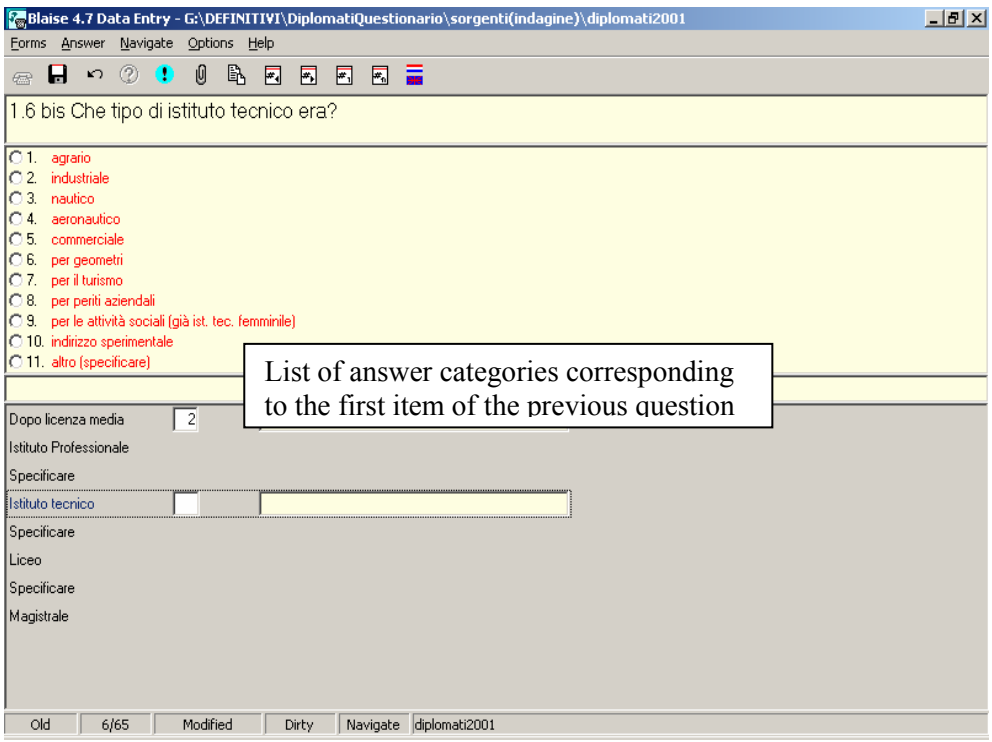
It is well known that a well designed CATI questionnaire must not contain questions with a long list of answer categories. Despite this general rule, it happens, sometimes, that a survey question does not respect this feature due to various causes.

In this case to simplify the interviewer's job and to avoid biased results (i.e. answers concentrating on the first or last answer items) we implement different and not standardised solutions depending on the survey questions. When it is possible, we split the question in more than one in order to reach the answer step by step: this can be done only when the interview's length is not to long (see *Example 1a and 1b*).

Example 1a How to split one question in more than one to avoid a long answering category list

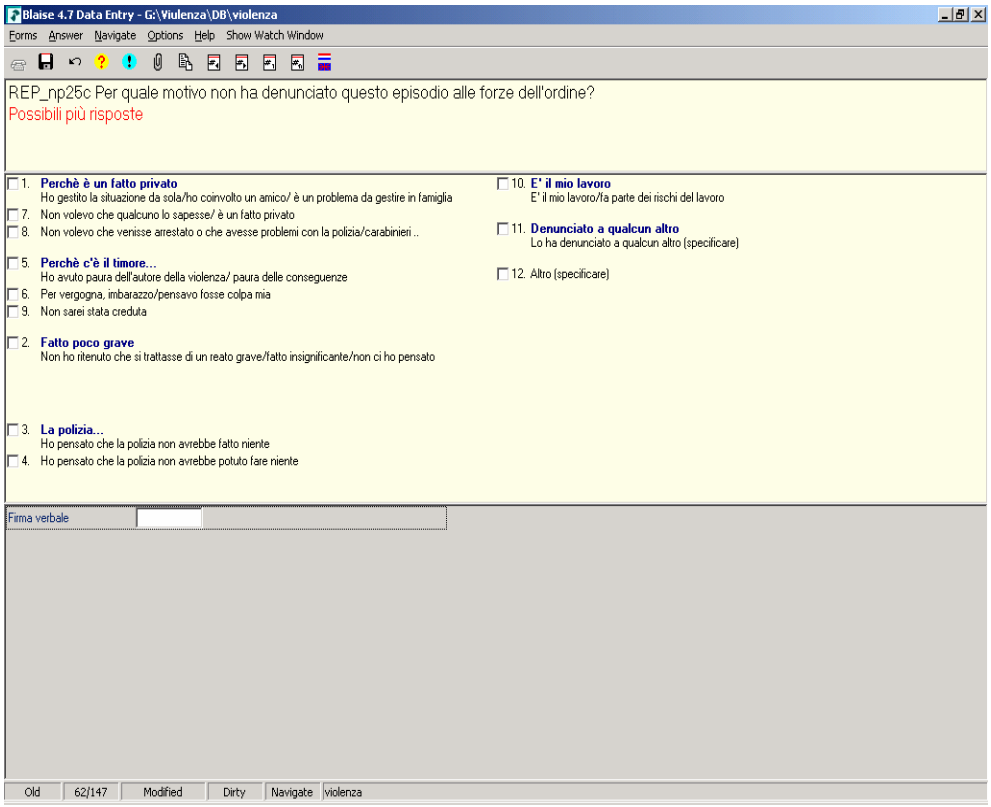


Example 1b How to split one question in more than one to avoid a long answering category list



Another solution is to design the graphical layout of the question items in such a way that the answer categories are grouped according to their content (see *Example 2*).

Example 2 Answer categories grouped according to their content



All of these solutions make the implementation phase heavy as well as the use of the existing Blaise source program for the rotation.

We think that Blaise could provide a function that can manage the items rotation without the need of defining the answer categories as variables and that works in the same simple way of the instruction used for the management of multi-response questions.

5.6. Assigning a code to the dial result when the max call number is reached

Generally speaking, sample units cannot be included in the daybatch file if they have reached the maximum number of calls specified in the scheduling parameters. There is no evidence of this in their dial results, because only the last contact is registered.

As previously said, we produce reports on contact results which indicate also how many units reach the maximum number of calls. This is a very important information, because, in case of a too quick growth of this kind of units, it is necessary to understand the reasons why. It could depend from:

- a wrong setting of the scheduler parameter;
- a not well planned survey timing;
- a not updated list of sample unit;
- a wrong management of the appointments;
- etc.

Once the causes are clear, then right actions are taken to improve the situation.

To report the amount of units that have reached the maximum number of calls we have implemented a function which assigns a certain dial result considering in it

different variables: the last dial result, the number of calls and, in case of appointments, the type of appointment and the date when it was last fixed. Apart from the difficulties we met in determining this rule and the necessity to repeat this set of conditions each time a report asks for it, we are not quite sure whether or not it works properly, since the algorithm that determines the composition of the daybatch file is not well known.

Therefore, the best solution would be that Blaise assigned a code to the dial result for these cases. This should be a "not definitive" dial result, since it could change when scheduling parameters are modified.

5.7. Automatic substitution among sample units

Sometimes for a CATI survey it is not possible to use an over-sampling rate to determine the dimension of the sample size suitable to get the interviews quota. This generally happens when it is the very first survey occasion or when there is a change in the quality of the unit list that could make the over-sampling rate different from the past experiences. In all these cases the sample is divided in two sets: the base sample units set and the substitution units set. Shortly, the survey starts with the base sample units and sub-set of substitution units are added periodically and in accordance to the number of no more contactable base units, until the planned amount of interviews is reached.

To meet this requirement we implemented a software procedure that:
a) individuates the base units to be substituted;
b) assigns a dial result for those units who are not eligible anymore since they reached the max number of calls;
c) determines the relative substitution units;
d) updates the Blaise database.

This procedure is conceptually the same among the various CATI surveys, but its content varies since it depends on the substitution criteria. Probably the same result would be easily obtained if Blaise contained a procedure that loads a database containing both sets of base and substitution units and, on the basis of some parameters, automatically makes the substitution among the sample units.

6. References

Blaise for Windows Developer's Guide 4.5

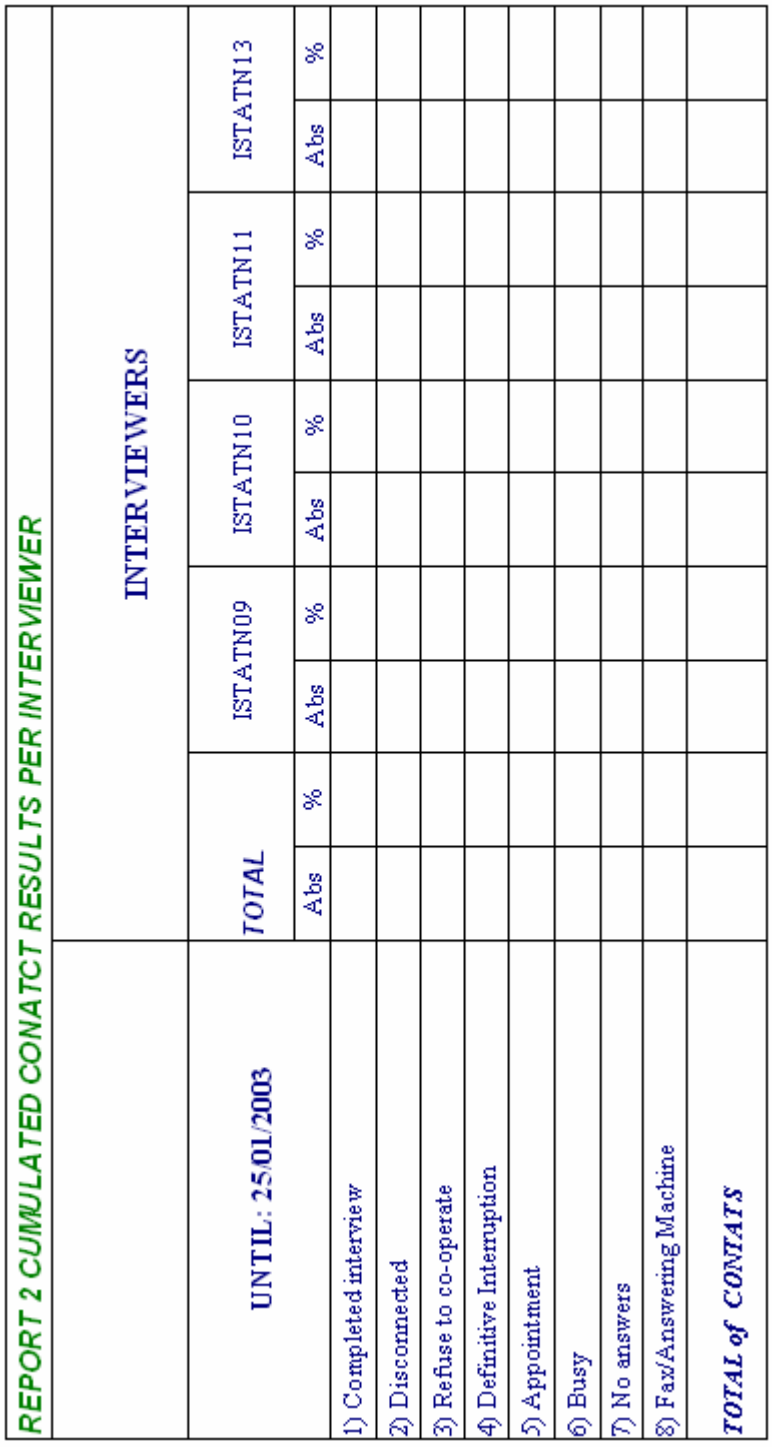
Macchia S. and Murgia M. (2004). "The *in-house* strategy adopted in Istat for CATI surveys".Q2004 European Conference on Quality and Methodology in Official Statistics. Mainz, 24-26 May.

Ryan T. P. (2000*)* Statistical methods for quality improvement (Second edition), Wiley, New York.

Degortes M., Macchia S. and Murgia M. (2003) "The usage of Blaise in an integrated application for the Birth Sample Survey in CATI mode" Ibuc 2003, Copenhagen, 21-23 May.

Appendix – Examples of reports for CATI surveys

Example 1. Example of report for the productivity of interviewers



REPORT 2 CUMULATED CONATCT RESULTS PER INTERVIEWER															
	INTERVIEWERS														
UNTIL: 25/01/2003	TOTAL		ISTATN09		ISTATN10		ISTATN11		ISTATN13						
	Abs	%	Abs	%	Abs	%	Abs	%	Abs	%					
1) Completed interview															
2) Disconnected															
3) Refuse to co-operate															
4) Definitive Interruption															
5) Appointment															
6) Busy															
7) No answers															
8) Fax/Answering Machine															
TOTAL of CONTATS															

Example 2. Example of report to monitor possible problems during the interview phase

REPORT 14 – DEFINITIVE INTERRUPTIONS PER QUESTION – Cumulated and weekly Results -

LAST ANSWERED QUESTION	Until : 27/02/2006		Week from: 27/02/2006 to: 27/02/2006	
	DEFINITIVE INTERRUPTIONS		DEFINITIVE INTERRUPTIONS	
	Abs	%	Abs	%
Qb19.disturbi[1].disturbi				
Qb1				
Qb12				
Qb13				
.....				
TOTAL				

Example 3. Report to monitor the management of appointments when the "Route back to interviewer" option is activated

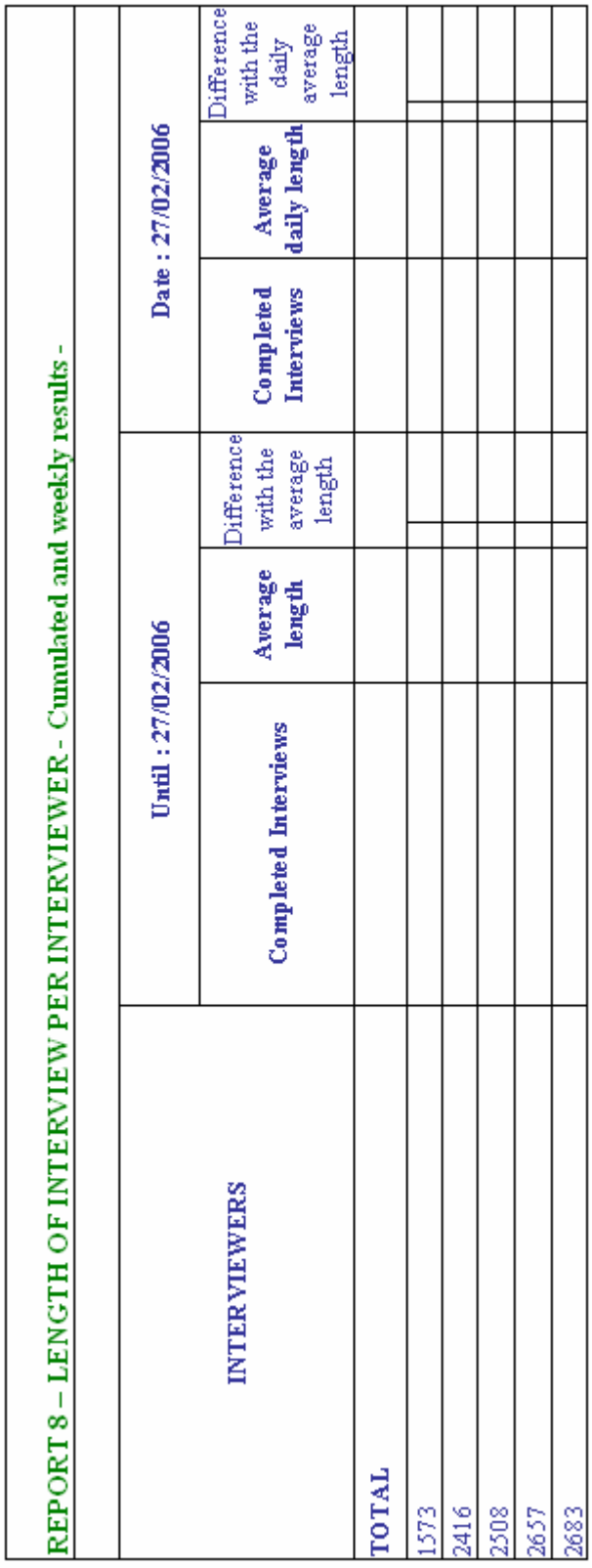
Interviewer code: 897

REPORT 17 ACTIVE APPOINTMENTS PER INTERVIEWER – Cumulated results

Date: 27/02/06

DATE	APPOINTEMNT TYPE							TOTAL	TIME SCLICES										
	Exact time and exacted date	Period and day part	Week day and exacted time	Period an no time	Weekday and no time	Only day part	No date no time		09:00-10:59	11:00-12:59	13:00-14:59	15:00-16:59	17:00-17:59	18:00-18:59	19:00-19:59	20:00-20:59	21:00-22:59	Total
No date																		
20060227																		
20060228																		
20060301																		
.....																		

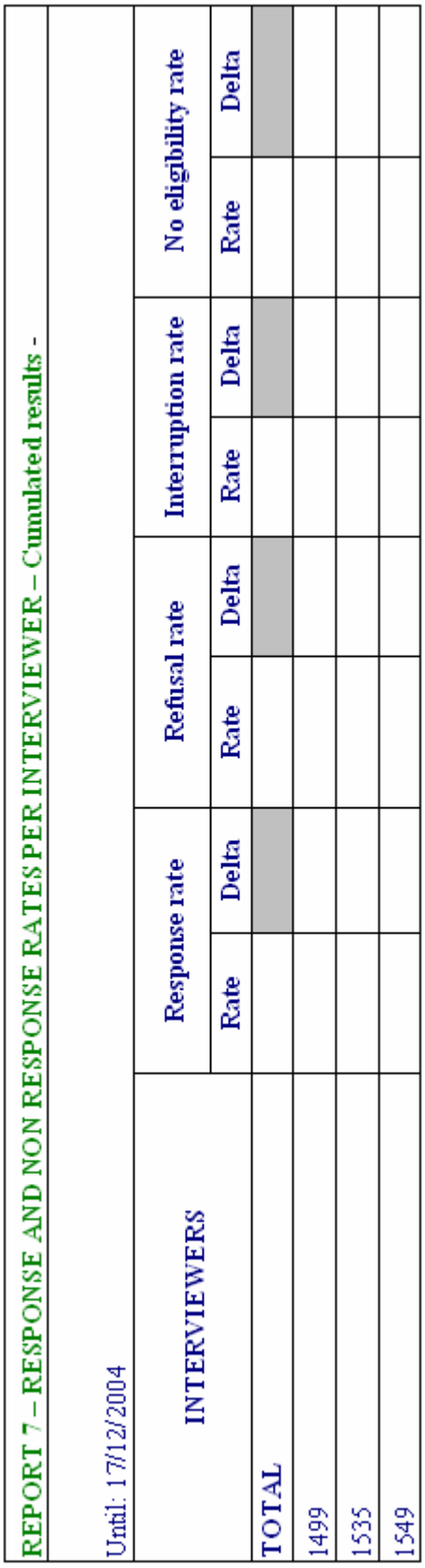
Example 4. Report to monitor the interview's length



REPORT 8 – LENGTH OF INTERVIEW PER INTERVIEWER - Cumulated and weekly results -

INTERVIEWERS	Until : 27/02/2006			Date : 27/02/2006		
	Completed Interviews	Average length	Difference with the average length	Completed Interviews	Average daily length	Difference with the daily average length
TOTAL						
1573						
2416						
2508						
2657						
2683						

Example 5. Report to monitor response and non response rates for each interviewer



REPORT 7 – RESPONSE AND NON RESPONSE RATES PER INTERVIEWER – Cumulated results -

Until: 17/12/2004

INTERVIEWERS	Response rate		Refusal rate		Interruption rate		No eligibility rate	
	Rate	Delta	Rate	Delta	Rate	Delta	Rate	Delta
TOTAL								
1499								
1535								
1549								

NASS CASIC Survey Administration System

Everett Olbert & Roger Schou (National Agricultural Statistics Service, USA)

1. Background

Since 1998, when the National Agricultural Statistics Service (NASS) opened its first Data Collection Center (DCC) specifically designed for consolidated calling, demand for DCC services has increased dramatically. Over the past several years, NASS has seen much of its data collection responsibilities fall upon its DCCs. With this increased burden and responsibility being placed on NASS's DCCs, one of the highest priorities of NASS has been to develop and improve survey administration capabilities for its DCCs concerning costs, response rates, case management, and interviewer performance. In response to that priority, the CASIC Section in Washington, D.C., is developing a Survey Administration System that will bring data together from the Blaise CATI history file, survey response data, NASS Blaise Interviewer database, and the National Association of State Departments of Agriculture (NASDA) cost data in order to give NASS DCC coordinators and NASDA supervisors the information they need on a daily basis to manage and evaluate their data collection efforts more efficiently and effectively.

1.1. Data Collection Center, Serviced State Relationship[1]

Over the last several years, the National Agricultural Statistics Service (NASS) has begun to have difficulty hiring CATI interviewers in several of our field offices. The wages that we can offer to prospective interviewers in cities like Denver, Colorado; Minneapolis, Minnesota; Lincoln, Nebraska; and Nashville, Tennessee are not competitive. As a result, the use of CATI has dwindled to virtually nothing in those and other field offices. On the other hand, in offices located in smaller cities with lower costs of living, hiring interviewers is not a problem. Therefore, NASS has opened five Data Collection Centers in Louisville, Kentucky; Little Rock, Arkansas; Oklahoma City, Oklahoma; Cheyenne, Wyoming; and Helena, Montana. The CATI workstation capacity of each center is shown below.

DCC Capacity	
Arkansas	25
Kentucky	29
Montana	34
Oklahoma	16
Wyoming	24

NASS will open another official Data Collection Center in Richmond, Virginia in March of 2006.

In a typical NASS field office, the CATI interviewers use the regular office staff's computers. There may be a few older computers dedicated to daytime interviewers. During the evening hours, interviewers are scattered throughout an office, sharing the desk space and computers of the professional staff. The interviewers are part-time employees and only work when the state is conducting larger surveys. In contrast to the typical state office, our official Data Collection Centers have and will have computers dedicated to CATI data collection. They are all in one general area, making supervision much easier. Although the interviewers are still employed on a part-time basis, they have steadier work because they are not dependent on one state's survey load. They are conducting surveys for multiple states. Since the workload is steadier, the level of expertise of the interviewers increases yielding higher quality data.

The field office with little or no CATI capacity is known as the Serviced State (SS). Once the SS analyzes their survey sample, they transfer cases selected for CATI to the DCC. The DCC collects the data for these cases and returns them to the SS. Staff in the SS then interactively edit the cases using Blaise. By returning the case to the SS for editing, we assure that all data in the state is edited in the same manner.

2. CASIC Survey Administration System Overview

The CASIC Survey Administration System is a Microsoft Access database in which data from several sources is populated and used to generate various charts and reports. The sources of data for the system include the Blaise CATI History file, the NASS Blaise Interviewer database, the Blaise survey dataset office use fields, and in the near future, actual cost data from the NASDA electronic time sheet system.

2.1. CATI History File

The CATI History file is an ascii comma separated text file in which every case treatment by an interviewer is tracked. Fifteen default fields are available in the file with the ability to add more by the user if desired. The fifteen default fields include the Primary Key, the Internal Key, Dial Date, Dial Time, Call Number, Dial Number, Who Phoned, Entry Priority, Dial Result, Exit Priority, Dial Line Number, Appointment Type, Exit Time,
Dial Time in Seconds, and Interview Time in Seconds.

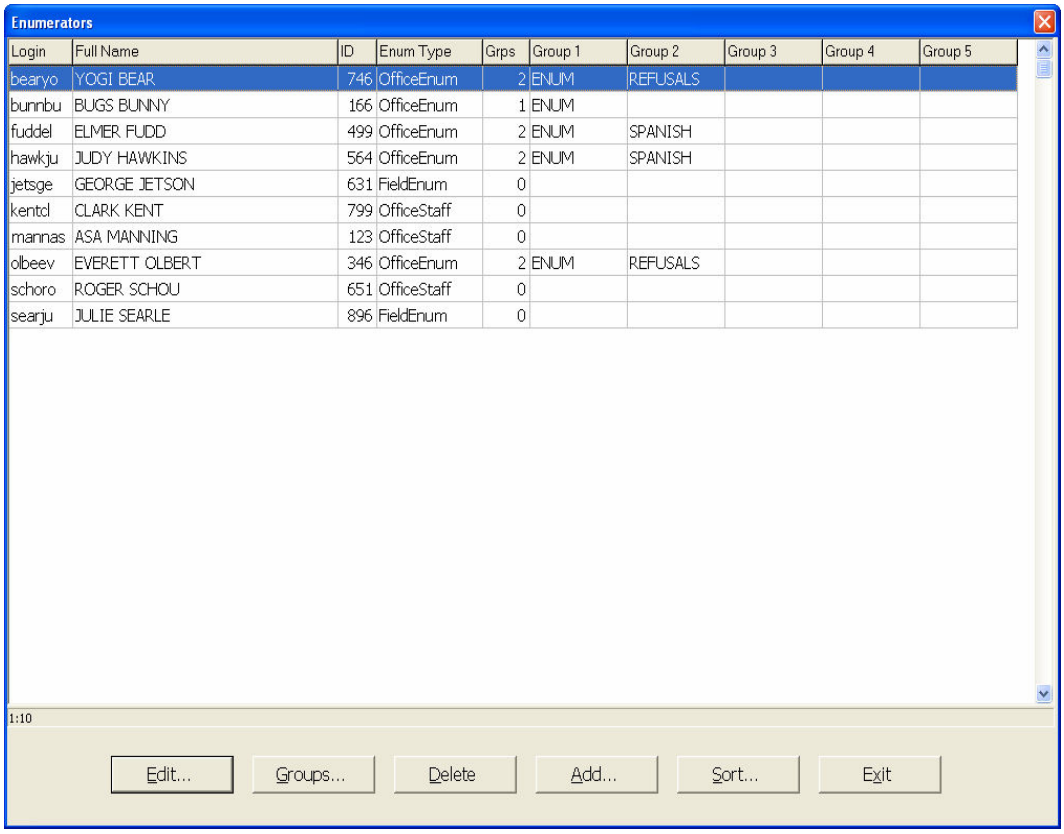
There is a wealth of information that can be gleaned from the Blaise CATI History file. The CASIC Survey Administration system uses the information contained in the CATI History file to generate a number of charts and reports to aid Data Collection Center Coordinators in monitoring survey activities and interviewer performance.

2.2. NASS Blaise Interviewer Database

A Blaise dataset stores information on each interviewer. This information includes the login name, the full name, the employee number, an indicator of office interviewer, field interviewer, or office staff, up to five CATI group memberships, and the supervisor's employee number. A ManiPlus setup is used to access this data to create an .IGL file for any survey. This data will also be used by the CASIC Survey Administration system by using the employee number and login name to provide a link between information in the CATI History file, Blaise survey dataset, and NASDA cost data.

A ManiPlus setup gives the Data Collection Center Coordinator access to this Blaise dataset, allowing them to add, delete, and edit information that is stored for each interviewer (see Figure 1).

Figure 1. Maniplus Table for Managing Interviewers in the Interviewer Dataset



This table may be sorted by login name (Login), interviewer type (EnumType), employee number (ID), or descending number of CATI groups (Grps).

2.3. Blaise Survey Dataset Office Use Fields

On nearly all of the surveys NASS conducts using the Blaise software, standard office use codes are utilized. These codes store information on Interviewer Employee Number, Interviewer Evaluation, Response Code, Respondent Code, Mode Code and Collection FIPS. Examples of these variables are shown below.

Interviewer Employee Number: ranges from 200-999

Response Code:
1 = Complete
2 = Refusal
3 = Inaccessible
4 = Office Hold
5 = Estimated Refusal
6 = Estimated Inaccessible
7 = Estimated Office Hold
8 = Known Zero

Respondent Code:
1 = Operator or Manger
2 = Spouse
3 = Accountant or Bookkeeper
4 = Partner
9 = Other

Mode Code:	1 = Mail 2 = Telephone on paper 3 = Face-to-Face Interview 4 = CATI 5 = Web 6 = e-mail 7 = Fax 8 = CAPI 19 = Other
DCC (Collection) FIPS:	05 = Arkansas 21 = Kentucky 30 = Montana 40 = Oklahoma 56 = Wyoming

This information is populated into the CASIC Survey Administration system each time it is executed and is used to generate reports on survey response rates, mode rates and individual interviewer performance with regards to response rates.

2.4. NASDA Electronic Time Sheet System

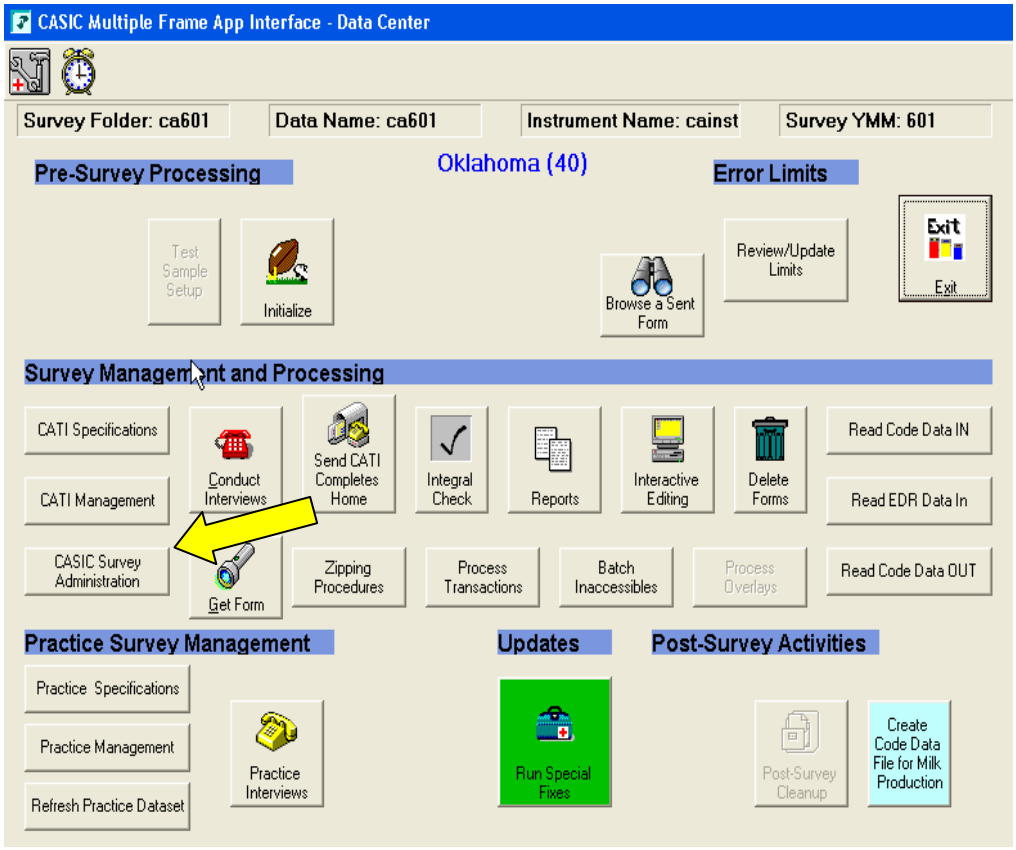
In the past, NASDA interviewers recorded their time on paper time sheets which were collected, keyed, and read into a time sheet system for summary and eventual payment over a two week period. This system did not lend itself to timely access to cost data by Data Collection Center Coordinators responsible for preventing cost overruns on many different projects. Coordinators found themselves calculating cost information by hand in order to ensure that they were staying within budget on survey projects which proved to be very time consuming and inefficient.

During February 2006, NASS introduced a new NASDA Electronic time sheet system written by its North Carolina Field Office. This new system allows interviewers to electronically fill out their daily time sheet right from the convenience of their desktops. With this information being recorded on a very timely basis, the CASIC Survey Administration system will eventually be programmed to retrieve cost data for the project and dates specified by the Data Collection Coordinator and this data will be used to generate project level cost reports as well as interviewer level cost reports.

3. Running the CASIC Survey Administration System

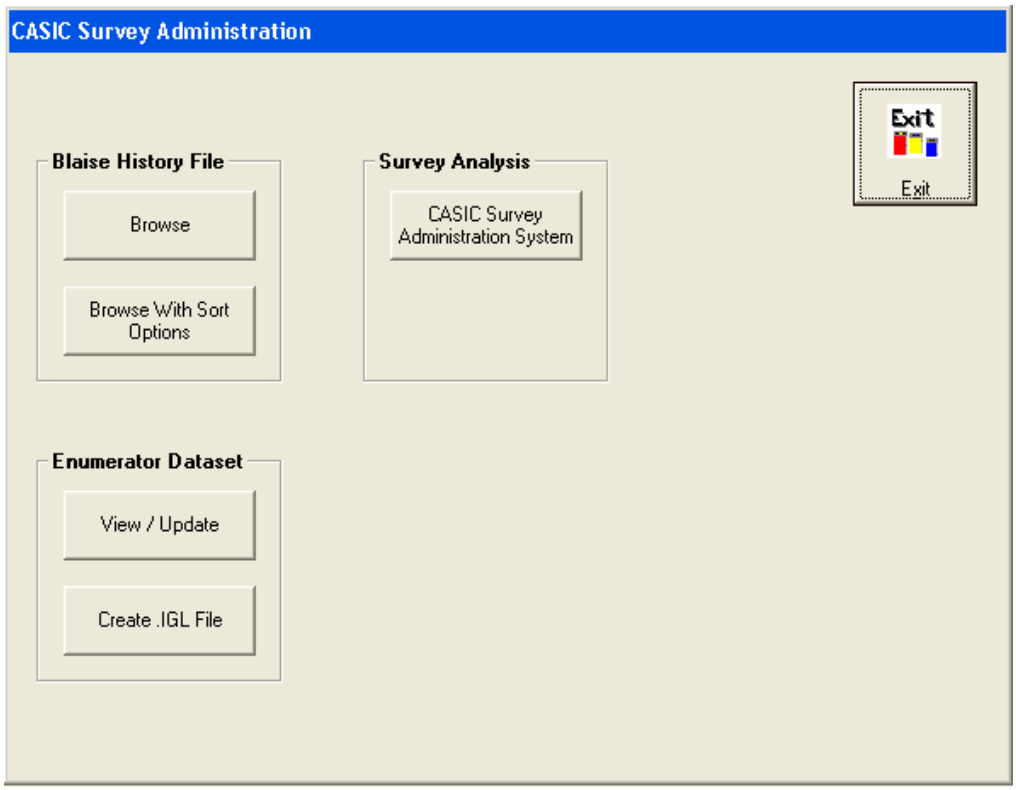
NASS uses a Visual Basic Menu system from which to execute its Blaise survey processes. The standard interface for the Data Collection Centers is shown in Figure 2.

Figure 2. Standard CASIC Visual Basic Menu For Data Collection Center



After clicking on the CASIC Survey Administration button, the form in Figure 3 is presented to the user:

Figure 3. CASIC Survey Administration Form



3.1. Blaise History File

From this form, the user can execute traditional Blaise History functions. By clicking on the BROWSE button the user can browse the History file using the History Viewer application (bthist.exe). BROWSE WITH SORT OPTIONS opens the History file into the Blaise Data Browser and gives the user the ability to sort the History data. The instrument used to view the data has the following three secondary keys upon which to sort: 1) record id, dial date, dial time; 2) interviewer login, record id, dial date, dial time; and; 3) dial date, interviewer login, dial time. Both these options have been a standard part of the CASIC Visual Basic Menu system for many years.

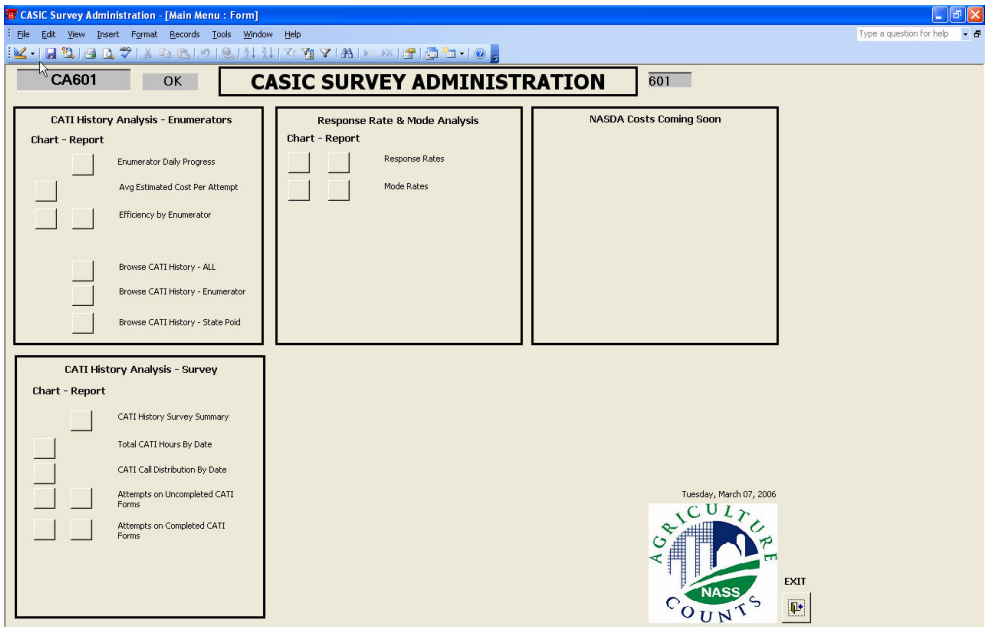
3.2. Interviewer (Enumerator Dataset

The VIEW/UPDATE button gives the user access to the Interviewer Dataset already discussed in section 2.2 above. From this form, the user can also click on CREATE .IGL FILE and an .IGL file will be created from information contained in the Interviewer Dataset for population into the CATI Specification file for the survey.

3.3. Populating and Launching the CASIC Survey Administration System

When the Data Collection Center Coordinator is ready to launch the CASIC Survey Administration system, they do this by clicking on the CASIC SURVEY ADMINISTRATION SYSTEM button. Once executed, the following events take place. The Interviewer Dataset is queried and required information is written to a temporary file, a copy of the CATI History file is made, and the Blaise survey office use data is queried for the survey of interest and written to a temporary file. A SAS program is executed which takes the information from all 3 sources, merges data where necessary, creates new variables, summarizes data, and populates all data into the Microsoft Access database. Once the Microsoft Access database has been populated, the following interface is presented to the user:

Figure 4. CASIC Survey Administration System



From this interface, the Data Collection Center Coordinator can run a variety of reports and charts to monitor and evaluate survey progress and interviewer performance. Some of these charts and reports are presented in more detail in the next section.

4. CASIC Survey Administration System Analysis (Charts and Reports)

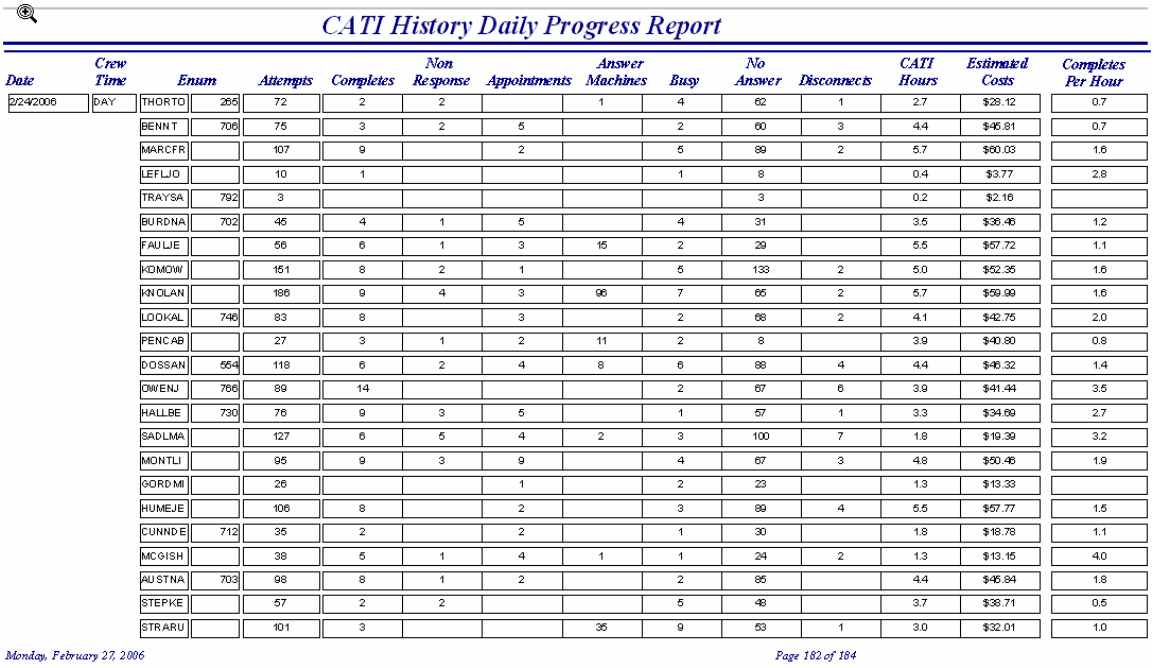
Providing a quality data product to Serviced States is a priority of each Data Collection Center. Maintaining high response rates is a goal and objective of each DCC in order to ensure that quality data is being delivered to Serviced States. The CASIC Survey Administration system will provide reports such as the CATI History Daily Progress Report and Survey Response Rate report on a real time basis as well as post survey final reports which can be sent to the Survey Administrator in Headquarters, Washington, D.C.

4.1. CATI History Daily Progress Report

The CATI History Daily Progress Report is broken down by Date, Crew Time (Day or Night), and Interviewer. It shows the number of dial attempts, number of completes, number of non responses, number of appointments made, number of answering machines, number of busy signals, and number of disconnects. It also shows the total number of CATI Hours registered by the Blaise system, and currently, an estimate of NASDA cost expenditures.

The report also gives totals and averages for each Crew Time (Day and Night) and finally totals and averages for the entire crew for the dates queried. This gives the Data Collection Center Coordinator the ability to evaluate not only individual interviewers but also the Day Crew and the Night Crew as separate entities. A new column just recently added shows Completes per Hour, and soon, actual NASDA costs and hours will be incorporated.

Figure 5. CATI History Daily Progress Report



CATI History Daily Progress Report

Date	Crew Time	Enum		Attempts	Completes	Non Response	Appointments	Answer Machines	Busy	No Answer	Disconnects	CATI Hours	Estimated Costs	Completes Per Hour
2/24/2006	DAY	THORTO	265	72	2	2		1	4	62	1	2.7	$28.12	0.7
		BENNT	706	75	3	2	5		2	60	3	4.4	$45.81	0.7
		MARCFR		107	9		2		5	89	2	5.7	$60.03	1.6
		LEFLJO		10	1				1	8		0.4	$3.77	2.8
		TRAYSA	792	3						3		0.2	$2.16	
		BURDNA	702	45	4	1	5		4	31		3.5	$36.46	1.2
		FAULJE		56	6	1	3	15	2	29		5.5	$57.72	1.1
		KDMOW		151	8	2	1		5	133	2	5.0	$52.35	1.6
		KNOLAN		186	9	4	3	96	7	65	2	5.7	$59.99	1.6
		LOOKAL	746	83	8		3		2	68	2	4.1	$42.75	2.0
		PENCAB		27	3	1	2	11	2	8		3.9	$40.80	0.8
		DOSSAN	554	118	6	2	4	8	6	88	4	4.4	$46.32	1.4
		OWENJ	766	89	14				2	67	6	3.9	$41.44	3.5
		HALLBE	730	76	9	3	5		1	57	1	3.3	$34.69	2.7
		SADLMA		127	6	5	4	2	3	100	7	1.8	$19.39	3.2
		MONTLI		95	9	3	9		4	67	3	4.8	$50.46	1.9
		GORDMI		26			1		2	23		1.3	$13.33	
		HUMEJE		106	8		2		3	89	4	5.5	$57.77	1.5
		CUNNDE	712	35	2		2		1	30		1.8	$18.78	1.1
		MCGISH		38	5	1	4	1	1	24	2	1.3	$13.15	4.0
		AUSTNA	703	98	8	1	2		2	85		4.4	$45.84	1.8
		STEPKE		57	2	2			5	48		3.7	$38.71	0.5
		STRARU		101	3			35	9	53	1	3.0	$32.01	1.0

Monday, February 27, 2006 Page 182 of 184

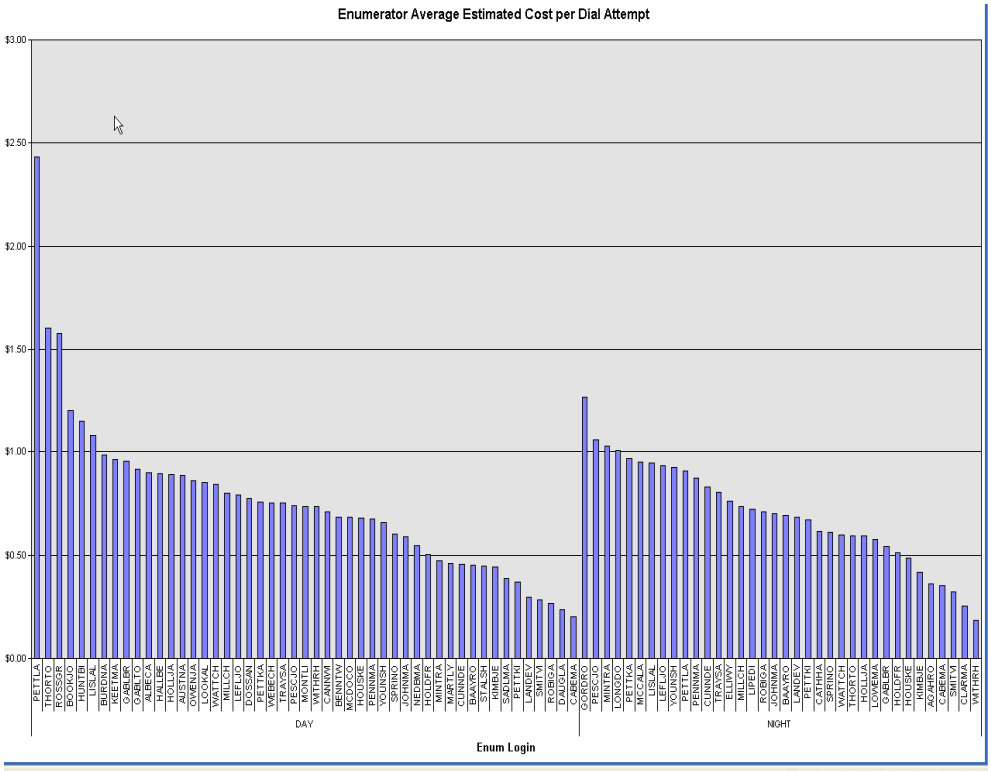
Date	Crew Time	Enum		Attempts	Completes	Non Response	Appointments	Answer Machines	Busy	No Answer	Disconnects	CATI Hours	Estimated Costs	Completes Per Hour
2/24/2006	DAY	JOHNMA		6						6		0.2	$2.30	
		NEDBMA	762	54	5		4		1	42	2	2.8	$29.60	1.8
		HUNTBI	737	104	10		5		6	76	7	3.9	$40.53	2.6
Summary for 'CrewTime' = DAY (26 enumerators)														
			Sum	1945	140	30	66	169	80	1411	49	87.1	$914.28	
			Avg	74.8	5.4	1.2	2.5	6.5	3.1	54.3	1.9	3.3	$35.16	1.6
2/24/2006	NIGHT	LEFLJO		40	2		2		1	34	1	3.8	$39.66	0.5
		BAAYRO	804	54	3	1	4		4	41	1	2.1	$21.85	1.4
		USLAL	744	55			6		4	45		2.0	$20.52	
		TRAYSA	792	67	4		3		5	55		3.2	$33.85	1.2
		CUNNDE	712	4	1		2			1		0.2	$1.81	5.8
		HOLLJA	735	65	6	2	1		2	54		1.4	$15.04	4.2
		PETTLA	773	48			2		1	45		1.4	$14.50	
		GORDMI		61	3	1	2		3	51	1	3.0	$31.39	1.0
		MINTRA		115	8	1	8		11	83	4	3.0	$31.92	2.6
		THORTO	265	25	6		3			16		0.6	$6.65	9.5
		AGAHR		23						23		0.4	$4.50	
		SPRINO	788	38			1		1	33	3	1.2	$12.19	
		WITHRH	801	134	8	1	5		3	110	7	1.7	$18.02	4.7
		GORDR		31	3		3			25		1.8	$19.07	1.7
		JOHNMA		100	8		1		7	83	1	3.3	$34.43	2.4
		SMITVI	786	28	4		5		1	17	1	0.7	$7.60	5.5

Date	Crew Time	Enum	Attempts	Completes	Non Response	Appointments	Answer Machines	Busy	No Answer	Disconnects	CATI Hours	Estimated Costs	Completes Per Hour
Summary for 'CrewTime' = NIGHT (16 enumerators)													
		Sum	888	56	6	48		43	716	19	29.8	$313.02	
		Avg	55.5	3.5	0.4	3.0		2.7	44.8	1.2	1.9	$19.56	1.9
Summary for 'DialDate' = 2/24/2006 (42 enumerators)													
		Sum	2833	196	36	114	169	123	2127	68	116.9	$1,227.30	
		Avg	67.5	4.7	0.9	2.7	4.0	2.9	50.6	1.6	2.8	$29.22	1.7
Grand Total			204,438	30,388	5,217	14,627	19,655	9,907	109,763	14,881	9,082.8	$95,369.14	3.3

4.2. Interviewer Average Estimated Cost per Dial Atttempt

An example of one of the charts that is available in the CASIC Survey Administration system to evaluate interviewer performance is shown in Figure 6 below. The chart shows the average estimated NASDA cost per dial attempt for all dial results. The chart is divided into the Day Crew on the left and the Night Crew on the right. Interviewers with a high cost per dial attempt can be identified and proper training or other action can be taken as necessary.

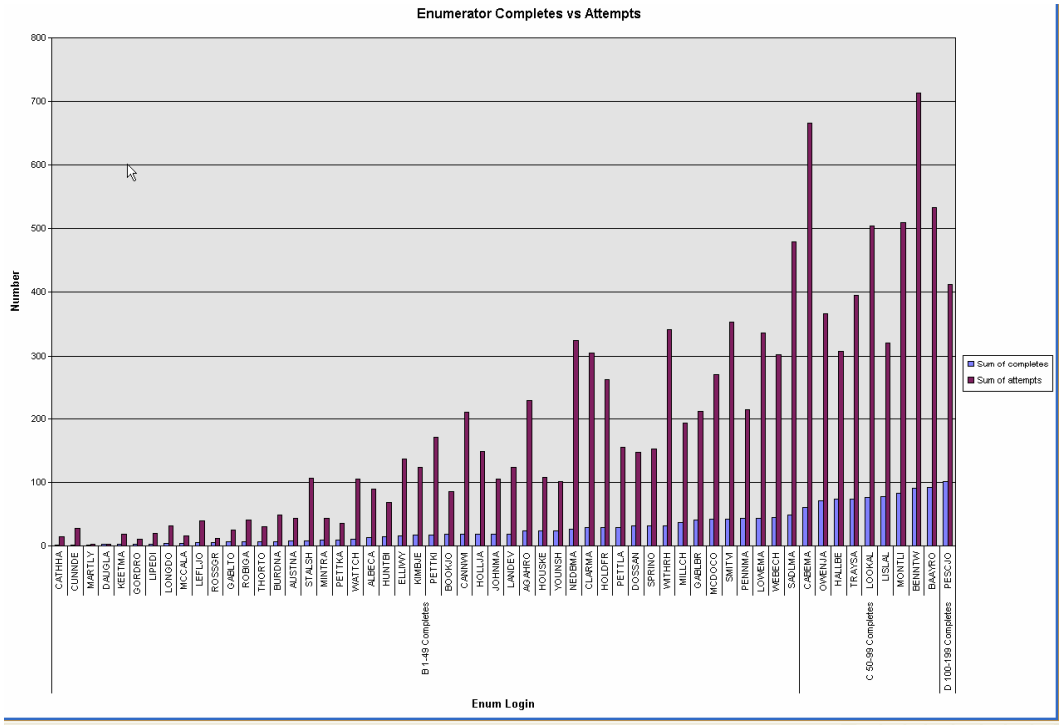
Figure 6. Interviewer Average Estimated Cost Per Dial Attempt



4.3. Interviewer Completes vs Attempts

Another chart available to Coordinators is the Interviewer Completes vs. Attempts. This chart shows the sum of completes next to the sum of attempts for each interviewer grouped by the number of completes. From this chart, interviewers with high attempt rates and low completes can be identified and extra training and guidance can be given.

Figure 7. Interviewer Completes vs. Attempts



4.4. Query the CATI History File

As charts or reports show potential issues with an interviewer, it may become necessary to drill down into the data. The CASIC Survey Administration system has 3 options for querying the CATI History data. The data can be queried by providing the specific interviewer, by providing the primary key of the sampled unit, or the entire history file can be viewed. Figure 8 below shows the History file queried for a specific interviewer.
This current report shows the Interviewer Login Name, Employee Id, State the sampled unit belongs to, Primary Key, Dial Date, Hour (Time Slice) in which the dial attempt was made, Dial Result, Dial Minutes, and Estimated NASDA Dial Cost.

Figure 8. CATI History File for Specific Interviewer

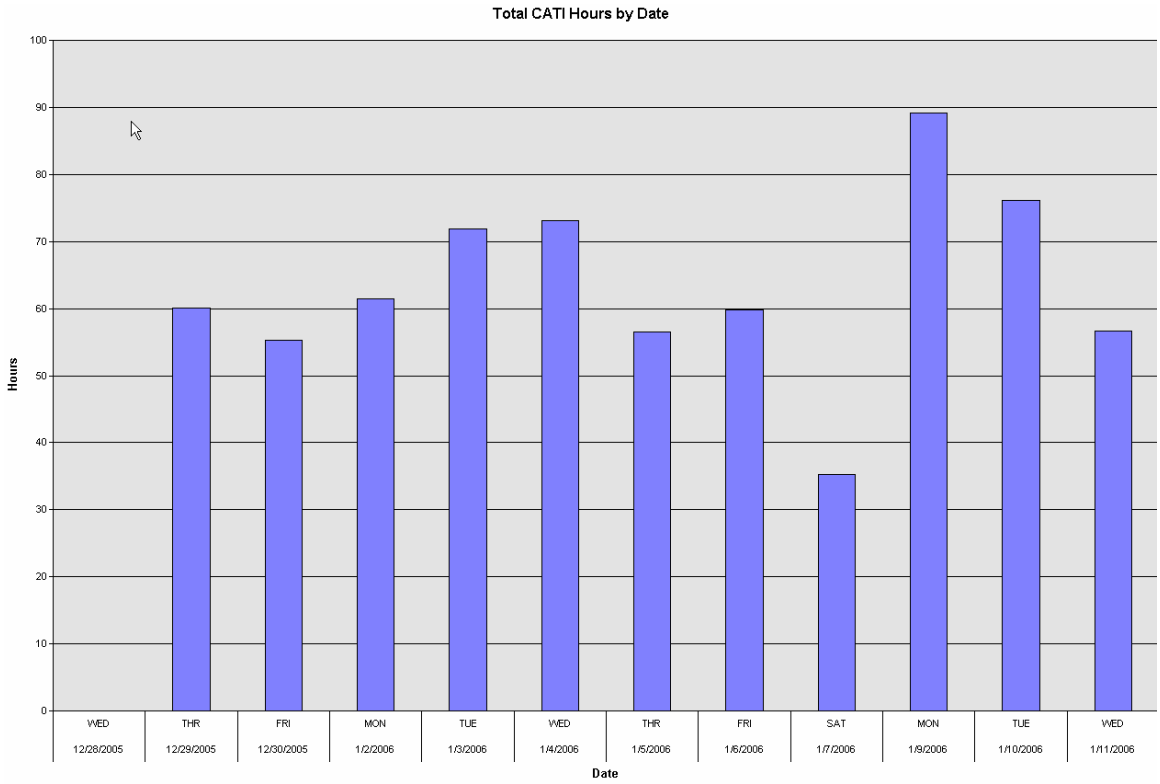
Enum	EnumId	StAlpha	StPoidTrSubTr	Dial Date	Time Slice	Dial Result	Dial Minutes	Dial Est Cost
ALBECA	802	TX	4882078929011	Tuesday, January 10, 2006	B 7am - 7:59am	Appointment	22.8	$3.99
	802	TX	4882078929011	Tuesday, January 10, 2006	C 8am - 8:59am	Appointment	20.3	$3.55
	802	OK	4094700586011	Tuesday, January 10, 2006	C 8am - 8:59am	No Answer	0.7	$0.11
	802	OK	4095705812011	Tuesday, January 10, 2006	C 8am - 8:59am	No Answer	0.2	$0.04
	802	TX	4830075669011	Tuesday, January 10, 2006	C 8am - 8:59am	Appointment	23.9	$4.18
	802	TX	4830075669011	Tuesday, January 10, 2006	C 8am - 8:59am	Appointment	0.2	$0.04
	802	TX	4830075669011	Tuesday, January 10, 2006	D 9am - 9:59am	Usable Complete	10.4	$1.82
	802	TX	4895702418011	Tuesday, January 10, 2006	D 9am - 9:59am	Usable Complete	18.5	$3.23
	802	TX	4810048029011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.4	$0.25
	802	TX	4882001306011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.1	$0.19
	802	OK	4030013446011	Tuesday, January 10, 2006	E 10am-10:59am	Appointment	3.8	$0.66
	802	OK	4095701243011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	5.0	$0.87
	802	OK	4030054341011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.4	$0.25
	802	OK	4091703893011	Tuesday, January 10, 2006	E 10am-10:59am	Usable Complete	17.8	$3.11
	802	TX	4830079621011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.0	$0.17
	802	TX	4810032354011	Tuesday, January 10, 2006	E 10am-10:59am	Appointment	0.6	$0.11
	802	TX	4830056547011	Tuesday, January 10, 2006	E 10am-10:59am	Busy	0.5	$0.08
	802	TX	4882039487011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.2	$0.21
	802	TX	4889701782011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	2.6	$0.46
	802	TX	4882038910011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.3	$0.23
	802	OK	4030003328011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.2	$0.21
	802	TX	4830325669011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.1	$0.20
	802	TX	4888736519011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	3.0	$0.52
	802	OK	4093702176011	Tuesday, January 10, 2006	E 10am-10:59am	No Answer	1.3	$0.22
	802	TX	4887721598011	Tuesday, January 10, 2006	E 10am-10:59am	Appointment	15.2	$2.66
	802	OK	4030037409011	Tuesday, January 10, 2006	F 11am-11:59am	Disconnect	4.4	$0.78
	802	TX	4891714317011	Tuesday, January 10, 2006	F 11am-11:59am	No Answer	0.7	$0.13
Summary for 'Enum' = ALBECA (89 detail records)						Sum	457.1	$79.99
						Avg	5.1	$0.90

Saturday, February 25, 2006 Page 12 of 390

4.5. Total CATI Hours by Date

Sometimes it is necessary for Data Collection Center Coordinators to review the number of hours they spent on CATI operations for a project. The CASIC Survey Administration system can give them a very strong estimate of the number of hours spent calling with CATI by date. By using the information contained in the CATI History file, the following chart can be generated (a report can also be generated). The chart shows, by date, the total number of CATI hours registered by the Blaise system.

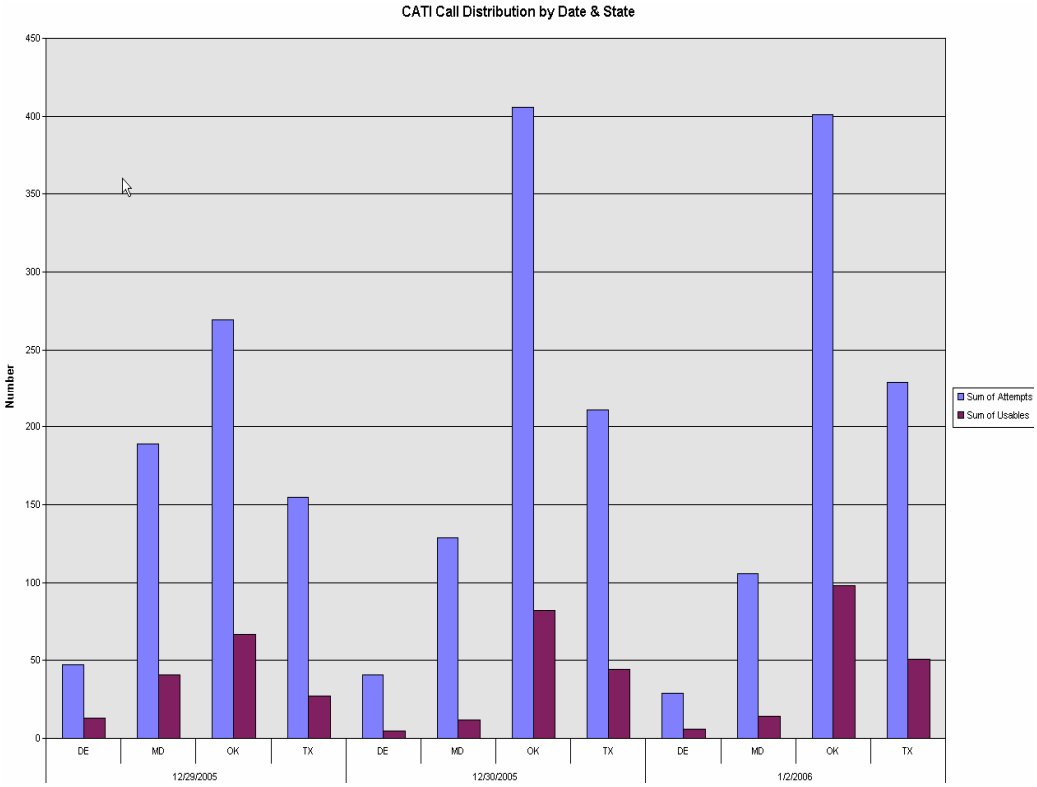
Figure 9. Total CATI Hours by Date



4.6. CATI Call Distribution by Data and State

Data Collection Centers call for themselves and typically 3 or 4 other states. It becomes necessary for the Data Collection Center Coordinator to make sure that calling resources are distributed equitably between all states being serviced. Currently, the Blaise CATI Management system does not offer any counts grouped by geographical region and therefore makes it somewhat difficult to make sure all states being serviced are receiving their fair share of the calling resources. The chart below can be used by the Coordinator to monitor the number of dial attempts and completes by date and state to ensure that states are being treated equitably. As inequalities in calling distribution are noted in the chart below, the Data Collection Center Coordinator can make adjustments to the day batch by excluding certain states for periods of time. If interviewer groups are being used, then adjustments to the number of interviewers in each state's group can be made to affect the number of calls being routed to that state.

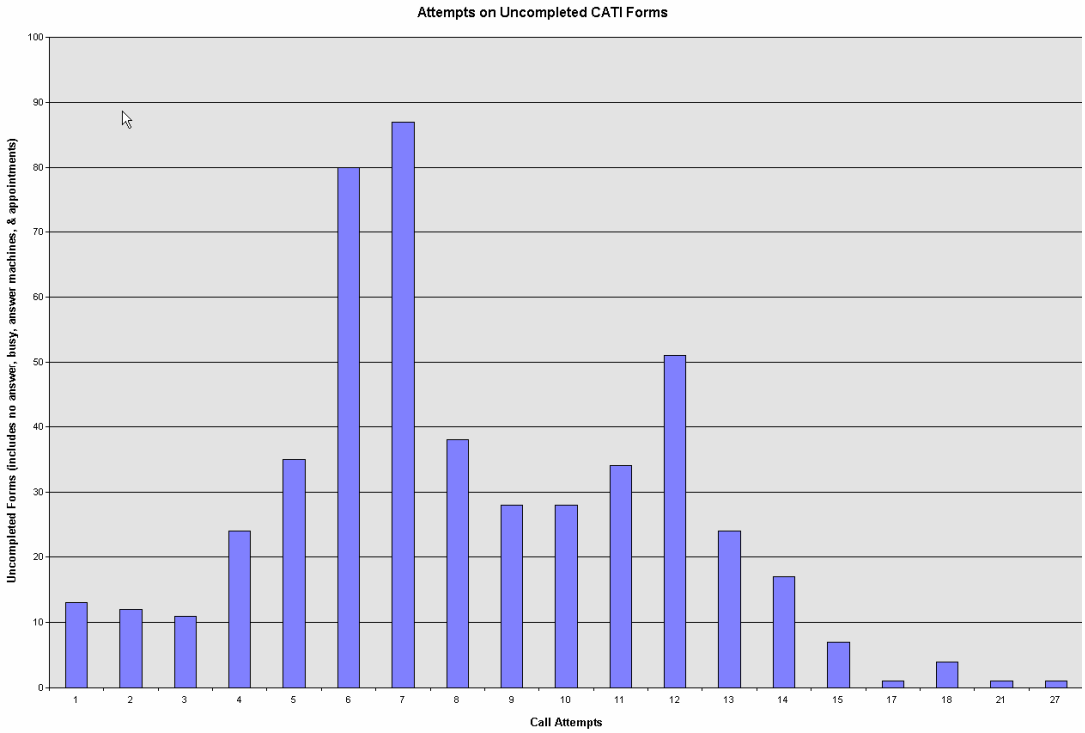
Figure 10. CATI Call Distribution by Date and State



4.7. Attempts on Uncompleted CATI Forms and Attempts on Completed CATI Forms

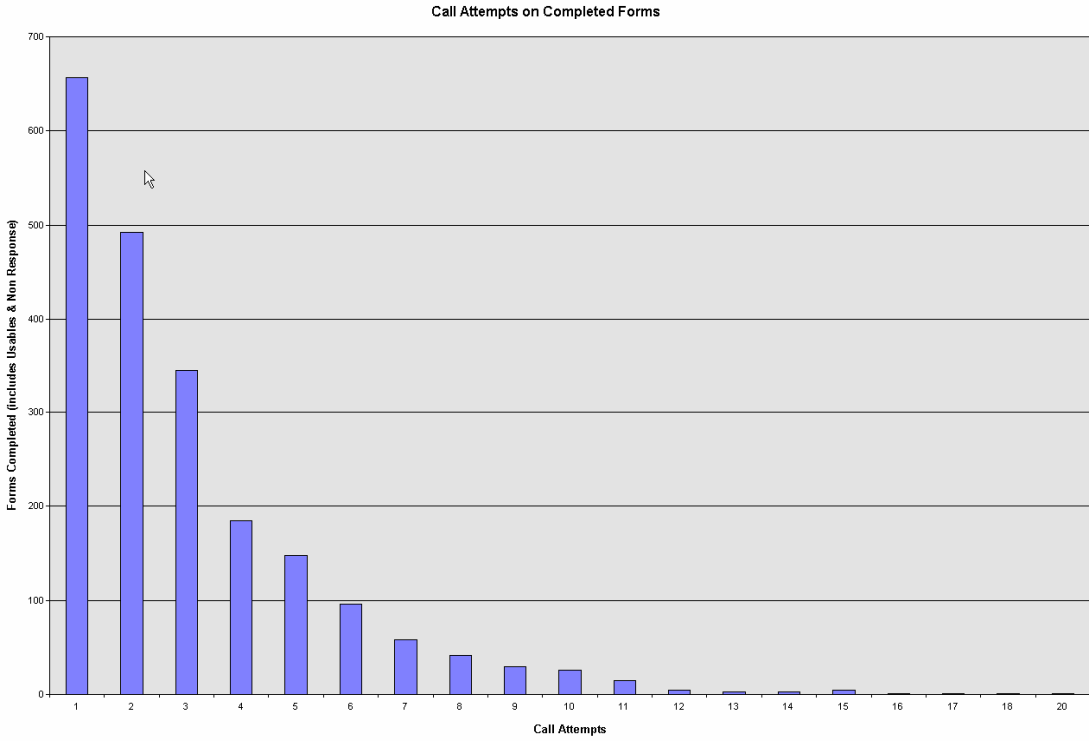
Although the Blaise CATI Management system does a good job of showing counts of records with up to 5 dial attempts, it doesn't present the full picture. The chart in Figure 11 below shows the Attempts on Uncompleted CATI forms. In this example, a good portion of our forms still available to be called have had 4 or more dial attempts. As resources for data collection efforts become more and more scarce, it may become necessary for Data Collection Center Coordinators to take information such as is presented in this chart and turn off cases that have had X number of dial attempts and stop spending resources on those cases. However, before, cases are turned off it must be guaranteed that they have been attempted over a sufficient number of time slices.

Figure 11. Attempts on Uncompleted CATI Forms



The next chart shown in Figure 12 below represents the number of Dial Attempts on Completed CATI Forms. In this example, most of our CATI completes were completed with 11 or fewer attempts. Examining the data that makes up the chart shows that 99% of CATI completes were completed with 11 dial attempts or fewer. In some NASS surveys it has been shown that 99% of CATI completes are completed with 5 or fewer attempts and that only 1% of completes are completed after 5 attempts.

Figure 12. Attempts on Completed CATI Forms



4.8. Survey Response Rates

Good response rates have always been important to NASS and the work it does; however, in recent years NASS management has taken a renewed interest in strengthening response rates. In fact, response rates in the NASS Field Offices are now tied to the performance appraisals of the management staff in those Field Offices.

In many cases, Data Collection Center Coordinators do not have easy access to response rates for the surveys they conduct unless they write special adhoc programs to summarize this data. Many times, they must wait until the final summary is executed, which is at the very end of the survey cycle, before they see any indication of response rates. Now, the new CASIC Survey Administration system will give them real time response rates at the click of a button so that they can monitor this very important statistic on a daily basis. The chart in Figure 13 and the report in Figure 14 show the response rates for the Montana Data Collection Center collecting data for Arizona, California, Colorado, Iowa, and Montana.

Figure 13. Response Rates

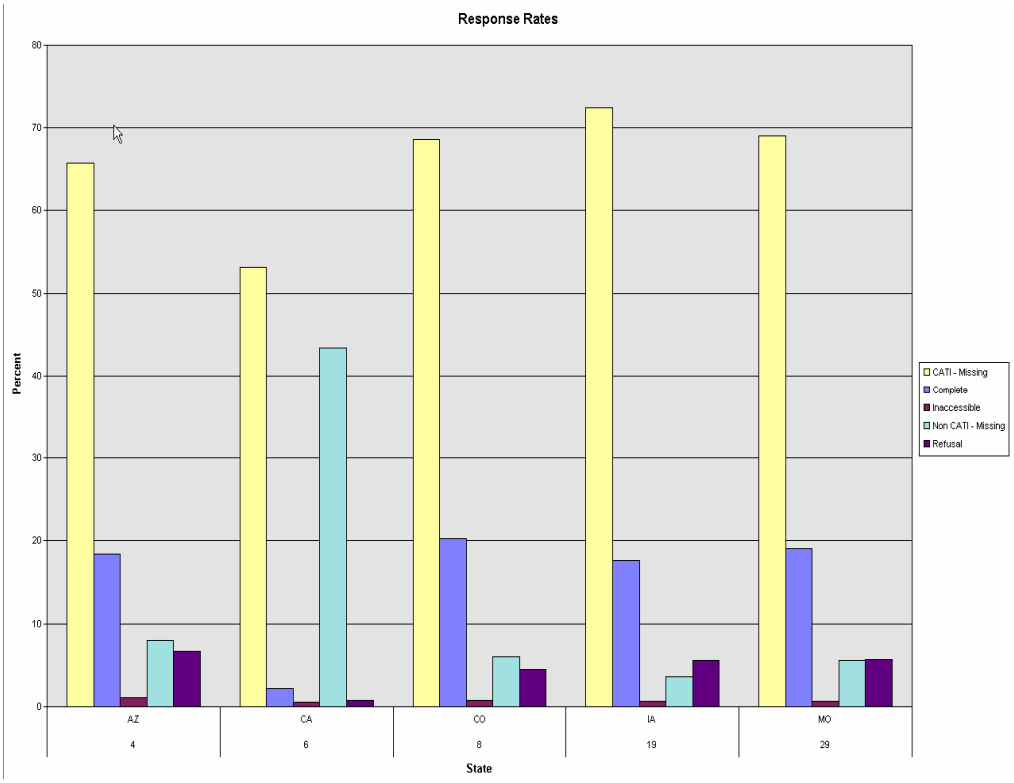
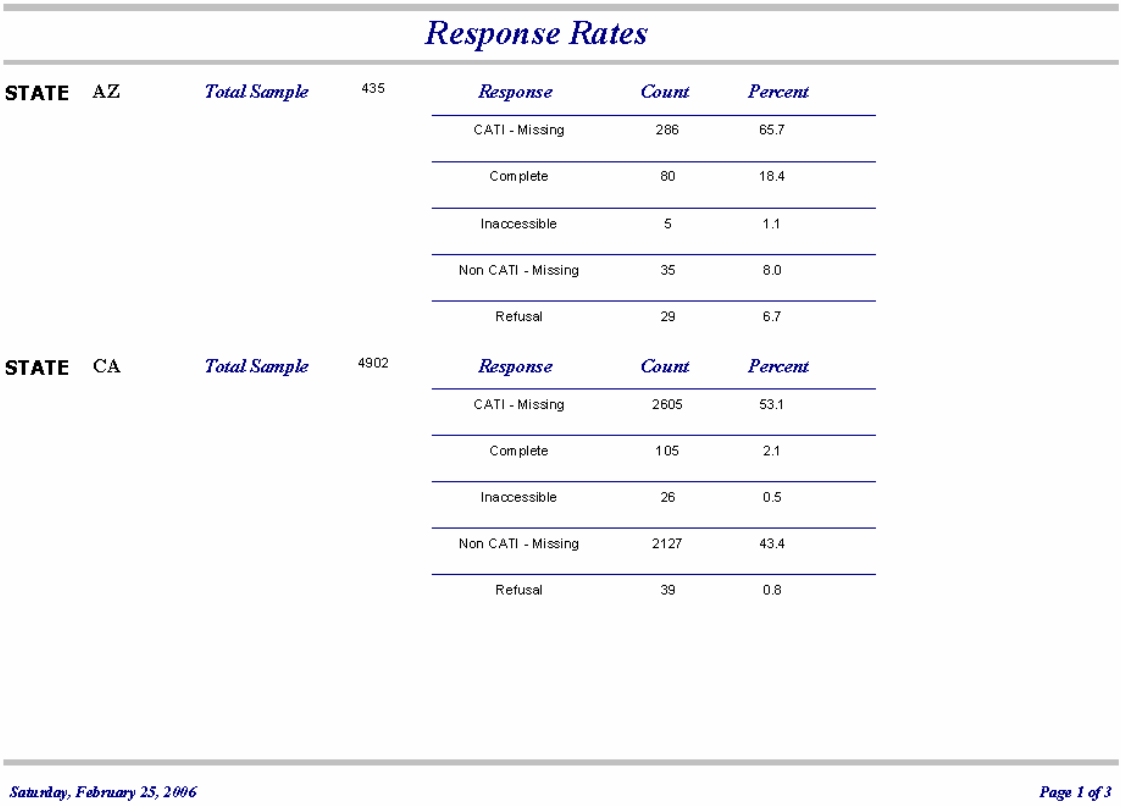


Figure 14. Response Rates



Response Rates

STATE AZ — *Total Sample* 435

Response	*Count*	*Percent*
CATI - Missing	286	65.7
Complete	80	18.4
Inaccessible	5	1.1
Non CATI - Missing	35	8.0
Refusal	29	6.7

STATE CA — *Total Sample* 4902

Response	*Count*	*Percent*
CATI - Missing	2605	53.1
Complete	105	2.1
Inaccessible	26	0.5
Non CATI - Missing	2127	43.4
Refusal	39	0.8

Saturday, February 25, 2006 *Page 1 of 3*

5. Future Direction

Currently, Data Collection Center Coordinators calculate costs on a weekly basis by hand which is very inefficient and time consuming. Coordinators need the ability and tools to run cost reports on a daily basis to prevent cost overruns. The CASIC Survey Administration system currently gives an estimate of NASDA expenditures based on the time registered in the Blaise CATI History file. However, with the recent deployment of the NASDA Electronic Time Sheet system, the CASIC Survey Administration system will be programmed to retrieve data from the electronic time sheet system and combine it with other data already present in the system to provide up-to-date actual NASDA cost expenditures to Data Collection Center Coordinators and NASDA Supervisors.

NASS has a Data Warehouse where all survey data is stored once final summaries are executed. In the future, it would be beneficial to NASS to also store Blaise CATI History files and NASDA cost expenditure data in its Data Warehouse. With this administrative data located in a central database, many of the charts and reports presented in this paper could be generated by anyone in NASS for any number of reasons including planning, budgeting, and research.

6. Conclusion

With more and more burden and responsibility being put on NASS's five Data Collection Centers to collect its data using CATI, it falls upon the CASIC Section in Headquarters to provide them with the tools they need to successfully fulfill their mission. The CASIC Survey Administration system is one step in that direction. By bringing information together from several sources, the CASIC Survey Administration system gives Data Collection Center Coordinators the

monitoring and evaluation tools necessary to successfully collect data and maintain a strong, efficient NASDA interviewing crew.

7. References

Schou, Roger, *Integrating the use of Data Centers into the NASS CAI Structure*, National Agricultural Statistics Service, USA, IBUC 2001 7th International Blaise Users Conference, 2001

A Multi-mode CATI-Web Survey Experience with Blaise

Craig Ray (Westat, USA)

1. Abstract

In 2005, Westat developed and implemented a multi- modal (CATI and web) household interview that collected nationally representative data on the public's need for, access to, and use of cancer-related information. The survey instrument was developed with Blaise software for both the web and the computer-assisted telephone interviewing (CATI). To support the survey, Westat deployed a complex management system to control and track survey responses. This paper discusses the system's implementation, including the technical approach, software development process, technical considerations such as security, potential obstacles, and experiences gained.

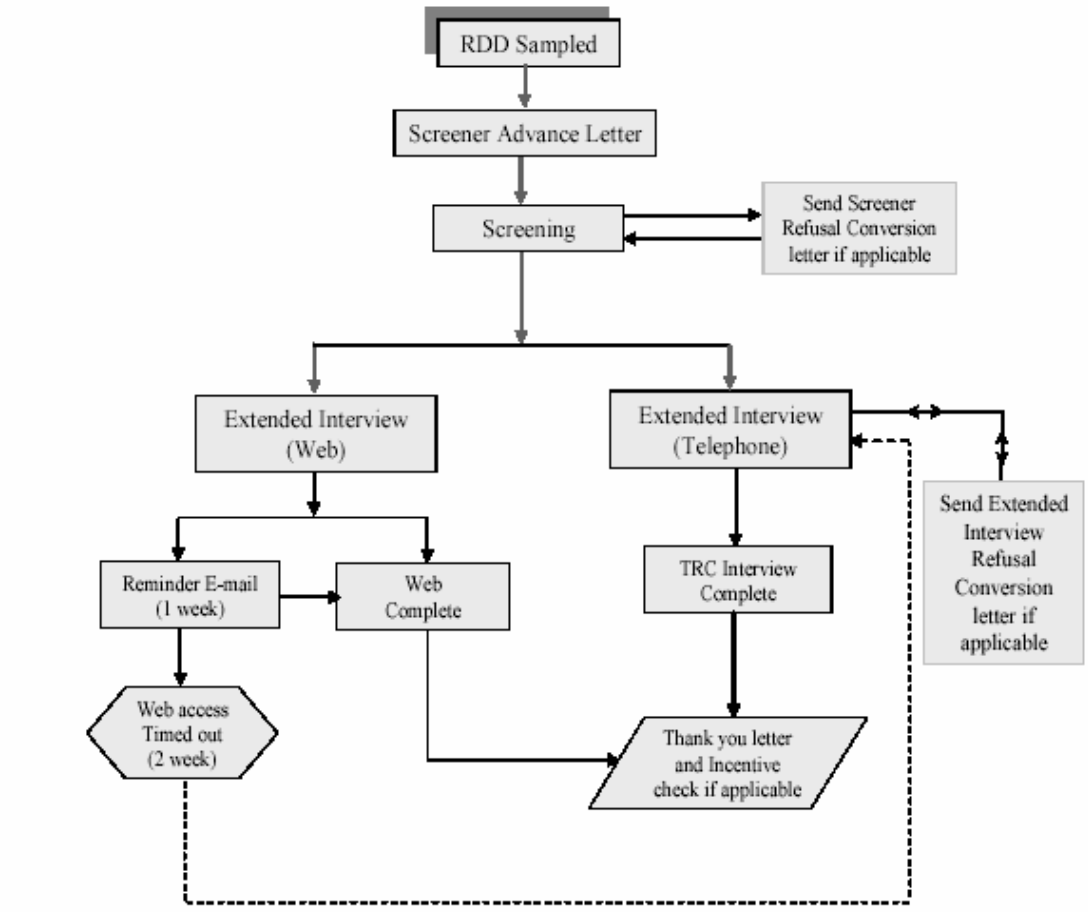
2. HINTS study protocol

The Health Information National Trends Survey (HINTS), sponsored by the National Cancer Institute (NCI), collects nationally representative data about the use of cancer-related information in the U.S.

For the second cycle of HINTS, NCI was interested in exploring the efficacy of using a web version of the questionnaire along with the primary CATI interview. A representative sample was drawn by random digit dial (RDD). All sampled phone numbers were randomly placed in a two-by-two experiment for an extended interview. The cells of the experiment indicated the mode of interview and an incentive amount and consisted of CATI/$5, CATI/$15, CATI-web choice/$5, and CATI-web choice/$15. For sampled phone numbers for which an address match was obtained, Westat mailed an advance letter containing a two dollar bill as an incentive to participate in the study. Each sampled phone number was screened via CATI and upon completion of the screener, an algorithm sampled one member from the household for participation in the extended interview. If the sampled person qualified for and chose web interviewing, we sent an e-mail or letter that contained the web site's URL and their unique username and PIN. Otherwise, the CATI interviewer immediately attempted to complete the extended interview or set an appointment for a later time. Web cases that did not complete the process within one week were sent a reminder letter, and after two weeks were transferred for CATI processing of the extended interview. For those who completed the extended interview, we sent a 'thank you' letter containing the promised incentive payment of either $5 or $15.

Figure 2.1 illustrates the basic study operational flow.

Figure 2.1 – HINTS II operational flow



3. Technical architecture

Figure 3.1 highlights the technical architecture of the entire system. The dual modality made the overall system complex. Westat deployed a number of technologies that were all controlled by the central Study Management System and Integrated Database. The system consisted of the following components:

RDD sample selection: We developed RDD sampling programs to select a nationally representative sample of randomly selected phone numbers, exclude non-residential numbers, and find matching addresses against national databases.

Telephone Research Center: Westat's Telephone Research Center (TRC) is equipped with a suite of software tools that assist the team of CATI interviewers. The survey instrument was developed in Blaise for Windows and a sophisticated scheduling component managed the CATI cases.

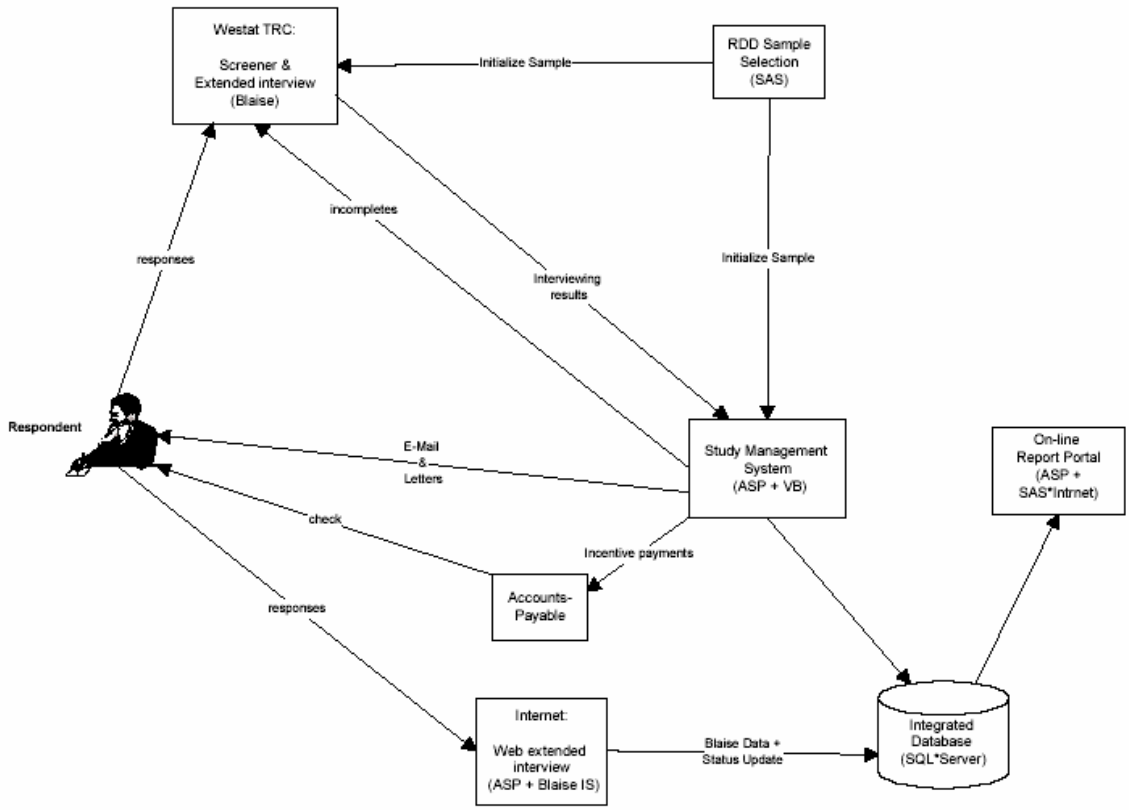
Study Management System (SMS): The overall study management system controlled the flow of cases through the phases of the study. The system consisted of a series of programs developed in Visual Basic (VB) for Windows applications and Active Server Pages (ASP) for an intranet site. The study management system was responsible for loading the sample, maintaining the central database, interfacing with the TRC system and Accounts Payable system, sending all household and respondent e-mail and letters, and providing an interface for the internal help desk staff.

Integrated database: We implemented the central database with SQL Server which contained the central repository for all cases.

Web extended interview: The web extended interview consisted of a respondent home page developed with ASP and instrument developed with Blaise IS software. The respondent was directed to the URL of the respondent home page, which prompted for a login ID and password (authentication) and launched the Blaise IS instrument.

On-line report portal: We implemented central case management reports within Westat's on-line report portal, which is implemented with ASP programs interfacing with SAS/Intrnet software. The team developed reports with SAS, which were implemented within the portal. The SAS programs extracted data from the integrated database in real-time using SAS/Access to OLE DB.

Figure 3.1 – HINTS Technical Architecture



4. The choice of Blaise

We considered several competing technologies for programming the computer assisted interview, but several factors made Blaise the obvious choice.

- The extended interview was lengthy and complex. Blaise is unique among computer assisted interview (CAI) development tools in its ability to handle instrument complexity. The survey instrument consisted of 16 sections and required constructs such as:
  - question order randomization,
  - question assignment randomization,
  - 'check-all-that-apply' questions,
  - text fills,
  - unit/quantity questions, and

- complex question routing.

- The extended interview had to be multi-modal. We expected to achieve economies of scale by using the same programs in both the TRC and the web environments.

- The extended interview had to operate in the CATI environment. The Westat CATI development team had already implemented many complex Blaise instruments. Moreover, the existing TRC technical infrastructure is fully integrated with Blaise.

- The extended interview had to be rendered as a self-interview via the internet. Because of the instrument complexity, a general purpose web development language, such as ASP, was clearly not an option. Among candidate web CAI development tools, none other than Blaise was able to express the complexity required in this survey. Moreover, key members of Westat's web development team were experienced Blaise programmers

Given these factors, Blaise was clearly the best choice of technology. However, at the time the team programmed HINTS instrument, Blaise IS was in beta release. Blaise IS had not yet been deployed for production data collection. The plan described in the following sections was developed through discussion and close interaction with the client and project team in an environment where risks, costs, benefits, and capabilities were understood and considered at all phases of deployment.

5. Instrument development

Ideally, Westat would have developed one Blaise instrument to run in both the CATI and web environments. However, the most prudent course was to focus initial efforts on the development of the CATI instrument. Then as a contingency, if the project suffered unanticipated problems deploying the web version, data collection could be accomplished strictly in the phone center and still accomplish the primary project data collection objectives. As a result, we decided to initially develop and test the CATI instrument and after the programs were deemed suitably mature, transfer the Blaise programs to a Blaise IS developer to customize for the web. The approach had the obvious disadvantage that it required us to maintain two versions of the programs, versions that were very close to identical. After the transfer, any changes to the instrument had to be programmed and tested in both versions. Nevertheless, the project team accepted this drawback.

The effort to recast the CATI instrument to a web instrument was relatively simple. Most of the work consisted of:

- Recasting question texts from interviewer-administered wording to self-administered wording,

- Defining and applying user-interface settings with the Blaise mode library,

- Working with the Blaise IS workshop to adapt the interview XSL stylesheet to implement specialized user-interface elements,

- Deploying the instrument to contain only one question per screen. This was considered an important study consideration; the web version simulated the CATI version because the respondent could not 'look ahead' to future questions, and

- Customizing standard Blaise IS ASP pages that handle authentication, respondent access, general information links, frequently asked questions (FAQs), etc.

In general, we found Blaise IS system exhibited significant advantages in terms of its robust development process and supporting tools. Programming the instrument, refining the 'look-and-feel' so that it was suitable for everyday web users, as well as testing, revision, and implementation all met or exceeded our expectations.

However, because Westat developed and implemented the web version using a beta release of Blaise IS, as expected the implementation encountered several problems with the software. To correct the problems, Statistics Netherlands frequently released new versions of Blaise IS. For each new release, we needed to reinstall the software and recreate the specification file. Sometimes that just meant using the specification interface to check and uncheck boxes or change text. The instrument specification accesses many files and we could not be certain which files were modified for the release. Therefore, Westat began the process anew, each time reapplying the customizations to the files contained in the new release. For example, Westat modified the Starter page, Receipt page, and Abort page manually. Because we did not know with certainty what items changed, the team reapplied the customizations each time.

6. Web interface to Blaise

The HINTS web self interview was an officially sponsored government data collection targeted at everyday web users. Ideally, the respondent home page would have the 'look and feel' of an official government web site, be intuitive, give assurances of privacy and security, and provide suitable study background. While Blaise IS generates a default starter page, the study required a highly customized, intuitive, content-rich home page. Developing the HINTS respondent home page required a substantial development effort.

Figure 6.1 – HINTS Respondent Home Page

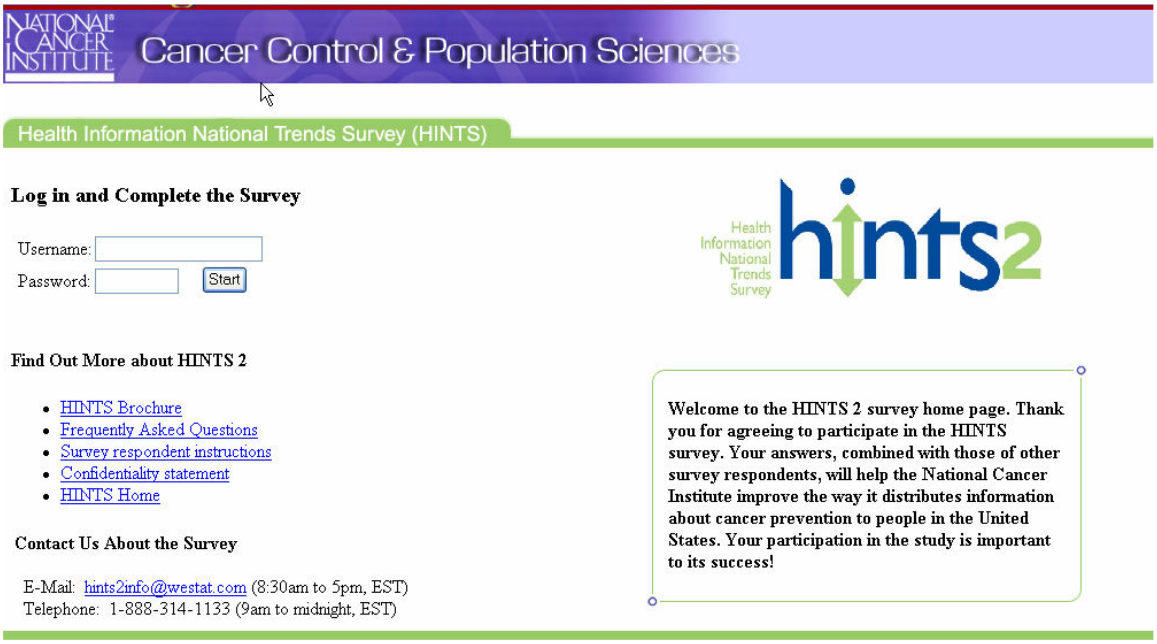


Figure 6.1 depicts of the respondent home page for HINTS. We sent a URL for this page via e-mail or U.S. mail for those who qualified for and agreed to complete the extended interview on the web. There are several notable features on the page.

- **'.GOV' URL**: Westat servers hosted the web site. However, the respondent was sent a URL to a related HINTS .GOV web site (a subdirectory off the main www.CancerControl.cancer.gov). Westat worked with the government client to have that page redirect to the Westat web site. Having the respondent link to a '.GOV' site helped assure respondents that this was an official government research project.

- **Site theme**: The team used colors and graphics to match the general government HINTS web site.

- **Authentication and access control**: The respondent home page has an area for each user to enter username and password. The web page checks the respondent's entry against the integrated SQL Server database (authentication). If the username and password are valid, a record of the login is recorded in the database. The page then determines whether to direct the user to the survey and if so, the respondent is directed to the Blaise IS interview. However, for any of the following reasons a respondent will not be granted access to the interview and a suitable message is displayed in bold red:

    - The user entered an invalid ID or password.

    - The respondent already completed the extended interview.

    - The respondent did not complete the interview within the two week time frame and the case was transmitted back to the TRC for completion (this avoids the conflict of interview being completed simultaneously in both the TRC environment and on the web; only one environment is the 'owner' of the case at any time).

    - The respondent has logged in within the previous 30 minutes. We programmed the message to prevent the same user from logging in twice in separate browser windows or from two users from logging in simultaneously. Simultaneous access of the same case could lead to the user becoming confused, or to inconsistent or inaccurate results.

  When the respondent is granted access to the interview, Blaise IS launches the instrument in a separate browser window. The Respondent Home Page browser window is closed which avoids the possible confusion of having multiple windows open. In the event that this is a resumption of a partially completed interview, the respondent is automatically directed to the first previously unanswered question on the survey route.

- **Study background**: The Respondent Home Page contained links to frequently asked questions, a study brochure, HINTS government home page, and a confidentiality statement.

- **Contact information**: The contact information included a phone number for and link to generate an e-mail to the Westat help desk.

- **Database interface**: Upon authentication and after access was granted the respondent home page contained the interface code with the integrated SQL Server database and the Blaise database. For a new case, the system

instantiated the Blaise record and, upon completion of the session, wrote status information to the database.

7. Blaise IS extended interview

Westat gave great consideration to make the web extended interview intuitive and easy to use and found the features of Blaise and Blaise IS very helpful.

The instrument began with a quick training section that included sample questions with instructions on how to respond and navigate through the survey. The survey seamlessly followed the training section. In order to simulate the CATI environment, during which the respondent is not able to 'look ahead' to future questions, we decided to have the web extended interview render as a single question per page. The undesirable consequence was that for respondents with slow, dial-up connections, the more frequent communication with the web server resulted in an increased length of time to complete the survey.

Figure 7.1 – Sample web extended interview question rendering

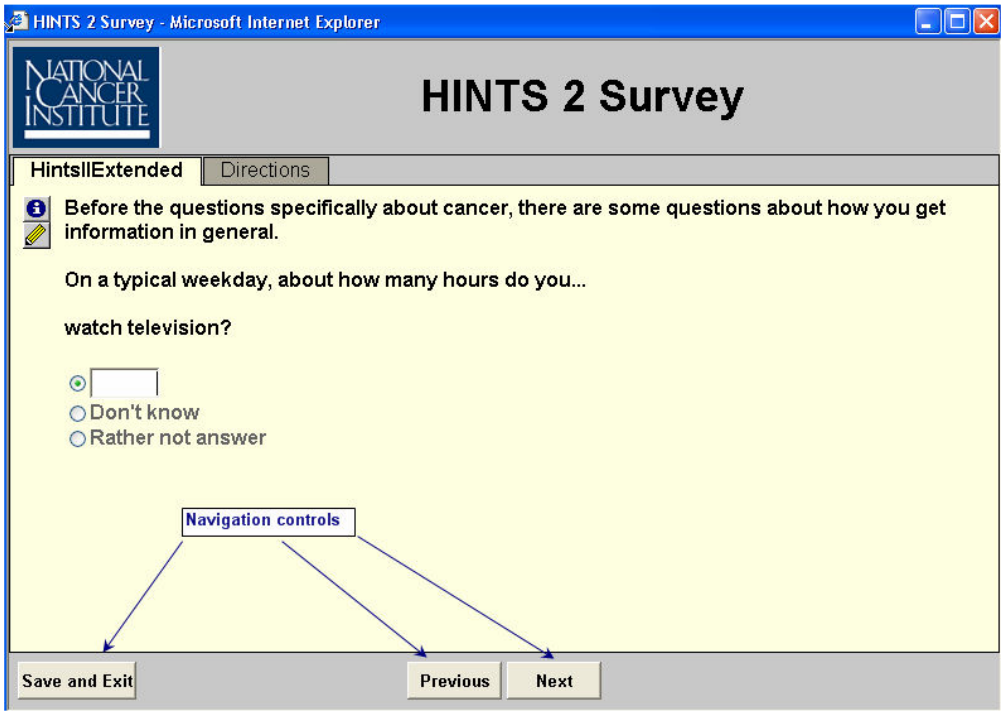


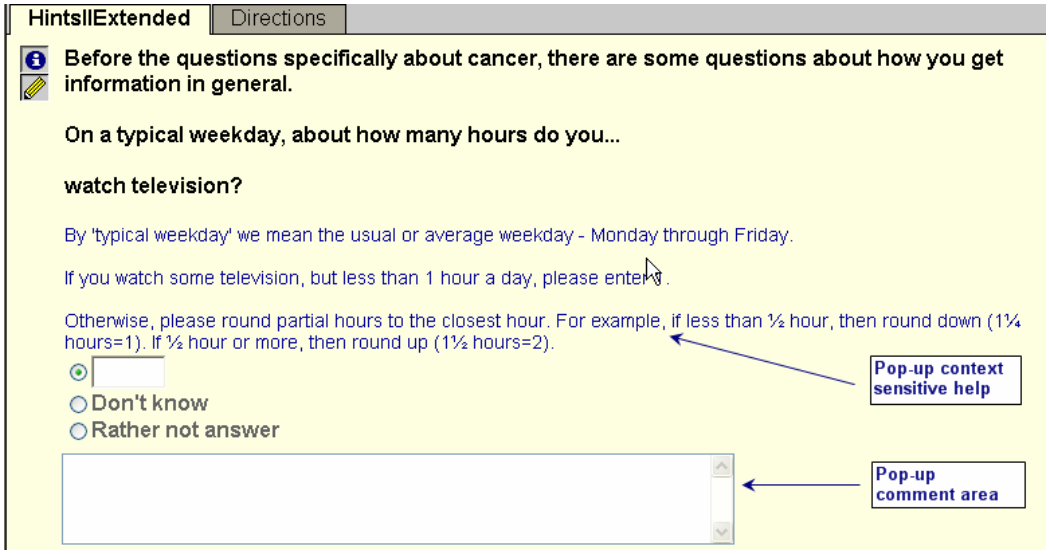
Figure 7.1 illustrates a sample HINTS Blaise IS screen. Blaise IS attempts to launch in a separate Browser window. This permits the software to control the browser appearance, most importantly, hiding all tool and navigation bars. By hiding browser features as <Back> and <Favorites>, it is more difficult and less intuitive for the user to be tempted to navigate to another web site. Rather, the buttons at the bottom of the screen for <Next> question, <Previous> question, and <Save and Exit>, provide all the navigation the user needs to complete the interview.

Westat enabled "don't know" (DK) and "refuse" (RF) for most of the questions. We chose to use a slightly different font on HINTS so the options would not be as prominent. The font and placement of DK and RF were controlled by editing the style sheet within the specification file.

Respondents obtain context sensitive help when pressing the button. We implemented 'help' as a second language within the Blaise code. Furthermore, the respondent may make a standard Blaise comment for any question by pressing the button, which results in a standard browser text area to appear. Figure 7.2 illustrates both features.

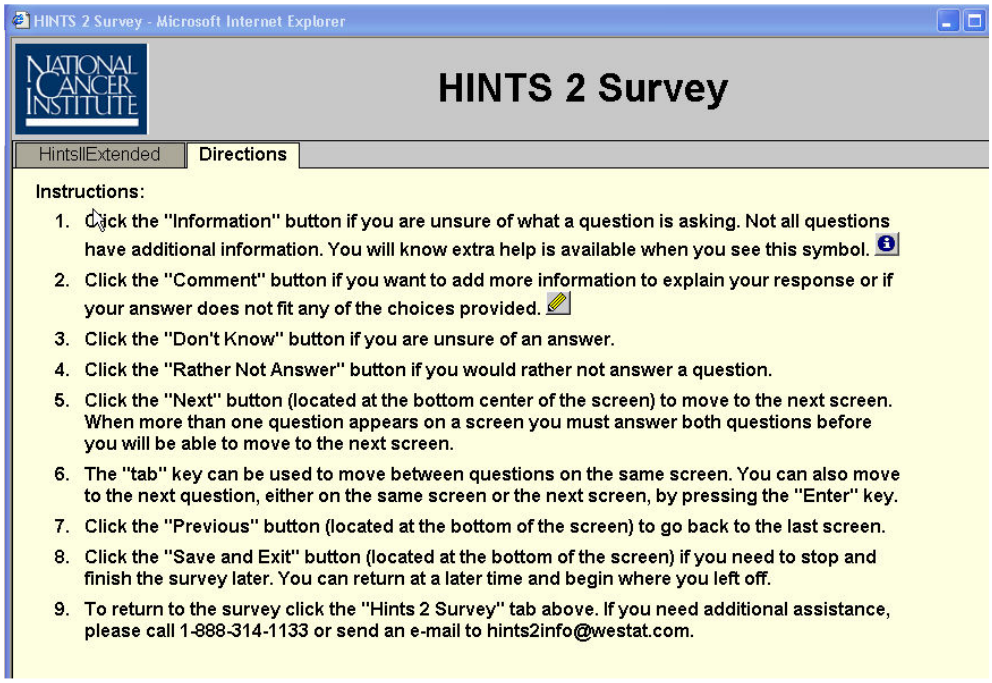
A final note about the web-rendered Blaise instrument is what *isn't* there. A Blaise IS instrument may be configured with a standard 'progress bar.' However, because the survey is relatively lengthy and being administered to volunteers among the general public, Westat determined that the slow progress of the 'progress bar' might be discouraging to the respondent. Therefore, we disabled it.

Figure 7.2 – Blaise IS rendering of comment area and context-sensitive help



Overall on-line help was always accessible to the respondent by clicking on the 'directions' tab. The on-line help feature, illustrated in Figure 7.3, was implemented as a Blaise parallel block.

Figure 7.3 – On-line help in Blaise IS



8. Blaise data storage

The simplest approach for storing Blaise data is in a single Blaise database containing all cases. Unfortunately, this option would require the Blaise database to be located on the relatively insecure internet-accessible web server. Because of our need to assure privacy and security, we implemented a more sophisticated approach. Westat deployed a technique that has become routine for many Blaise for Windows implementations – creating a separate Blaise database for each subject and storing the Blaise database in the larger SMS database. The Blaise database is packaged as a WinZip file and stored in a BLOB field in the SMS database (SQL Server data type Image). To support database communication, Westat developed a database access middle layer implemented as a COM DLL. All ASP programs called appropriate middle layer methods to read and write to SQL Server.

After a respondent successfully authenticated, the system determined whether it was a restarted case. If so, the existing Blaise database was retrieved from the SMS database and 'unzipped' on the web server, before launching Blaise IS. If it was the respondent's initial login (a new case) then a new Blaise database was created and the appropriate data preloaded using the Blaise Component Pack (BCP).

When the respondent terminated the session, either completing or suspending the interview, the system extracted the status of the case (for example, 'complete' or 'partial complete') from the Blaise database, created a WinZip file from the Blaise database, then called the appropriate function within the middle layer to update the SMS database.

Abandoned web sessions presented a tricky problem. An abandoned session occurs when, for example, a user closes the browser, powers down the computer, or navigates to another web site prior to either completing the interview or pressing the 'Save and Exit' button. For an abandoned session, the program to update the SMS database is never triggered. To resolve this, Westat implemented a 'sweeper' program, developed as an ASP program, which was manually run daily. The program scanned the web site for abandoned sessions, and for each one, performed the same functions as ordinarily executed for a normally terminated session.

9. Data security and privacy

Westat is committed to maintaining the security of our computer systems and the data we collect. Security is particularly important in the web environment which by its nature is more open to attack. In addition, we are equally committed to privacy of data - minimizing access to the survey data or associating survey responses with an individual. The development team took the following precautions to protect the security of the data and help maintain respondent privacy.

- Access to the web survey was only permitted after the entry of a username and four digit PIN.

- The web site was accessed by Secure Socket Layer (SSL), the industry-standard 128-bit encryption method. SSL encrypts data between the client computer and the web server.

- The web session time-out interval was set to 60 minutes. After the session times out, the respondent must re-enter their username and password. This

feature helps reduce the possibility that an unintended person can view or alter survey responses if the respondent temporarily leaves the computer unattended.

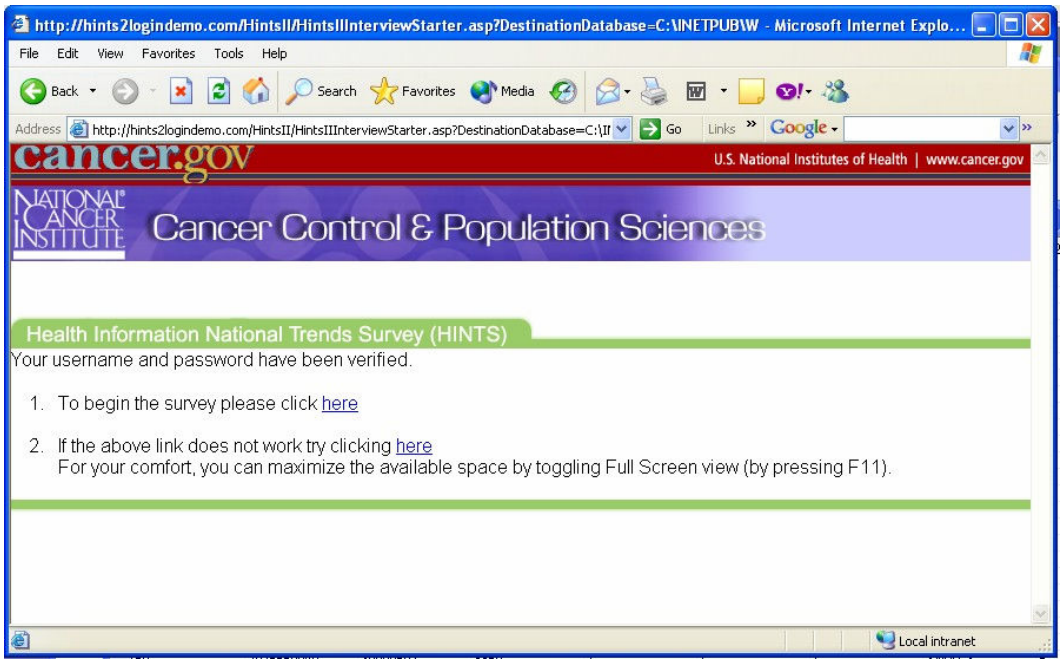
- Generally, the Blaise data was stored on the web server (the least protected area of the Westat network) only for the duration of the survey. Upon the end of the web session, the survey data was stored in the password-protected SQL Server database in a separate zone in a WinZip file.

10. Blaise IS challenges

We experienced a few miscellaneous challenges presented by the deployment of Blaise IS for web interviewing.

- **Pop-up blockers**: Blaise IS ideally launches in a separate browser window. However, when the respondent has an activated pop-up blocker, by default, Blaise is prevented from opening a new window. The standard Blaise IS implementation detects a pop-up blocker and displays a special screen which provides the user with alternatives for bypassing the pop-up blocker or opening Blaise in the same window. For a general population of computer users, the screen may cause confusion and for some may result in the respondent's inability to participate in the survey. Westat altered the standard Blaise screen in an attempt to make launching the instrument more seamless. We added our standard project banners and made no mention of 'pop-up blockers' in the text. Figure 10.1 illustrates the Westat customized screen that is rendered when a pop-up blocker was detected. Option 1 provides a 'link' as opposed to a 'button' as an alternate way of launching a new browser window. Many pop-up blockers will launch a new window based on a link but not a button. If the respondent is still not able to access the survey, Option 2 launches the instrument in the same browser window. The standard browser navigation remains visible but we provide a gentle suggestion for the respondent ('by pressing F11') which would hide the standard browser navigation.

Figure 10.1 – Special screen when pop-up blocker is detected



- **Browsers other than Internet Explorer (IE)**: Because the HINTS survey was be completed by a randomly chosen population, Westat could not control the type of browser that would be used. Blaise IS was designed to work well with most standard browsers, including Internet Explorer, Firefox, AOL, Netscape Navigator, and the Mac browser. Prior to the final HINTS deployment, Westat tested Blaise IS with many of the leading browsers. We discovered several severe flaws in the rendering of the interview in non-IE browsers. When Statistics Netherlands was able to reproduce the flaws, they quickly addressed the major problems and released an upgraded version of the software. This corrected most of the problems. Further Westat testing revealed just a few minor cosmetic issues in some non-IE browsers, but we determined that none would hamper a respondent's ability to complete the survey.

- **Scalability**: Before launching the system, Westat performed a stress test of Blaise IS to determine if there was a degradation of system speed with many simultaneous users. The test was conducted with a beta release of the software and with one Blaise database for all cases on the web server. We had 15 people from within the company access the HINTS instrument at the same time. The system performance slowed dramatically as people went farther into the interview. After about 15 minutes, the system's response time was aggravatingly slow, most of the time more than 20 seconds; many of the responses 'timed out' (response time out was set at 30 seconds). However, based on the anticipated survey volume, we estimated that simultaneous use would not approach this volume and generally, the performance would be acceptable. We are in the process of evaluating alternate configuration settings, as well as new features in more recent releases of Blaise IS, in the expectation that these will significantly improve scalability.

11. Central Study Management System

The central Study Management System (SMS) controlled and tracked responses for all cases through all phases of the study. The SMS consisted of several computer components that interfaced with the central SMS database.

- **Notifications**: The team developed several programs in Visual Basic to generate e-mail and letters to communicate with the respondents. The programs interfaced directly with Westat's e-mail servers to send them immediately. For letters, the programs produced ASCII delimited files which were processed into letters using Microsoft Word Mail/Merge capability. The team prefers this approach (as opposed to embedding the content within programs) because the content of the letters can be maintained as standard Word documents, which are more easily maintained.

- **Help desk**: Westat staff required a tool to support all study operations and to answer respondent questions and concerns. The team deployed an intranet site developed in ASP to support all contacts with the respondent. The web site provided the following features:
  - Flexible respondent search on any combination of name, status, phone number, and e-mail address,
  - Status history of the case,
  - Review of the notifications (e-mail and letters) sent to the respondent and the ability to request new notifications,
  - Username and password lookup, and
  - Address, phone number, e-mail address lookup and update.

- **Interface with the TRC**: The SMS interfaced with the TRC via file based communication. To support central reporting, the TRC provided the central management system with data about completed screeners and extended interviews. The SMS provided the TRC web cases that did not respond within the designated period. The team developed Visual Basic programs to generate and process the files.

- **Interface with Accounts Payable**: Some of the respondents who completed the extended interview received an incentive payment. We developed a Visual Basic program to interface with the standard Westat Accounts Payable system. The program extracted records from the central database and created a standard ASCII file to pass to the Accounts Payable department.

- **Reporting**: Westat developed several reports to analyze response rate and help control the study operations. The reports were developed in SAS and deployed within an automated web report portal environment. The web report portal is an ASP internet site that enables programmers to deploy SAS reports in SAS/Intrnet without having to develop web programs or configuring SAS/Intrnet. Within the report portal, the user is able to choose between HTML, PDF, RTF, and Excel for output format. The SAS programs access the integrated SMS database in real time, using SAS/Access to OLE DB.

12. Future considerations

The development of the HINTS study was Westat's first use of Blaise IS. We expect improvements in both the technology and our use of it. These items bear monitoring for the future:

- **Blaise OLE DB Interface**: With the Blaise OLE DB Interface (BOI), Blaise can store data directly into a database in interview mode. This has great promise for the future and may simplify the technical model used for the HINTS study. Survey data would no longer reside in a separate file (a Blaise database) during the interview. By interfacing directly with SQL Server, the survey data is always current in the database. The complex infrastructure (described above) to detect the end of a web session, update the SMS database with the Blaise data as a WinZip file in a BLOB field, and update status information in the database will no longer be required.

- **Blaise IS Scalability**: The results of the stress test conducted by Westat were a cause for concern. Blaise IS performance was adequate for the simultaneous use required for HINTS. Nevertheless, a development team should proceed cautiously for a new study that anticipates greater simultaneous volume. Statistics Netherlands has been aware of the Westat stress test results, which was conducted while Blaise IS was still a beta release. They have made subsequent improvements to the initial production release (version 4.7) and have announced further improvements for the upcoming version 4.8 release. Westat plans to conduct further scalability testing.

- **Common Blaise Code**: For practical reasons, Westat deployed two separate versions of the Blaise programs for the two environments, CATI and web. Having one common set of programs that operate in both environments is a better development model. The majority of the Blaise code is identical between the environments. Once we have more experience with Blaise IS, Westat will evaluate this approach for future multi-mode surveys.

13. Conclusion

The HINTS study protocol consisted of a large multi-mode survey and a complex management system to support the survey and operations. Blaise proved to be an excellent choice for development of the instrument. Substantially the same Blaise programs were deployed in both the CATI and web environments. Blaise has been used many times in our CATI environment and has met all our expectations. Moreover, the Blaise suite of tools consists of many tools that facilitate its integration with a large system. There were a few challenges using Blaise IS, but none proved to be major obstacles for the HINTS study. We gained valuable experience using Blaise for multi-mode web surveys. Westat looks forward to future enhancements in the Blaise IS technology to combine with the HINTS experience to implement future studies.

14. References

O'Reilly and Ray (2005): Web Data Collection with Long and Complex Questionnaires, Proceedings of the Federal Conference on Statistical Methodology.

Session 2.3: Multi-mode

Disparate Modes and Their Effect on Instrument Design
Mark Pierzchala
(Mathematica Policy Research, USA)

Questionnaire Design in Blaise for a Multimode Survey
Hilde Degerdal & Jan Haslund
(Statistics Norway)

Distributed CAPI Interviewing with Blaise® IS for the Web for the Adolescent Medicine Trials Network for HIV/AIDS Interventions
Rick Mitchell
(Westat, USA)

Blaise IS and Accessibility
Jim O'Reilly
(Westat, USA)

Disparate Modes and Their Effect on Instrument Design

Mark Pierzchala (Mathematica Policy Research, USA)

1. Introduction

Multimode surveys are increasingly common for several reasons including attempts to increase response rates, lower costs, and accelerate data collection. As practitioners of multimode surveys, we concern ourselves with many issues, but few are as important as the production of the electronic (Blaise) instrument. In a multimode survey we may have not only two or more modes, but different question forms for the same item, across modes. This paper elaborates on the concept of *disparate modes* introduced in Pierzchala et al (2006) and their impact on two instrument design methods; *unimode design* and *generalized mode* design. The paper is based largely on the experience Mathematica Policy Research, Inc. (MPR) gained on three complex multimode surveys. The words *instrument* refers to the electronic (Blaise) instrument for data collection or capture.

2. Disparate Modes

In the 2003 National Survey of Recent College Graduates (NSRCG) that MPR conducted for the National Science Foundation, nearly 50 percent of the items had a different question form for at least one of the paper, web, or Computer Assisted Telephone Interviewing (CATI) modes and some items had three question forms (Pierzchala et al. 2004). Some surveys, however, display little difference in question form across modes. For example, the National Beneficiary Survey (NBS) sponsored by the Social Security Administration with CATI and CAPI modes, and the Kauffman Firm Survey (KFS) sponsored by the Ewing Marion Kauffman Foundation with web and CATI modes, exhibit little difference in question form between modes (Pierzchala et al. 2006). An example of different question forms would be an item asked as a SET (code all that apply) question on paper and a series of Yes/No questions in CATI.

We consider four common modes of survey administration. Table 1 shows key differences and similarities between modes in how the respondent receives, processes, and communicates a response to a question. Survey methodology literature uses summaries of this sort to understand differences in responses to same questions across modes.

Table 1: *Attributes of Modes of Surveys That Affect Cognition and Response*

Aspect / Mode	CATI	CAPI	Web	Paper
Presentation	Aural	Aural	Visual	Visual
Transmission of response	Spoken	Spoken	Typed	Written
Segmentation of form[1]	Segmented	Segmented	Varies	Not segmented
Dynamic/ Passive medium[2]	Dynamic	Dynamic	Dynamic	Passive
Administration	Interviewer	Interviewer	Self	Self
Pace	Respondent / Interviewer	Respondent / Interviewer	Respondent / Computer	Respondent
Communication channel	Telephone	In-person	Browser	Paper

[1] Segmentation refers to seeing only one or a few questions at the same time on a computer screen, as opposed to seeing the entire questionnaire at once on paper (Groves et al. 1980).
[2] "Dynamic" refers to a computer-assisted mode, and "passive" refers to other modes.

The analysis of interest is in instrument differences that go beyond question statement. An analysis of instruments from several multimode surveys found that three of the dimensions in Table 1 are responsible for nearly all the differences in question form, and in operation, between modes. These dimensions, called *critical dimensions*, are (1) Aural versus Visual presentation, (2) Self- versus Interviewer-administration, and (3) dynamic versus passive medium. Table 2 summarizes the degree of disparity between modes based on this analysis.

Table 2: *Degree of Disparity Between Common Modes of Survey Administration*

Mode Combination	CAI (CATI/CAPI)	Web	Paper
CAI (CATI/CAPI)	-	2	3
Web	2	-	1
Paper	3	1	-

The paper/web combination has only one degree of disparity because the paper mode is passive and the web mode is dynamic, but both modes are self-administered and visual. The web/CAI combination has two degrees of disparity because the only critical dimension they share is their dynamic nature. The CAI/paper combination is the most disparate because there are differences in all three critical dimensions.

In practical terms, producing a single electronic instrument that works across modes becomes more difficult the larger the degree of disparity and the larger the number of modes. This paper illustrates how differences in each dimension affect the production of one instrument.

3. Multimode Instrument Design Traditions

Multimode surveys have two instrument design traditions. Both seek to provide the same stimulus to respondents across modes. This paper interprets the *unimode* design (Dillman 2000) as maintaining question form and attributes Don't Know (DK), Refusal (RF), and EMPTY across modes. The NBS is a CATI/CAPI study where the modes are identical in the subject matter questions, the use of the dynamic Blaise system, and the use of interviewers. As a result, its design was naturally unimode. In the KFS, the CATI and web modes differ in two critical dimensions. However, MPR executed a unimode design through compromises in implementation between the modes (Pierzchala et al. 2006).

Generalized mode design seeks to optimize survey implementation in each mode (De Leeuw, 2005). It strives for *cognitive equivalence* across modes. It asserts that, because of the different ways the respondent processes information between modes, this equivalence may at times be better achieved with different question form across modes. This paper also considers differences in operation of the instrument (for example, handling edits or conducting a followup), as an aspect of generalized mode design. The 2003 NSRCG used a generalized mode design in its paper, CATI, and web modes.

In all three surveys, MPR put together one instrument and one database to handle the different modes. CAI and data entry modes were fielded in Blaise Windows and the web mode was implemented in a browser using the C2B software from CentERdata. MPR finds that, the greater the degree of disparity between modes, the more dramatic the impact on instrument design can be. Thus, a CAI/paper survey may be more difficult than a web/CAI survey to field in a unimode fashion.

4. Impact of Differences in Critical Dimensions on Instrument Design

Next we summarize mode differences along the three critical dimensions. This summary is based on an inspection of the three Blaise instruments mentioned in the introduction. In the tables below, the symbol => should be interpreted as a tendency. The notation => UMD or => GMD implies that the aspect of the critical difference tends to be accommodated by unimode design or tends to require a generalized mode design. Where a generalized mode design is indicated, a unimode designs may still be accomplished through compromise in implementation in one or more modes. Differences that require only a change in text from one mode to another are marked as tending to unimode design.

4.1. Aural Versus Visual Presentation of the Survey Question and Responses

The terms *aural* and *visual* refer to how the respondent receives the question statement and any response options. The *aural* method requires the respondent to not only hear and understand the question and choices, but also to remember them.

Table 3: *Design Implications of Aural Versus Visual Presentation*

Question Part or Type	Critical Dimension of Presentation		Instrument Design Implication
	Aural	Visual	
Any question / single question	Interviewer accent or pronunciation can intefere; respondent may not speak study language; speech is linear; pace depends on personal interaction; the respondent must remember information.	Respondent must be literate to the level of the question statement and to the complexity of the subject matter.	-Shorter questions, simpler words in Aural mode. -There may be question wording differences between aural and visual modes. **=> UMD**
Complex sentences	It can take too long to say, and the respondent may not be able to understand all of it without repeating.	Comprehension depends only on the respondent, who can re-read text more easily; you can place larger cognitive burden than for aural modes.	- Break up one question into two or more questions in aural modes. **=> GMD**
Series of related questions: e.g., Yes/No items	Stem is (usually) expressed just once. Treat one item at a time and in sequence.	Respondent can process more than one item at once; respondent can visually navigate and compare items; stem is always visible; respondent can answer in any order.	-Lead-in explanation may be necessary for aural -The question stem may be re-displayed for the interviewer, for every item even if not always said. **=> UMD**
Enumerated (code-one) choices for non non scale options (such as educational attainment).	Limited number of choices can be spoken at one time to limit the respondent's memory task.	Many choices may be presented at one time.	-Aural modes may break up the question into two or more branching questions **=> GMD**
SET (code-all-that-apply) choices for non scale options (such as educational attainment).	The question may be rendered as a series of Yes/No questions.	Many choices may be presented at one time.	-Use of Yes/No for all modes **=> UMD** -Alternate forms between modes **=> GMD**
Conversational norms	Wording is more conversational.	Wording may be more formal	-There may be question text differences. **=> UMD**
Open ended	Answer lengths tend to be short.	Respondent may provide longer, more detailed answers	-Provide Blaise OPEN instead of STRING for all modes. **=> UMD**

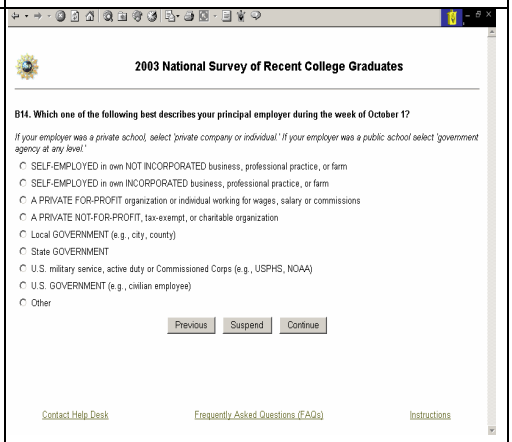
An example from the 2003 NSRCG illustrates a few differences in question form across modes. The question, concerning description of employer, was asked as an

enumerated question on paper and on the web, but was implemented as a group of four branching questions in CATI. The reason for the difference in question form between modes is that the statement of the response choices is long and complex. It would be impossible for a respondent to remember the meaning of all choices if presented aurally.

Figure 1: *Question B14 in Paper, Web, and CATI Modes from the 2003 NSRCG*

<table>
<tr><td>Paper</td><td>Web</td></tr>
<tr><td>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</td><td> 
</td></tr>
<tr><td colspan="2">CATI</td></tr>
<tr><td>B14_1. Was your principal employer during the week of October 1, 2003. . .
1. a private company or individual (go to B14_3)
2. a government agency at any level (go to B14_4)
3. were you self-employed (go to B14_2)
4. did you work for some other type of employer (go to B14_specify)
DK, RF

B14_2. Were you. . .
1. Self-employed in your own NOT INCORPORATED business, professional practice, or farm
2. Self-employed in your own INCORPORATED business, professional practice, or farm
DK, RF</td><td>B14_3. Was that . . .
1. A private for-profit organization or individual paying your wages, salary or commissions
2. A private not-for-profit, tax-exempt, or charitable organization
DK, RF

B14_4. Was that. . .
IF EMPLOYER WAS A SCHOOL: [State schools, colleges, universities are “state government” and schools run by local school districts are “local government”].
1. Local government, such as city or county government
2. State government
3. U.S. military service, active duty or commissioned corps, such as USPHS or NOAA
4. U.S. government as a civilian employee
DK, RF</td></tr>
</table>

4.2. Self Versus Interviewer Administration of the Survey

Both self- and interviewer-administered modes of interviewing have advantages and disadvantages. With an interviewer, there can be comprehension problems between the interviewer and the respondent. For sensitive questions, a respondent may be influenced to give *socially desirable* answers (O’Reilly et al, 1994). On the other hand, interviewers secure cooperation. They are skilled at persuading the sampled person to respond and can keep a respondent engaged until the end of the interview. In addition, an interviewer is trained in the data collection software and instrument and can perform complex actions such as error correction and navigation that one would not expect in self administration. However, interviewer modes are usually more costly. Table 4 shows how aspects of this critical dimension influcence choice of unimode versus generalized mode design. Any difference in the operation of the instrument (for example, handling edits) is labeled as tending to a generalized mode design.

Table 4: *Design Implications of Self versus Interviewer Administration*

Question Part or Type	Critical Dimension of Administration		Instrument Design Implication
	Self	Interviewer	
Social desirability for sensitive questions	More likely to give honest answers	Interviewer presence may elicit socially desirable, but not correct, responses	-There may be special modules and/or software in for sensitive sections, as in Laptop CASI. **=> GMD**
Training	Untrained	Trained	-Interviewer can resolve consistency edits and navigate => **GMD**
Probes	Not possible	Interviewer can probe within same question form.	-More text in interviewer mode. => **UMD**
Instructions and optional text	Not possible or limited	Can expect the interviewer to follow instructions or to read optional text if necessary.	-More text in interviewer mode. => **UMD**

An example of a difference in instrument design based on self- versus interviewer-administration is in the application of a consistency edit. Such an edit compares data between two or more fields and flags seemingly inconsistent values. The resolution of such an edit is a high-level action that requires the interviewer to be able to communicate the error to the respondent, navigate to the field in error, make the correction, and get back to the point of the interviewer, all the while keeping the respondent engaged in the interview. When the edit involves widely separated fields (especially on widely separated browser screens), it is not usually appropriate for the web mode, but is invoked more commonly for interviewer-administered modes. Figure 2 shows the Blaise source code for such an edit.

Figure 2: *Edit Statement Blaise Source Code for Interviewer-Administered Mode*

```
IF AdministrationMode IN [CATI_, CAPI_, Editing] THEN
  IF WeeklyPayroll/(NumEmployees * 40) < 3.0 THEN
    ERROR "The hourly rate seems too low."
  ENDIF
ENDIF
```

The edit as specified above would be applied only in interviewer-administered modes or in post collection processing. It would be bypassed for web mode or for data capture from paper. However, this kind of consistency edit has been applied in a web survey at MPR using "WebSurv", its own in-house web survey software.

4.3. Dynamic Versus Passive Data Collection Medium

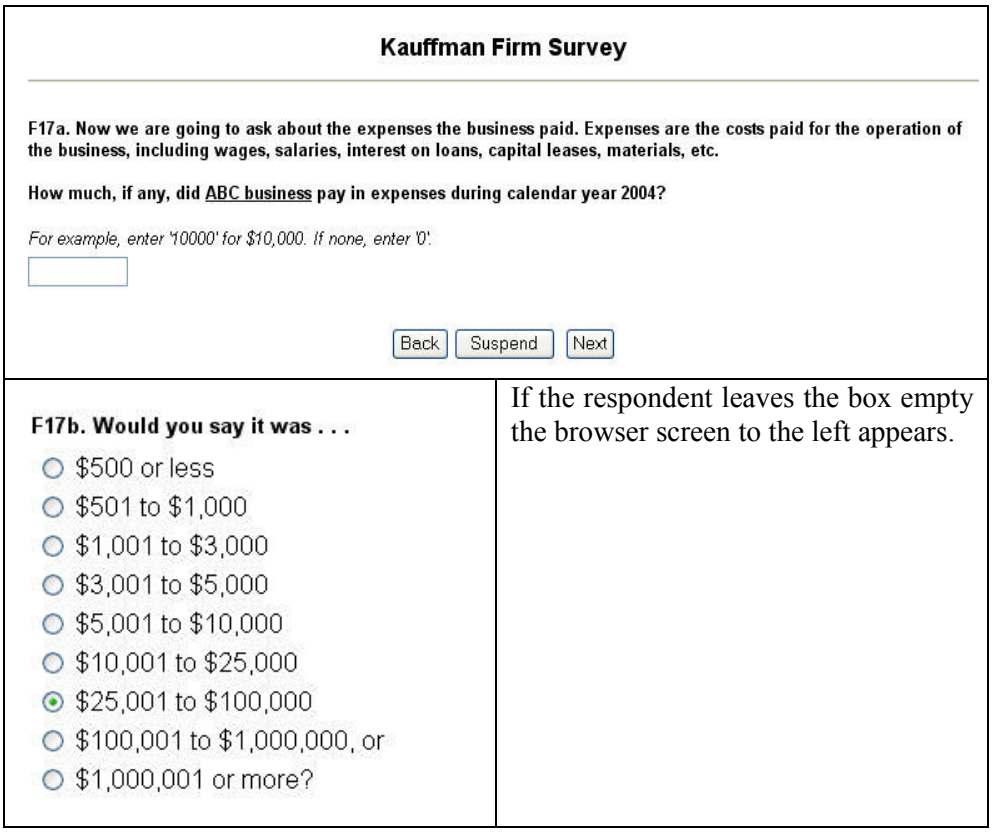
A *dynamic* medium implies the use of a computer-assisted system where text statement, routing, appearance, computations, language, sampling, and edits can all be executed during data collection. A *passive* system usually refers to paper collection, but it can also refer to a static HTML page used either in an email survey or in some web survey systems. Dynamic systems provide far more flexibilty than passive ones in most respects; on the other hand, they rarely provide the overview and context that paper questionnaires provide. The concept of *space* is vastly different between the two kind of media. A dynamic medium offers a virtually unlimited number of pages for an instrument, whereas a paper questionnaire is limited to a finite, usually much smaller, number of pages. The much more constrained space of the passive medium, in conjunction with the fact that all the questions can be seen at one time, often influences question form.

Table5: *Design Implications of Dynamic Versus Passive Medium*

Question Part or Type	Critical Dimension of Medium		Instrument Design Implication
	Dynamic	Passive	
Space in terms of number of pages or screens	Unlimited browser or Windows pages	Limited number of paper pages	Dynamic medium allows for more questions or probes per same instrument **=> GMD**
Space within a page or screen	Browser: respondent controlled Windows: Controlled by survey organization Either: Less space than paper questionnaire.	Paper page has massive pixel density. Great freedom in display of text, instructions, directions, and other elements.	Design possibilities within a page are far greater for a paper questionnaire than for a dynamic medium **=> GMD**
Freedom of movement	Software limits user movement.	Great freedom.	Paper gives better overview, respondent can see where to go. **=> GMD**
Required answers	Can require answers, but may allow EMPTY.	Cannot require answers.	Pasive mode may leave some answers blank **=> UMD** with complex conditions, or **=> GMD**
Probes	Can be provided in a follow-up question.	No follow-up questions are possible.	More probes in dynamic modes **=> GMD**
Skips	Automatic and enforced (taking into account allowed EMPTY)	Arrows on page and other instructions, respondent may not follow skips.	In paper, on-route data may not be answered, and off-route data may be provided **=> UMD** (with extra programming)
Consistency Edits	Possible to compare two or more values during data collection	Not possible to compare two or more values during data collection.	Disable edits in passive modes during capture **=> GMD**
Responses	Difference between code-one and code-all can be enforced.	Respondent can answer more than one choice for a code-one.	Define question as code-all for all modes, but enforce code-one in dynamic through edits. **=> UMD**
Question statement and responses	Can customize text dynamically based on previous answers.	Cannot customize text.	Text-only differences between modes **=> UMD**
Question and response order	Can rotate or randomize questions or responses.	Cannot rotate or randomize, except with custom printing.	Anchor text for data capture **=> UMD**

The following example illustrates the relative power and the limitations of dynamic and passive media. In collecting monetary amounts in the KFS, the dynamic mode allows probes if the respondent does not give an exact amount. Figure 3 shows this in the web mode; the CATI mode works the same way.

Figure 3: *Collecting Monetary Amounts in the KFS*



This pattern of two screens, for each financial item, appears several times in order to collect expense information for a business. A respondent who gives exact values will never see the probes. The KFS does not have a paper mode. If it did, however, this kind of probing would not be possible. Furthermore, a paper version would display the expense information as one data collection structure on one page.

5. Question Attributes and Consistency

Question attributes such as Don't Know (DK), Refusal (RF), or EMPTY are usually presented to the respondent differently between modes. A common web and paper convention, for example, is to not provide explicit DK or RF options, but to allow an EMPTY answer. On the other hand, CATI and CAPI usually require the interviewer to answer with a DK or RF if a valid response is not possible.

Handling these conventions differently for each mode complicates specification and programming. Branching conditions have to take into consideration the mode of administration. MPR has been able to easily implement these additional conditions, but with a different programming strategy, as shown in Figure 4.

Figure 4: *Handling DK, RF, and EMPTY Differently Between Modes*

```
FIELDS
  NumEmployees "How many employees do you have?"
               : 0..1000, DK, RF, EMPTY
. . .
RULES
  NumEmployees
  IF AdministrationMode IN [CATI_, CAPI_, Editing] THEN
    NumEmployees <> EMPTY "Answer required."
  ENDIF
```

In the example above, the DK and RF would not be displayed in the browser by use of configuration options; technically they are available but the respondent does not have a way to select them. The EMPTY attribute, not typically employed for CAPI/CATI surveys, is used in multimode surveys involving web or paper modes. The Blaise language can allow this differential mode treatment at the item level through edits added in the RULES, for every item, in CAPI or CATI. The extra work to implement these validation edits is minimal. More important, what has changed in the self-administered modes is the concept of record validity. We are simply more willing to accept data faults in web and paper modes than in CATI or CAPI. The important things are to (1) get data in a mode in which the respondent is willing to provide it, (2) get the data less expensively, and (3) accelerate data collection schedules. We do not wish to dissuade a respondent from completing the questionnaire. A good faith respondent in web or paper modes will not cause problems. One who skips questions, or makes routing errors, can be treated in a followup, as detailed below.

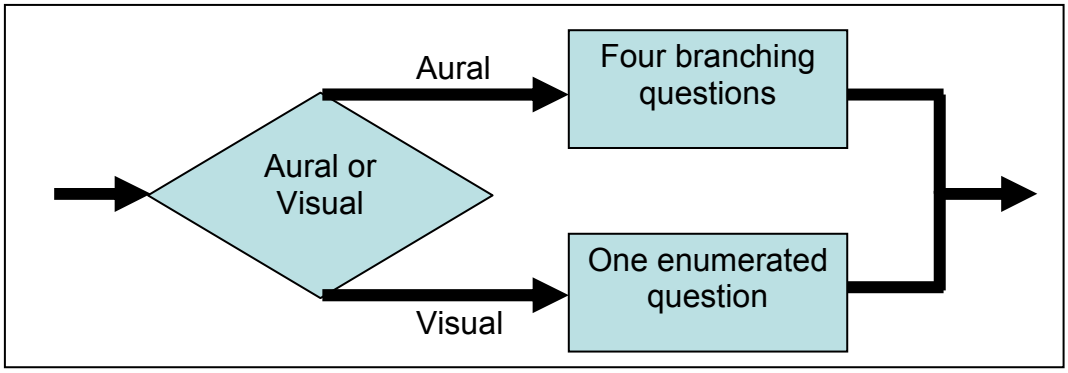
6. Special Capabilities of a Multimode Instrument

Special capabilities of a multimode instrument include handling disparate question forms between modes in one instrument, switching modes in mid-case, and critical item followup.

6.1. Handling Different Question Forms Between Modes in One Instrument

Where there are two (or more) question forms in one instrument, it is necessary to branch to the specific question form depending on mode as shown in Figure 5 for the 2003 NSRCG question B14 (Figure 1 above).

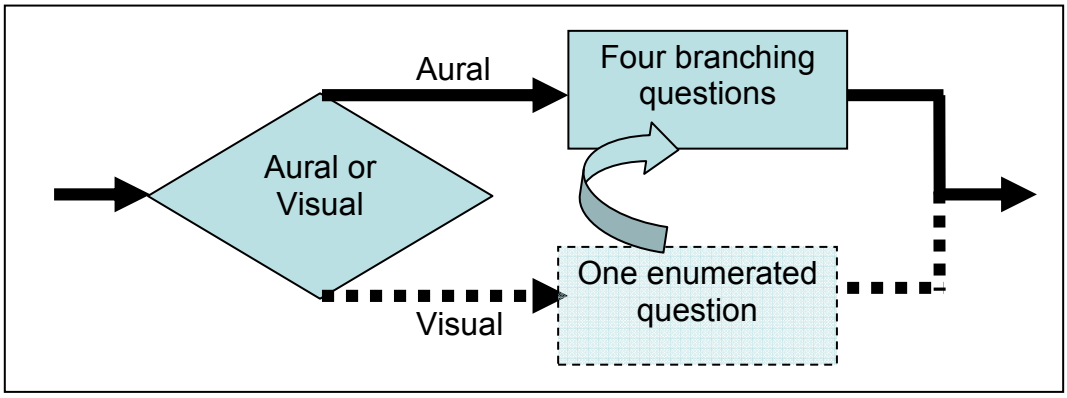
Figure 5: *Branching Question Forms Between Modes*



6.2. Mode Switching Within a Record

Sometimes it is necessary to switch modes in a case (for example to complete a case in CATI that was started in web mode). Where there are different question forms between modes, it is necessary to transfer data from one question form to the other as shown in Figure 6.

Figure 6: *Data Transfer Between Question Forms When Switching Modes*



This item-level data transfer is done on the fly as the case is brought up in the second mode. This way there are no issues with branching, and the interviewer is free to navigate to the current point of the interview.

6.3. Critical-Item Followup in the Blaise Windows Platform

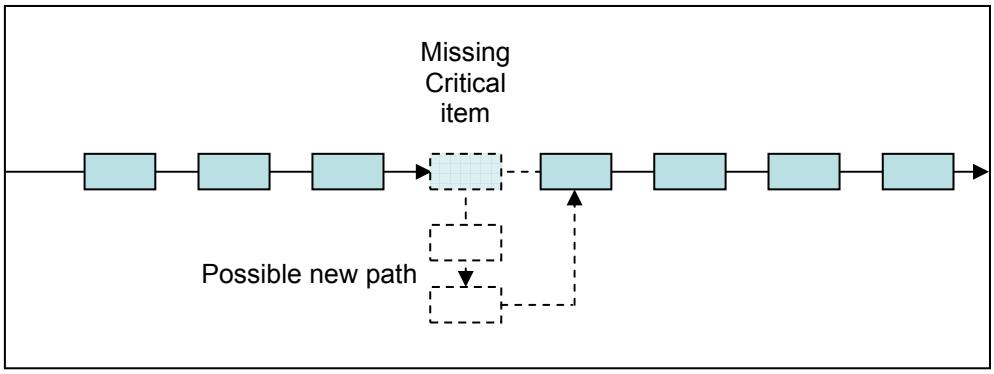
A case completed in any mode may have a severe data issue. For example, critical items may have been answered with a DK or RF, or they may have been left EMPTY. Depending on the survey, and at the discretion of the client, it is possible to follow up these cases. MPR conducts this followup in the Blaise Windows platform because of its native navigational capability. A key part of this capability is the Blaise selective checking mechanism that recalculates flow every time a data value is entered. It is desirable to maintain the data collection path and the original values in the context of the originating data collection mode, in other words, to divorce platform from mode. The item followup is within record and within place, in the original data record. Figure 7a shows source code that marks an item for critical-item followup.

Figure 7a: *Marking an Item for Critical Item Followup*

```
RULES
  IF (NumEmployees <> RESPONSE) AND (FollowUp = Yes) THEN
    CHECK
    ERROR "Your response to this question is important.
           Can you tell me how many employees you have?"
  ENDIF
```

This method of followup has several advantages. It puts the data value directly in its place in the data record. If the value results in a new path opening up, the data along this newly opened path can be collected as part of the followup. This includes the collection of other critical items that might be found on the new path (see Figure 7b). Finally, it avoids having to put together a different system (on paper or by computer) to collect these critical values and then merge them later.

Figure 7b: *Illustration of Critical Item Followup in Blaise Windows*



7. Discussion

MPR has been able to implement both unimode and generalized mode designs into a single Blaise instrument operating on one database. For some surveys, it has been appropriate to use the browser and Windows computer platforms simultaneously. MPR uses them differently, and they have different capabilities and users. Concepts of what constitutes a page, and how pages should be arranged, are also different. Because the screens in both platforms are computer generated, not hand drawn, this dual platform capability comes at very low cost.

The higher cost in producing a dual- or triple-mode instrument comes from handling different question forms and from accounting for the additional cost of specification and programming when conventions between modes are different (for example, DK, RF, and EMPTY are handled differently). From the standpoint of ease of specification, programming, and cost, unimode design is preferred.

The choice between unimode and generalized mode design also involves measurement issues. It is true that the same question form in different modes may lead to different response patterns, and that in order to achieve cognitive equivalence, a generalized mode design might be preferred. However, in addition to the additional costs of implementation, the rendering of a survey question into two or more forms will also incur the cost of evaluating the different options. This evaluation would be carried out wherever there are different question forms.

8. References

De Leeuw, E. D. (2005). To Mix or Not to Mix Data Collection Modes in Surveys. *Journal of Official Statistics, 21*, 2, 233-255.

Dillman, D. A. (2000). *Mail and Internet Surveys: The Tailored Design Method*. New York: John Wiley & Sons.

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.

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., Wright, D., Wilson, C., Guerino, P. (2006). Telephone Collection as Part of a Multimode Survey. Paper presented at the 2006 Telephone Survey Methodology Conference II, Miami, Florida (U.S.), January, 2006

Pierzchala, M., Wright, D., Wilson, C., Guerino, P. (2004). Instrument Design for a Blaise Multi-Mode web, CATI, and Paper Survey. Paper presented at the 2004 International Blaise User's Conference, Gatineau, Canada, September, 2004 (at http://www.blaiseusers.org\IBUCPDFS/2004/24.pdf).

Questionnaire Design in Blaise for a Multimode Survey

Hilde Degerdal & Jan Haslund (Statistics Norway)

1. The project

For the 3rd Continuous Vocational Training Survey 2006 in Norway (CVTS3) the data collection is conducted in a multimode design. The survey objective is to implement the CVTS3 according to the specifications agreed between Eurostat and EU Member States, in the CVTS Commission Implementing Regulation. Those specifications are extensive, e.g. There are extremely great demands on consistency of the data. The codebook is strict, and a lot of the fields don't allow missing. These specifications require a good tool for the collection process.

In addition to CVTS3 objective itself, Statistics Norway will provide a pilot addressing enterprises with 1-9 employees and one addressing new NACE categories. Further, Statistics Norway will implement new methodologies for CVTS3 i.e. web questionnaires. The latter will be one of the key elements in our design.

So, in addition to our survey objectives, we have two experimental objectives:
1. To study possible instrument effects of the combination of methods originally proposed in the project
2. To study how interviewers most efficiently can be used in business surveys

In the CVTS3 survey we will have two designs, one for large and one for medium size enterprises. It is planned that enterprises with more than 250 employees should be contacted by telephone and then interviewed by personal interviews, while smaller companies will be contacted by post and offered to respond on web or a paper questionnaire. In the first design there is no separate reminding strategy. In the second design, however, those who have not responded before the deadline will receive a postal reminder. Schematically these two designs can be described like this:

Design1:
Telephone contact ▸ Computer Assisted Personal Interviewing (CAPI)

Design2:
Postal contact ▸ Paper and Pencil or Computer Assisted Web Interviewing (PAPI/CAWI) ▸ Postal reminder

In addition we would like to compare these two designs with a third one that we believe might be a more cost efficient way of using interviewers in business surveys. This design can be described like this:

Design3:
Telephone contact ▸ Paper and Pencil or Computer Assisted Web Interviewing (PAPI/CAWI) ▸ Telephone reminder

The telephone reminder may cause an interview, and then we need a CATI-instrument.

All data is going to be analysed in one joined data file. We want the data manipulation to be as straight forward as possible. For CVTS, with its strict requirement of the data quality, it is convenient that all data are collected with the

same set of rules. That is one reason why we decided to make one questionnaire for all methods. The tool we have for making different instruments out of one questionnaire is Blaise.

2. Requirements

2.1. The requirements from CVTS Commission Implementing Regulation

Their list of controls for validations and consistency codes, requirements for no-missing and so on is a document of 13 pages. These claims from the committee are a challenge, especially since we know from earlier experience that a lot of the information is not easy for the respondent to find.

We really need to take those requirements in consideration during the work of making good data collection tools, in order to avoid too much struggle afterwards.

2.2. The layout requirements

In Statistics Norway we have clear guidelines for what a web questionnaire should look like. Gustav Haraldsen has made a manual on how to make good web questionnaires, and we want to follow those guidelines. It means very high focus on user-friendliness.

Figure 2.2.1. Layout

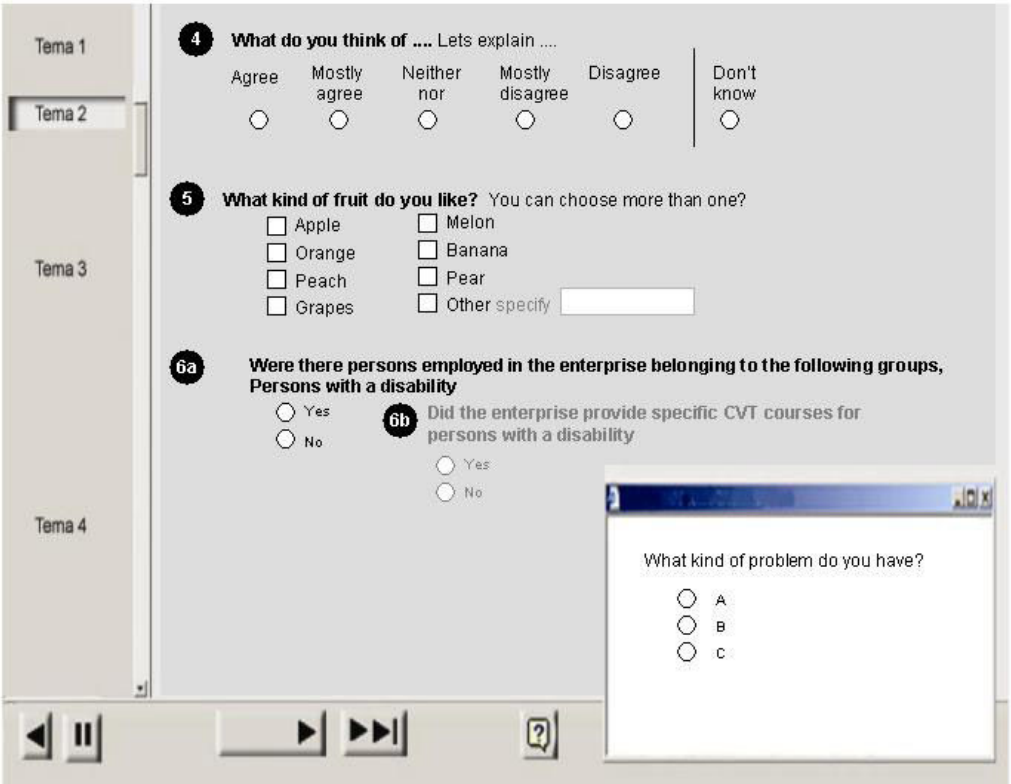


Figure 2.2.1 is a screenshot giving some examples of what kind of requirements - based on the guidelines) we have.

3. Organization and schedule

The work with the instruments was separated into two different subgroups in the project. A group of four persons, "the questionnaire group" had responsibility for the quality of the questionnaire, and what it should be like in the different modes. They should make a good basis for the programming of the questionnaire and the different tools. This group included some Survey methodologists from the division for Data collection methods.
A group of two more people, Jan Haslund and Hilde Degerdal, had more technical tasks. We should also realise the different instruments according to instructions given by the first group

A brief timeschedule:

Receiving the last instructions from Eurostat	December 2005
Making the basis questionnaire as a basis for Blaise programming	January 2006
Blaise programming, Blaise IS and CAPI	February
Data collection	15. March ->

Questionnaire group's tasks were
1. Translate the questionnaire into Norwegian.
2. Learn about the situation out in the enterprises. Such as who in the organization holds which kind of information that we would ask for, (if anyone at all). They call a number of enterprises to learn about it.
3. Adapt experience from earlier CVTS to avoid similar problems
4. Sort out what kind of information we need to ask for, and what we could retrieve from registers
5. Decide on which phrases to use to be understood similarly in the different sorts of enterprises
6. Decide on structure of questionnaire, taking into consideration what they learned about who could answer which questions
7. Decide on level of controls, and don't know-handling. This was difficult because there was a big difference between what kind of quality we want, and what they have learned about what kind of information the enterprises had.

When they had finished the steps 1-6 above, they had a scheme of questionnaire ready, with a draft text for the questions. Then we had a basis for starting the Blaise programming.

4. The Blaise work

4.1. Visit to the Blaise team

In 2005 we spent a few hours to look at the Blaise IS. The impression we got was that almost any kind of layout is possible to make in Blaise IS. If you make the right changes in the modelib and/or in the style sheets.

But it does not seem straightforward to find out how to build the different layouts. Hence we invited ourselves to Heerlen in January. We sent some examples of wanted layout in advance. (Figure 2.2.1). When we came to the Tim Carati's office the 23th of January, he had made solutions for us, that met most all of our requirements.

4.1.1. What we discussed – and learned

- The figure 2.2.1 shows that we want name of the Block as buttons in the left margin. This is not possible in Blaise 4.7, but will be in Blaise 4.8 if the blocks are parallel blocks.

- We want the field tag to have a certain layout. Specifying a gif-file as background in the style sheet biHTMLWebPAge.xsl solved this wish. This solution was implemented in the style sheet during our visit

- We had some concerns about scale questionnaires like quest 4 in the figure. The distance between the different answer categories should be equal, and not depended of the amount of text. It was also an issue that the Don't Know answer should not influence the visible impression of the scale. We knew that we should use the ScaleType for this type of question. And we learned that the distance between the categories is calculated out of 100% divided on number of categories. We also learned that using "no-wrap" on the text could destroy the equal alignment of the answer categories. The wanted separator before “Don't know” was implemented in the style sheet during our visit.

- We learned how to use the OtherSpecifyType, which met our wishes on question 5.

- An answer consisting of a number combined with Don’t Know gives radio buttons in front of the inputfield for the number as well in front of the Don’t Know answer. We did not want that. We got a solution where the radio button before the inputfield is hidden.

- The size of textboxes does not have to correspond with the length of the STRING-fields; it can be adjusted in the modelib by using a special field pane.

- A question with an answer that leads to just one follow-up question, the layout should be like in quest 6. The solution here was to use the MultiColumnType. Even though we are not able to have the first questions text above both columns, this is an adequate solution. The wish of having the follow-up question disabled until it is on the rule, is met by using a variable as a font code in the Field and answer text. It is first set to gray, and when the question is on the rule, it is recomputed into black.

- We also got the possibility to choose text or pictures on the footer buttons. We wanted an additional button - for End. And that was implemented as well. It was done by a change in the style sheet. It will be standard in 4.8.

- Due to pop-up blockers, a separate window for help-instructions and error messages, as shown in the figure, is not a recommended solution. The instruction text or error messages shows up as a separate line under the Field text at the actual question.
  - If a question has a help-text an ikon shows up beside the Field text. By clicking the icon the help-text shows. It is also possible to have the help as a separate html-document and link to an anchor in this.
  - For error messages involving more than one question, it is possible to put links in the error message, making it possible to jump to the other questions involved. We got this functionality that will be included in 4.8
  - The survey methodologists want the help text, or error message, to be a kind of guide helping you through the problems, using questions and answers. This is not possible.

- We also learned about storing of data, and to use a working data base. Saving is done by each new page, and by each "critical field". By unexcepted disruption of the session, the data from it is stored as uncompleted in the "working database". When the questionnaire is completed, the data is transfered to the main database.

- In Norwegian we have two languages, and we discussed different methods of choosing language. What we wanted was to have a choice together with the login, and that is possible.

- Ordinary Audit trail is not possible in Blaise IS. But we can get access to a lot of process data, using an additional Blaise data base, the journal.bdb. On every contact with the server, information will be written in the journal.

We also discussed other issues during our stay in Heerlen. We will just point out some of the issues here.

- Javascript disabled
- Screen resolutions
- Browser compatibility
- Consequences of Pop-up blockers for the whole questionnaire
- The Progress bar
- Other ways of answering questions, for example; calendars, gliders, scales to point in.
- How to fill in a questionnaire without a mouse. It is possible just to use the keyboard
- Tailor-made text in questionnaires and answers is of course no problem, but causes use of critical questions that can slow the performance.
- Lookup and coding
- Multimedia, like use of pictures can be implemented via html tags in the field-text

Our two days in Heerlen were extremely useful and gave us a lot of valuable advice. And for the rest of the process we appreciated the possibility of sending emails and getting quick answers from Tim Carati!

4.2. Making the questionnaire in Blaise

We had some plans on how to work to get the highest benefit of using one software for the different instruments. The questionnaire should be kept in one document as long as possible. Our intention was to start to author an ordinary questionnaire, as we know it from CAPI. We were aware that the different modes would require different error messages and level of how strict the checks should be executed. And of course we knew we would have to adjust the help-instructions according to mode of data collection; taking into consideration that some of the instruments are meant for professional data collectors (interviewers) and others are for amateurs. And we had an idea that the field text as well had to be adjusted after method of collection.

We planed to start the work like we normally do for our CAPI-surveys, and then to do some slight changes.

Well, it does not go that way. In end of January we had the basis for our work. It looked like a paper questionnaire, with headings and instructions for self-interviewing. It meant that the basis was more like the web-questionnaire than a CAPI-instrument. Anyway it was what we had, and we were in a strict time schedule, so we started the work.

We will now tell about the work and the solutions within the different sections in the Blaise Datamodel.

4.2.1. Types

We defined the types as usual. There was no need for many different types in this questionnaire. It was mostly YesNo, and amount of Nok, hours and persons. To customize it for web, we had to do some changes. We want the layout for some questions to be like quest 6 in fig 2.2.1. So we had to make a special type for those questions, MultiColoumn. Later, during the evaluation, we skipped this layout and used ordinary types for those questions. Some questions we needed the OtherSpecifyType, so we implemented this one. These questions were SET OF. We learned that we had to put the "SET OF" in the Type-section and not in the Field-section to get the wanted effect.

A lot of the types consist of numbers of many digits. To make the typing easier for the respondent, and thereby improve the quality of the data, we wanted those numbers to use the digit group option. Unfortunately this option did not work in the Blaise IS.

There was a lot of questions in the questionnaire that could have the answer 0. As an example the total number of participants of courses should be grouped by kind of arranger. If the enterprise did not use a certain kind of arranger, it is not obvious that the respondent has to type 0 in that cell. In the CAI-version it is easy to put an instruction to the interviewer "if none type 0". But what we wanted in the Web questionnaire was a separate box to tick "none". Using the REFUSAL answer solved that. We translated the word refusal into the Norwegian word for none. And then it shows up like this figure.

Figure 4.2.1.1

Figure 4.2.1.1 is a screenshot showing use of don't know and refusal

First the inputfield, the Don't know, and Refusal/none shows up vertically. As default we want the answer categories vertically, but in this case we had preferred it to be horizontally. We asked Tim Carati if that was possible, and it was.

Figure 4.2.1.2

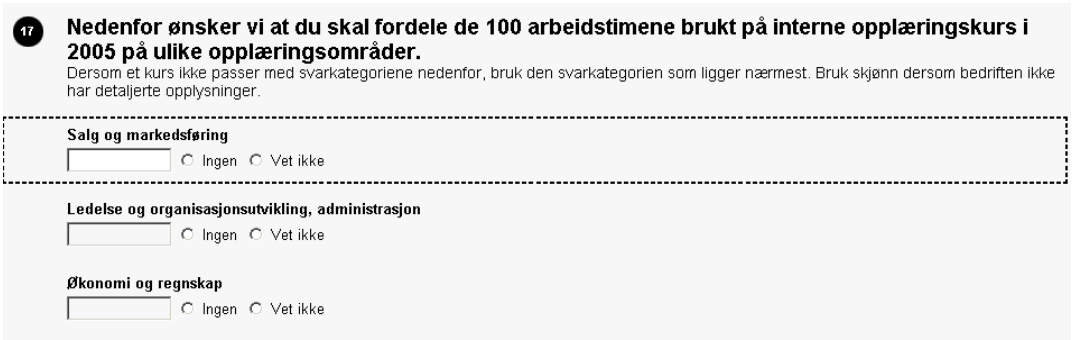


Figure 4.2.1.2 is a screenshot showing use of refusal as "none" ("ingen")

4.2.2. Fields

We wrote one field text like in the specification we got from the "questionnaire group". We used auxfields for headings and subheadings. To remove the input-field from those auxfields we defined a special fieldpane in the modelib. Those headings were fine for the web, but in the CAPI questionnaire they did not show unless you went back to them. So to make texts for transition between the different sections in the CAI-instrument we needed to do what we usually do, to put an additional text in start of the first question in the sections. There was also a need for making other adjustments in the text to customize it for the interviewers. That means both in the text to be read and in the instructions. Our solution of this challenge was to make two languages for the questionnaire, one “web”-language and one “cai”-language.

4.2.3. Rules

The part of the rules handling which fields to be on the route (the go-to-instructions) was straightforward. This part was also similar in all instruments. The challenge in the rules was the checks. Since we knew that some of the information was difficult to provide, we could not use hard errors. We had to use signals for all sort of controls. The " questionnaire group" discussed and tried to figure out which of the numberless controls from CVTS to implement in the different instruments. Meanwhile we wrote all of them in Blaise. We knew that we had to run them all one day, if not earlier so as a part of the revision after the data collection. All controls were written in a separate section in the document. At first we put them all as comments, and when we decided to include a control in the questionnaire we just removed the brackets around the check, and implemented a suitable error message. The error messages were different for the web and for the cai questionnaire so again we gain from the solution with the WEB- and CAI-languages. Some checks should only be conducted in the CAI-questionnaire. Those were grouped together, so we easily could put them in a pair of comment-brackets for the web-version.

The handling of DontKnow was a big issue. For the purpose of the experiment it was wanted that it should look similar in the different instruments. It could be that don’t know should be shown as an ordinary answer category both on web and on CAI. It could also mean that don’t know should not be a visible answer for none. That would mean that we set an empty-attribute on every question and let the meaning of empty be like don’t know. Then it was wanted to have a warning when the respondent let a question be unanswered. The consequence of this would be a huge amount of checks!

The conclusion was for a while to disable the DONTKNOW and put the Don’t know as an ordinary answer category for each question. But again we need the Don’t know for the questions that should be answered by a number. The final conclusion was near the standard solution. Use the Don’t know answer to be a button or a Function key in the CAI-instrument as the interviewers are familiar with. In the Web-questionnaire it is a separate answer category.
Some of the fields were not allowed to be answered by Don’t know according to the rules from CVTS. For those questions we set a signal and tried to persuade the respondents to give an answer.

4.2.4. Critical/Newpage

When the web questionnaire was almost finished and we had decided on the layout; the fonts and the space between the questions, and the length of the lines of text, we could start the work on separating the questionnaire in different pages. Here we had to take into consideration both the visual impression, and the most practical way of running the rules. In addition to the newpage, we marked some fields as critical. In our work here we had some clear guidelines. We do not want

any scrollbars. And we do not want new questions to show up on a page when the respondent clicks the next-button. Checks should be conducted straight away so they don't meet the situation of getting a "red" page of error messages when leaving a page they hoped they had finished. On the other hand, we could not put critical on each field. But we found an acceptable compromise here.

4.2.5. Help text /instructions

In our questionnaire we have no use for neither a help language nor a separate help file. The needed instructions were implemented in the question text.

4.3. Layout

We had several discussions with the Survey methodologists on matters of screen layout. We need to agree on the look and content of the different parts of the screen, the header, menu and footer. We skipped the progress bar. The background colour was first grey, so pastel green, and so grey again. Another topic was the action buttons; should they have symbols or text? In which order should they be aligned, where should they be placed? We had to type some changes in the style sheet biHTMLWebPage.xsl to meet the wishes of "the questionnaire group". The left pane was also a topic for some discussions. We did not need any menu, but we want to use it for giving the respondent an overview of the questionnaire. We should also offer a visible link to support. We tried to make a html-document, but the solution was to make an image and an image map for the e-post address.

4.3.1. Fonts

To clarify the visual understanding of the web questionnaire, we needed different presentations of text, one for each category:

- Questionnaire title (Arial 20 Bold (in the header))
- Section headings (Arial 12 bold)
- Transition text (Arial 9)
- Questions (Arial 8 bold)
- Instructions (Arial 8)
- Answer labels (Arial 8)

We agreed on font specifications as described in parentheses.

They were implemented in the modelib. To avoid misunderstanding for the interviewers we needed to change the modelib we use for them as well. We want them to have a layout similar to other surveys they do.

4.4. Other Blaise IS issues

In addition to the work on the questionnaire itself and the layout we had some other tasks.

We had to translate the texts in the Blaise IS environment into Norwegian. It is not easy to find similar phrases in another language. An issue here is also to be sure in which context the phrase shows up. Another of course, is that the respondent should understand the phrase. One example here was a Norwegian phrase for "Suppress" for the Signals.

We also got a problem with the Norwegian letters. In the questionnaire itself it was handled correctly. But in the heading, and if we use menu-buttons, the Norwegian letters were not represented correctly. To solve this problem we had to change the encoding="ISO-8859-1" into "encoding="ISO-8859-1" in the biHTMLWebPage.xsl

The list below is on topics we have to handle, and have made some preparations for, but not yet completed.

- To learn as much from the project as possible, we want to collect Process data when the respondent fills in the web-questionnaire. We set up a Journal file that gives us information about id number of the respondent, the session number and time when the respondent enters a critical question, or went to new page in the questionnaire. We also received the error messages the respondent activated, so we can figure out which questions are difficult for the respondents to answer.

- We gave the descriptions for the WEB server to the IT-department in Statistics Norway. They configured a WEB server with Internet Information Services. The server is located outside the internal net. So we have to use Terminal services to get access to the server and we have to send the blaise-databases via email to the internal net.

- Due to performance issues we use working database for saving uncompleted forms.

- The respondent shall open a certain record in the database for filling in the questionnaire. Therefore we need a login with user id and password. We send the user id and password to the respondent in a letter. This letter also contains information about the survey and of course it gives the URL to the survey.

- The last thing we had to implement before data collection started, was to build a system for supervision of the data incoming. It should give a good overview of data coming from the different modes of collection. The system should also offer good solutions on retrieving data for revision.

5. Conclusion

First of all, when writing this paper early in March it is too early to conclude. Due to a delay in the survey project, we have not finished our work yet.

But to conclude about the work so far. We are satisfied with the result. We have been able to produce instruments that meet nearly all requirements from our guidelines and expert group.

We have learned a lot in Blaise-, survey- and web- subjects. Hopefully we have gained from using just one software for the different instruments. We kept the questionnaire in one document nearly to the end. And that meant that we ensured we had the same structure and rule in the different instruments.

Our hope of just making a questionnaire, and then doing some slight changes to adjust it to web, was not fulfilled. It was hours of intensive work between those two instruments.

But of course a lot of what we have done to produce the Web-questionnaire is making standards that would be used in later Blaise IS projects. Another part of it is of course learning, testing and testing again.

At the end we will emphasize the importance of Tim Caratis advice and help. We would not have received a finale product like the one we have, without him

Figure 5.1. How it ended up

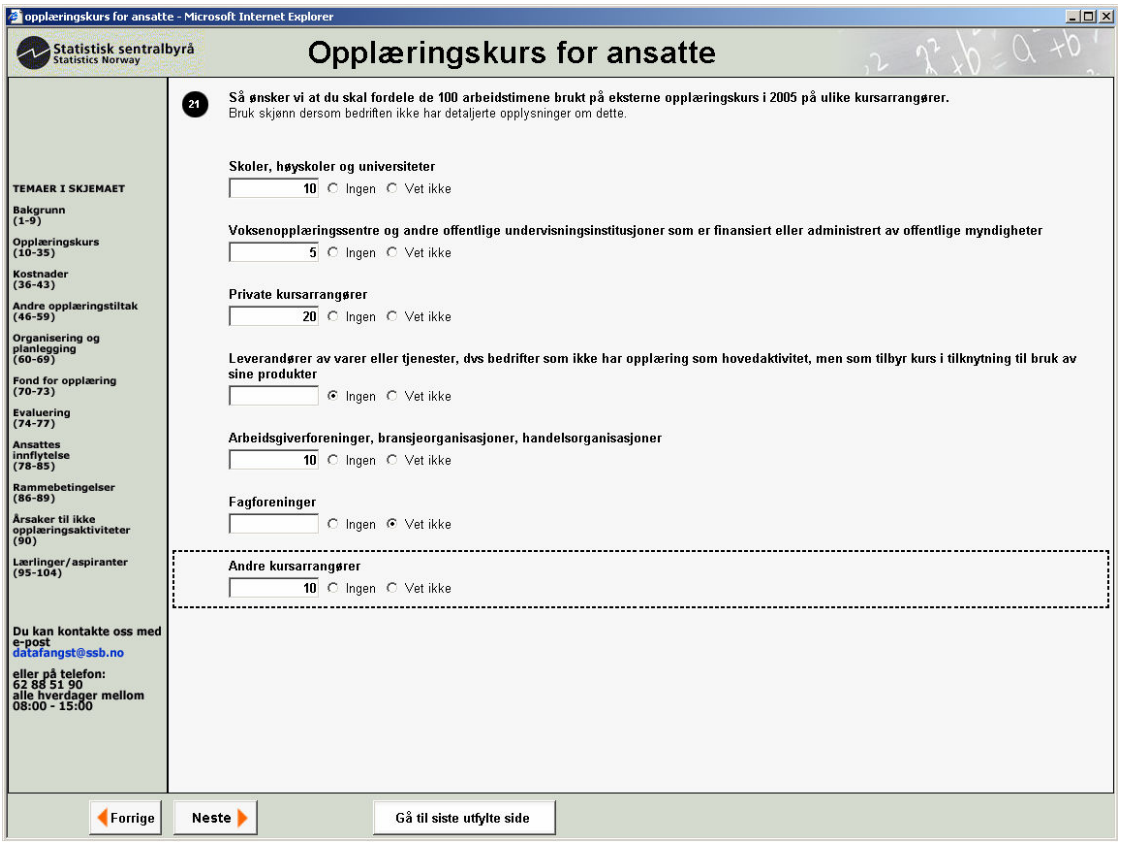
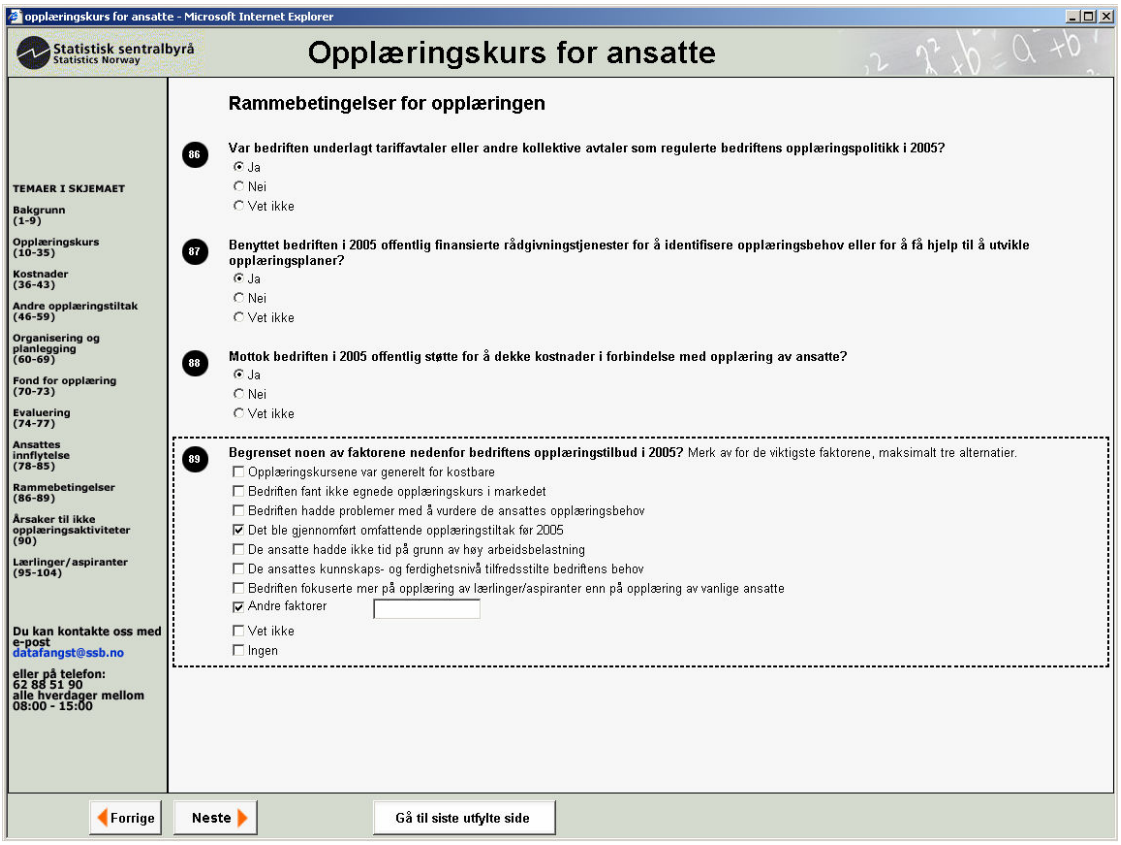


Figure 5.2. How it ended up



Distributed CAPI Interviewing with Blaise® IS for the Web for the Adolescent Medicine Trials Network for HIV/AIDS Interventions

Rick Mitchell (Westat, USA)

1. Abstract

The Adolescent Medicine Trials Network for HIV/AIDS Interventions (ATN) has introduced its newest behavioral Computer Assisted Personal Interview (CAPI) via the Internet to multiple clinical sites throughout the United States. The CAPI has been administered by trained interviewers to HIV negative adolescents as part of an HIV/AIDS intervention study. This paper will discuss the ATN's distributed CAPI interviewing experience with Blaise IS for the web, which in its early stages has already been well received by developers, interviewers, and our clients. The wealth of experience that has been gained from the ATN project will undoubtedly be extremely beneficial to future web-based CAPIs.

2. Introduction

For the past five years, the ATN project has developed five different Audio Computer Assisted Self Interviews (ACASI) that have been utilized across multiple protocols in a variety of settings. The ATN project has relied heavily on Blaise and Visual Blaise and has developed detailed standard approaches to coordinate getting instruments out into the field and subsequently, getting the data successfully back to Westat. While these projects have been a success, Blaise IS has offered the opportunity to shift the project to a Web CAPI operations model in which multiple users out in the field can access a single, central instrument through the web. This is possible because of ATN's site based structure in which interviewers are not administering surveys out in the community, but are actually conducting surveys through their existing subject populations within their own clinical settings. Since this particular ATN CAPI only planned to have five participating sites with two users at a time at each site, there would only be 10 total possible users at any given time. In reality, only a few users would probably ever be on at a time because of the clinic locations throughout the country with varying time differences. The limited stress on the system itself as well as the ability to handle technical questions seemed to make the ATN project the perfect candidate. Having received considerable notice ahead of time, a very reasonable development timeline was possible, which allowed ample time to work out any bugs or issues that arose with Westat's new experiences with Blaise IS.

3. Overview

This paper will describe the ATN project and how it shifted from a community laptop based CAPI operations model to a site based Web CAPI operations model.

- What is the ATN?
- The ATN Survey
- The Web CAPI, A New Site Based Operations Model
- Accessing the CAPI via the Web
- A Peek at the Web Survey
- Feedback Report

- Development Issues
- Getting Started
- Help Desk Support

4. What is the ATN?

The Adolescent Medicine Trials Network for HIV/AIDS Interventions (ATN) is a collaborative group whose goal is to strengthen the prevention and treatment of HIV/AIDS among adolescents ranging in age from 12 to 24 years old. The collaboration consists of multiple clinical sites throughout the United States, individuals with various areas of expertise, leadership teams, and a Data Operations Center (DOC), which represents Westat's role. The ATN has implemented approximately 40 different protocols covering therapeutic, community based, and behavioral studies. The ACASI projects for the ATN as well as the CAPI that is the focus of this paper have all been behavioral studies where participants are recruited at HIV clinics and then interviewed after they consent to participate. There are 15 key ATN sites throughout the country as well as additional subsites and collaborating sites, so essentially the DOC has worked with approximately 40 sites, five of which were chosen for the Blaise IS CAPI.

5. The ATN survey

The ATN survey was lengthy and consisted of approximately 516 behavioral type questions covering a participant's use of alcohol and drugs, sexual experiences, sexual practices, and medication compliance. The interview was conducted in a clinic setting that allowed for easy Internet access. The survey was administered at a baseline visit and then at 3 month follow-up visits where various rules were put in place to allow (and disallow) visits based on pre-defined time dependent windows. Baseline interviews took approximately 2 to 3 hours to complete and follow-up interviews took approximately 1 hour to complete. A participant's responses to various skip patterns could potentially decrease this number significantly. For example, if one never drank alcohol, then he or she would not be asked subsequent questions relating to alcohol. The ATN project was even more ideal for this initial exposure to Blaise IS. Since the interviewer was actually the one asking the questions, there was not as big of a concern at this point on dealing with the user friendliness of the system and navigational issues; although obviously, these were still somewhat important to keep in line with Westat's exceptional survey standards. At the completion of the interview, a "Feedback Report" was generated which the interviewer would immediately print out and go over with the participant. This real-time feedback gave interviewers the opportunity to suggest specialized centers where participants could seek additional help and support for serious problem areas that were identified during the interview.

6. The web CAPI, a new site based operations model

With ATN's site based structure, the project no longer needed to be under a "household" CAPI type model in which individual laptops were a necessary

component for interviewers administering surveys out in the community. The ATN project was able to benefit from a new operations model, the web CAPI, in which personal interviewing could be done at select site locations by a trained interviewer via the Internet. This new model represents multiple users out in the field being able to access a centralized CAPI from the luxury of their own PC within their office setting. As the project shifted to this new web based model, cost benefits were realized primarily in the areas of laptop logistics, data transmissions, and platform updates. These benefits are described below:

6.1. Laptop Logistics

Identifying optimal laptops for administering a survey, cloning the laptops, and then customizing and shipping each of the laptops to their designated destinations are all critical pieces of the laptop logistics that Westat has mastered across its extensive array of CAPI projects in past years. Each of these steps represents years of refining and tweaking to create efficient and cost effective processes for Westat's clients. Nevertheless, there are costs associated with these logistics. Because of the unique nature of the ATN project in which all surveys are conducted in a dedicated clinical setting with reliable web access, it was not necessary to utilize the household CAPI type of operations model and therefore Blaise IS could be utilized.

6.2. Data Transmission

Previous operations models required that interviewers electronically transmit data from a laptop back to Westat after each interview, but with the new Web CAPI operations model, data are stored real-time at Westat. A single dedicated server along with adequate backup servers was established at Westat to accommodate this new model.

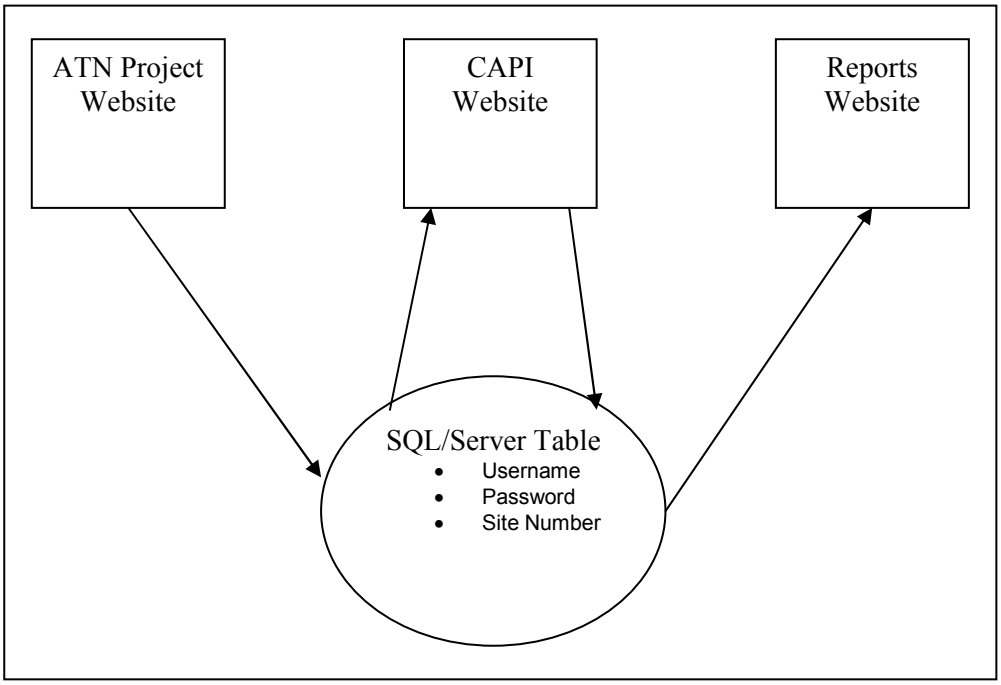
6.3. Platform Updates

As technology evolves and information gathering helps redefine research directions, projects sometimes require updates to either the CAPIs themselves or to corresponding software associated with the CAPI such as word processing and spreadsheet software. Because the CAPI is accessed through the web, it is only necessary to make these updates in a single location, the data center at Westat.

7. Accessing the CAPI via the web

The most obvious difference between CAPI and Web CAPI is that instead of having multiple instruments out in the field for multiple users, there is just one instrument used by multiple individuals. For a large-scale project that represented 100 different sites, one can imagine the decreased logistical complexity involved by having those 100 sites merely access a single centralized instrument for which the data center has complete and total control. The two primary requirements for users of a Blaise IS CAPI are that they must have (1) a PC, and (2) reliable Internet access. An "up to date" browser (preferably Microsoft's Internet Explorer (IE)) is also needed, but functionality may vary. There are actually three separate websites that are interconnected to make the CAPI happen from start to finish including (1) the ATN project website, (2) the CAPI website, and (3) the Reports website. Effort was taken to make the transition from one website to another as transparent as possible to the user and still have the same "look and feel" across websites. To the typical user, it should appear that he or she is within one website. Figure 7.1 on the following page shows how user information is passed across each website so that appropriate access can be authenticated without the user having to perform multiple logins. A user first must log into the ATN project website, and then user information is automatically passed to the CAPI website.

Figure 7.1 – Authenticating a user across websites

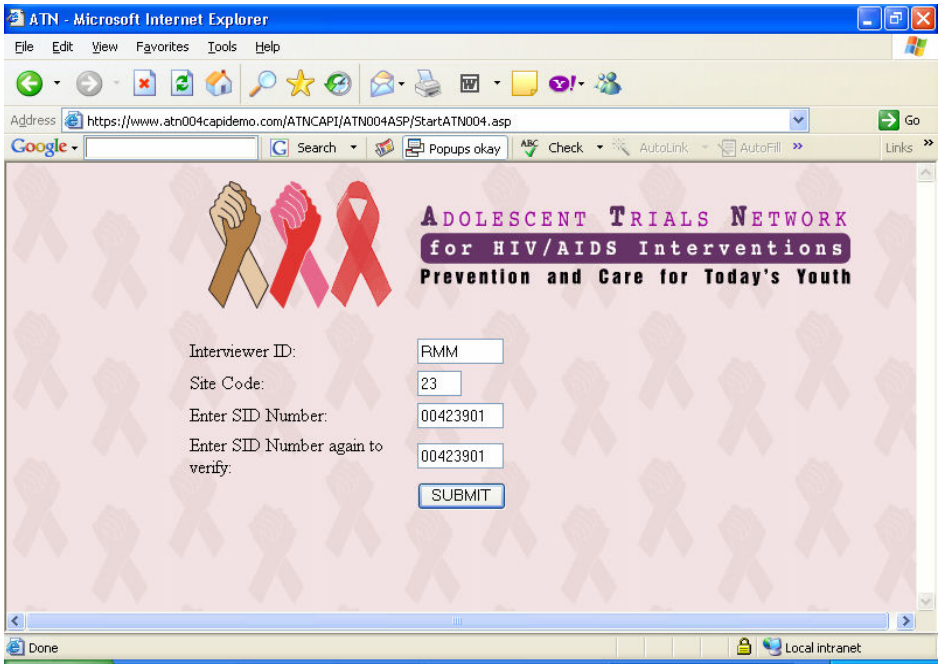


During the process, the ATN project website also extracts the user's site number from a centralized SQL/Server database and then passes the site number to the CAPI website to ensure that a site user's data is stored and accessed appropriately. Subsequently, the interviewer enters a Subject ID (SID) number from a pre-defined list of authorized numbers. These numbers are automatically linked to each site such that other sites using the system cannot access these numbers and their corresponding data. Finally, after completing the CAPI, the interviewer is then directed to the Reports website where again, user information is automatically passed. A site is only able to see data and corresponding reports for their own site's participants.

8. A peek at the survey

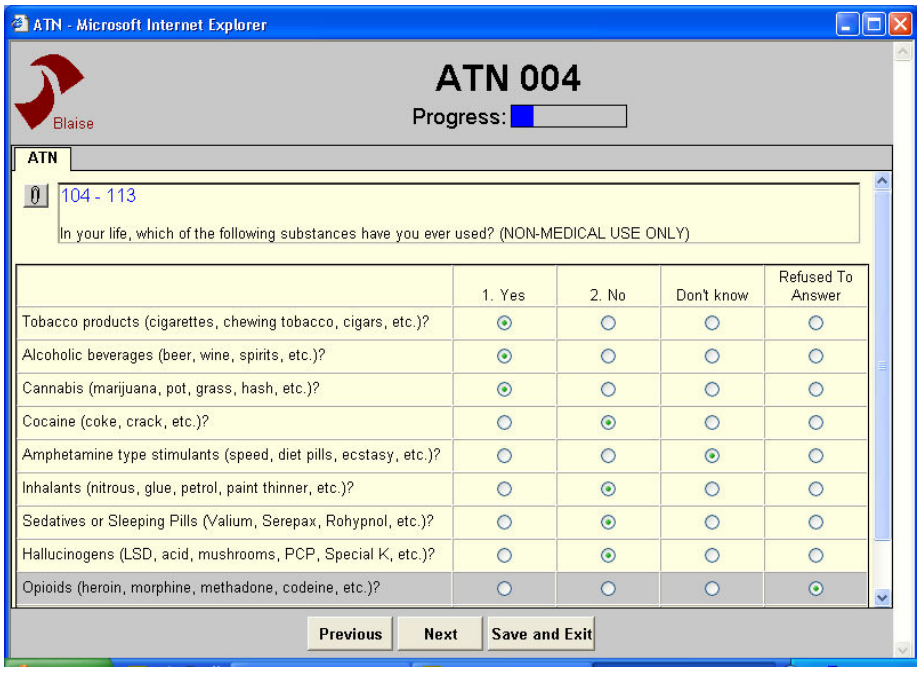
After successfully logging into the ATN website, one may access the main CAPI screen (Figure 8.1 on the following page) where information is collected for the interviewer ID, the site code, and SID number of the individual who will be taking the survey. The SID number is requested twice to decrease the number of typos that may result in access to an inappropriate survey number. Note that since the user logged into the ATN website first, the system already knows the user's corresponding site number, so entering a different site number will prevent access. It is not possible to reach this main screen of the CAPI without first logging into the ATN website, as necessary parameters must be passed between the two systems to initiate the CAPI.

Figure 8.1 – Accessing a subject's survey



In Figure 8.2 below, one can see a sample screen from the ATN 004 CAPI. This particular screen represents questions regarding a participant's use of tobacco, alcohol, and select drugs. Note that multiple questions with varying column responses can be utilized through the web with Blaise IS.

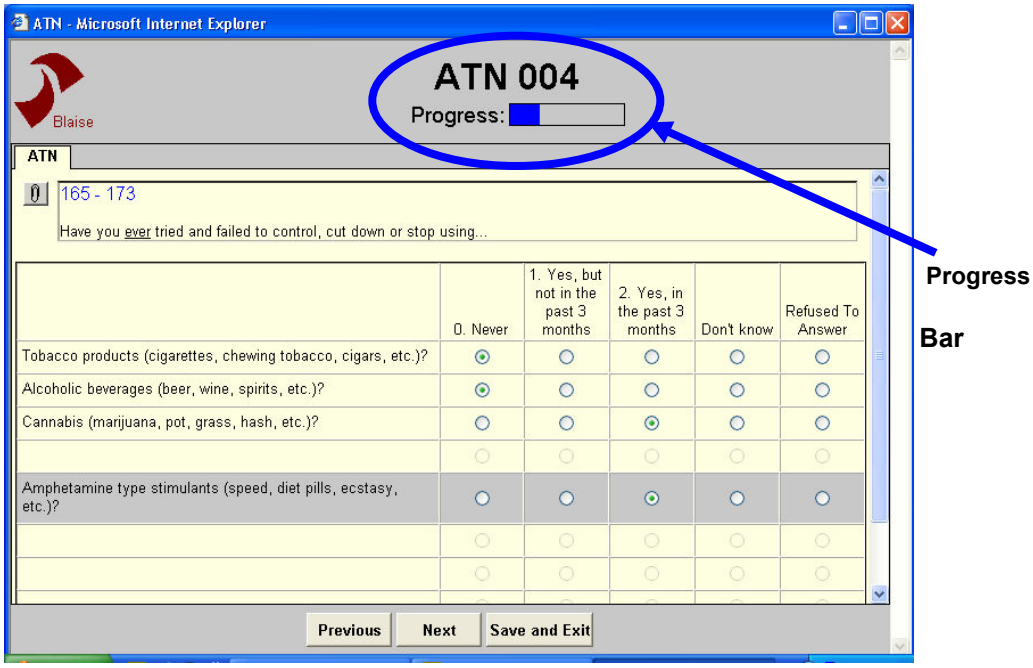
Figure 8.2 – Multiple questions/responses



One of the advantages of administering an automated survey is that skip patterns and edit checks can be built in not only to help manage the data, but to appropriately channel a participant through to only necessary sections of the survey. Note that in Figure 8.2 participants were asked whether they EVER used select drugs while Figure 8.3 on the following page asking a follow-up question on this same topic and excludes irrelevant drugs (i.e., drugs which the user already indicated as "never taken"). A very important component of Blaise IS is also shown in Figure 8.3 as one can now have a much better idea of one's progress

through the survey. The "Progress Bar" gives one a general idea of how far the user has gone through the survey and about how much more of the survey is remaining.

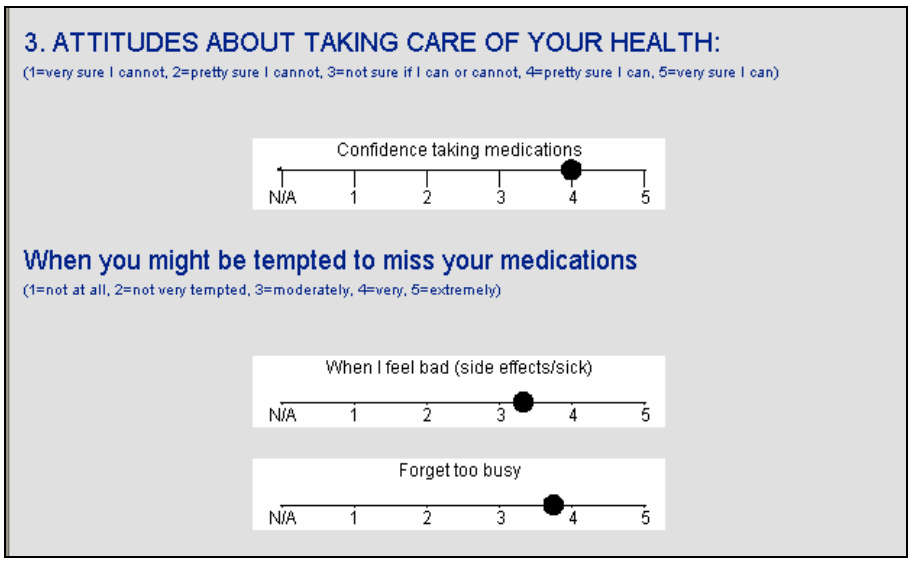
Figure 8.3 – Skip patterns and the progress bar



9. Feedback report

At the conclusion of the survey, it was necessary to generate a "Feedback Report" (Figure 9.1 below) that the interviewer and interviewee could go over together to assess any immediate problems that should be addressed.

Figure 9.1 – Sample section of feedback report



Pre-defined algorithms determined these "problems", which would indicate where further guidance or counseling for the participant may be needed. For example, if a subject indicated thoughts of suicide, then the coordinator would suggest possible locations in the participant's community where help could be sought out. This real-time reporting resulted in a 17-page HTML report that summarized and highlighted key topic areas. Because of its complexity in terms of data

presentation and the need for many scales to be calculated with hundreds of variables, SAS IntrNet software was used to program the report. This was achieved by first transferring the data from Blaise IS to a Reports website where the data could be immediately processed and presented to the interviewer and interviewee. As was done between the ATN project website and the CAPI website, user authentication information was again passed between systems as this information was relayed from the CAPI website to the Reports website. The following steps were needed before processing the data in SAS:

- Select data were extracted from Blaise IS
- An XML string was sent to a SQL/Server database accessible by the ATN project website, the CAPI website, and the Reports website
- Data were extracted from the XML string via SAS and then converted and processed for the report
- The final HTML report was then generated by SAS onto the Reports website

10. Development issues

There were several important issues that were either problematic or explored as the development team progressed through the project. A few of these noteworthy issues included performance (speed), stress, and multiple Blaise IS version releases.

10.1. Performance (speed)

The development approach for the survey was modular such that each section of the survey was initially developed and tested individually. It wasn't until the survey components were fully integrated that the development team discovered that the survey slowed down considerably the farther into it that one went. Further investigation showed that several sections in particular involved complex skip patterns and looping where information was gathered for multiple "partners" of each subject. If one had a complex sexual history, then detailed information was gathered on each partner. Similarly heavy drug use over long periods of time also lead to complex looping patterns within the program itself. Programming a more efficient looping process and installing an upgraded version of Blaise IS ultimately solved this problem, fortunately before it went into production.

10.2. Stress

No apparent stress problems were discovered in the system with Blaise IS during its early use. Because of the limited size of the ATN project audience, it is not yet known exactly how much stress on the system by multiple users will cause significant performance problems and/or breakage of the system. However, during testing, multiple users intentionally administered the survey at the same time to look for any potential stress problems, and none were evident.

10.3. Multiple Blaise IS releases

Since Blaise IS is a "cutting edge" software, it is still very much evolving. While it is advantageous to have the "latest and greatest" software, the ATN development project took place during a significant time period of this evolvement, and it was necessary to be a part of multiple installations of Blaise IS version releases as they came out. These were very beneficial to the developers in terms of improved functionality and resolved bug problems (a significant number were noted in the early stages), but the process added time and effort.

11. Getting started

The process of getting all of the participating CAPI sites onboard was an initial project concern. A process was developed to slowly phase in each of the sites as they received their IRB approval rather than opening all of the sites at once. This approach allowed time to appropriately train sites (and the data center) as well as deal with any last minute tweaks with the system. Key elements of this approach included:

- Conducting multiple webcast training sessions for off-site users
- Including help desk and study staff in training sessions
- Setting up a demo area for practice
- Requiring users to complete practice cases before distributing production login information
- Phasing in users by one site at a time

12. Help Desk support

One of Westat's strengths is its ability to utilize help desk staff across a wide-range of projects. The projects may be different, but the manner in which a survey is developed and administered is often not so different once it is out in the field. A help desk staff member can be dedicated to multiple projects and thus not create too much of a financial burden on any one given project, but still give adequate support to the project (e.g. a staff member may be dedicated to project X at a 10% effort, project Y at a 15% effort, etc.) by giving "pieces" of time throughout the day to each project. With the introduction of Blaise IS, there will still be a need for help desk support. While some problems from the past may be resolved, other problems may linger on and new problems will undoubtedly be identified as teams gain further experience with Blaise IS. In the shift from CAPI to Web CAPI, help desk support has changed somewhat in terms of content, but certainly not in terms of need. Westat prides itself with being able to effectively "hold a client's hand" and guide he or she through an entire problem whether real or perceived. Getting the user back on track quickly and efficiently is what Westat's help desk does best. Types of help desk support are described in Figure 12.1 below.

Figure 12.1 – Help Desk support

Support **NO** longer needed:
Laptop navigation and functionality
Power source
Battery replacements
Software updates and upgrades
Lost Data (e.g. stolen or damaged laptop)
Support **STILL** needed:
Survey navigation
Remembering username and/or password
Entering wrong ID number
Trying to access expired ID number
Support needs that are **NEW**:
Pop-up blocker
Server down
Firewalls at the site
Trying to work "offline"
Old browser version
Non-compatible browser

13. Conclusions

A wealth of knowledge has been gained in both the areas of Blaise IS development and technical support of Blaise IS CAPI applications via the web. Preliminary feedback of the ATN CAPI has been extremely positive from users in the field. With laptop coordination and support no longer needed for web-based applications, the complexity of many CAPI components has decreased significantly. The collection of data from a single, central location such as the web allows the data center to have full control over the data and the application. The interoperability of multiple websites allows projects to link a variety of website tools with the CAPI. While some new issues have been experienced that are web related, these are certainly not insurmountable and can be dealt with as they arise. The key message delivered in this paper is that Blaise IS has provided the ability to replace multiple instruments used by multiple users with only a single instrument used by multiple users.

14. Acknowledgements

Blaise® is a registered trademark of Statistics Netherlands.

The Adolescent Medicine Trials Network for HIV/AIDS Interventions (ATN) is funded by grant No. U01 HD HD40474 to the ATN Data and Operations Center from the National Institutes of Health through the National Institute of Child Health and Human Development (A. Rogers, L. Serchuck), with supplemental funding from the National Institutes on Drug Abuse (N. Borek), Mental Health (A. Forsyth, P. Brouwers), and Alcohol Abuse and Alcoholism (K. Bryant).

The ATN 004 study was scientifically reviewed by the ATN's Behavioral Leadership Group. Network scientific and logistical support was provided by the ATN Coordinating Center (C. Wilson, C. Partlow), at University of Alabama at Birmingham. Network operations and analytic support was provided by the ATN

Data and Operations Center at Westat, Inc. that is funded by grant No. U01 HD HD40474 (J. Korelitz, B. Driver).

DISCLAIMER: The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat or Westat's clients.

Blaise IS and Accessibility

Jim O'Reilly (Westat, USA)

1. Introduction

An important benefit of information technology is that persons with disabilities can participate in many activities and functions more fully than in the past. As the centrality of IT and particularly the Internet to communications has grown in the last two decades, the value and necessity of accessibility for the disabled has increased as well.

Many governments have policies mandating or promoting IT and Web accessibility. The W3C lists[3] government policies for the EU and sixteen countries[4] as well as those of Australian and U.S. states and Canadian provinces.

This presentation will focus on accessibility and Blaise in terms of the U.S. Section 508 standards for Accessibility for Electronic and Information Technology and People with Disabilities. While Section 508 is a U.S. regulation, its goals, standards and issues are quite general and applicable to Blaise users in other counties.

We will discuss the 508 requirements and process, review our experience with accessibility and the Blaise for Windows platform, and then focus on Blaise IS and Web accessibility.

2. Background

Section 508 is part of a Federal law requiring executive-branch agencies to provide comparable access to electronic and information technology (EIT) for its staff and the public with disabilities. The law creates an Access Board and a Federal Acquisition Regulation (FAR) to determine how to meet the requirement.

Most of the specifications for software concern usability for people with vision impairments (alternative keyboard navigation, animated displays, color and contrast settings, flash rate, and electronic forms, among others).

Section 508 Web-based technology and information are based on the Web Accessibility Initiative of the World Wide Web Consortium. These ensure access for people with vision impairments who rely on various assistive products to access computer-based information, such as screen readers, which translate what's on a computer screen into automated audible output, and refreshable Braille displays. Verbal tags or identification of graphics and format devices, like frames, are necessary so that these devices can "read" them for the user in a sensible way.

The standards do not prohibit the use of web site graphics or animation but require that such information is also available in an accessible format. The specification also addresses the usability of multimedia presentations, image maps, style sheets, scripting languages, applets and plug-ins, and electronic forms[5].

[3] www.w3.org/WAI/Policy
[4] Australia, Canada, Denmark, Finland, France, Germany, Hong Kong, India, Ireland, Italy, Japan, New Zealand, Portugal, Spain, United Kingdom, United States of America
[5] http://www.section508.gov/index.cfm?FuseAction=Content&ID=11

3. Blaise for Windows and Screen Readers

Blaise and Westat first focused on 508 and accessibility in terms of the Blaise for Windows platform. The approach was to rely on Microsoft Windows® support of accessibility and universal design and on the Blaise system's rigorous implementation of Windows standards. Westat also published a statement on Blaise and accessibility, www.westat.com/blaise/508info/blaise508stmt.htm

The Blaise team and Westat also committed to addressing accessibility issues that are reported by customers and users. In 2003 a customer reported that a blind application software tester could not make Blaise applications run in the Windows environment with the JAWS screen reader.

We confirmed the problem and investigated possible solutions. We quickly learned that custom-developed scripts for JAWS are common. The scripts specify for JAWS how specialized elements of an application's user interface work and how they should be treated by the screen reader. Once a script is developed and installed, JAWS will automatically use the script when processing specific types of files based on the file extension.[6]

Westat contracted with a consultant experienced in JAWS scripting for development of a JAWS script for the Blaise Data Entry Program (DEP) user interface. The consultant developed and delivered the script expeditiously, saying only a few elements of the DEP UI were problematic for JAWS. We tested DEP applications in JAWS and then passed the script along to the original blind software tester. She reported that it made DEP applications fully functional with JAWS.

Blaise for Windows applications were demonstrated running in JAWS with the script at the last IBUC conference in Ottawa, September 2004. The script is available from blaise@westat.com .

4. Blaise IS (Internet Services)

Web surveys present significant challenges and opportunities for accessibility. Key elements are:

- The Browser environment is diverse, changing and controlled by the user.
- Survey users are often the general public not just agency staff or employees. User experience, training, and support can't be assumed.
- The Web offers rich presentation features and user-interface elements that must be communicated properly by the screen reader.

The dynamic, diverse nature of web survey content also means that accessibility testing and evaluation must be pro-active and continuing rather than waiting for problems to be reported. Survey applications produced by the same system and structure may look and function quite differently because of the flexibility of the Web presentation. As a result, testing needs to take place at the application level, as well as the general system level.

[6] We also learned that a lively community of consultants experienced in JAWS scripting can be found through www.blindprogramming.com and other sites.

5. IS architecture – a higher level approach

The architecture of Blaise IS provide a major advantage in terms of the ability to implement accessibility features in Web surveys. IS is built on the technologies of ASP, XML and XSL. Little or no direct HTML coding is done in authoring an IS survey data model. The developer works in the Blaise Control Centre and builds the data model in same way as Windows CAPI or CATI—coding the familiar fields, rules, blocks, edits, etc.

Web presentation and look-and-feel is implemented with the settings in the mode library and LAYOUT statements in the data model. Other web survey characteristics are set outside of the data model in the specification file—the start and end ASP pages, the prepared data model, the XSL interview style sheet to generate HTML pages, and others elements.

Once the IS survey package is installed and active on the server, the interviewing process proceeds:

- The IS server receives requests and responses submitted by the web survey user.
- Data model rules are run.
- The next page's information is generated in a XML document.
- The document is passed to a process that applies the specified IS interview XSL style sheet and generates the HTML page.
- The HTML page is sent to the user.

So the XSL interview style sheet, as the generator of the HTML document, is the key to survey look-and-feel. It is the single location for implementing accessibility enhancements. Addressing web survey accessibility issues generally mean adding appropriate HTML tags or attributes to enable the screen reader to interpret the page properly. With Blaise IS this can largely be accomplished by adding commands to the XSL interview style sheet to generate these elements and their values. Once defined, the XSL interview style sheet can be used for future surveys with no further effort.

It's important to understand how large and sophisticated the primary IS interview style sheet is. There are some 9,000 lines of code, complex structures and many intricate relationships. Although it is a text file in a standard language that others could edit on their own, in practice only specialists on the Blaise development team should make changes. This is because they know how, because they will maintain the document, and because the evolving document will be distributed with each update of the system.

It is hard to overstate the importance for accessibility of the Blaise IS architecture and the interview style sheet. It provides an extensible capability well suited for flexibly improving and evolving the accessibility features of the system.

6. Section 508 Standards

Section 508 standards are comprehensive and general, as illustrated by Figure 1 for Web-based information and application.

Figure 1*: Section 508 Standard 1194.22 for Web-based intranet and Internet information and applications.*

(a) Text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content).

(b) Equivalent alternatives for any multimedia presentation shall be synchronized with the presentation.

(c) All web pages information conveyed with color also available without color, e.g. from context or markup.

(d) Documents organized so they are readable without requiring an associated style sheet.

(e) Redundant text links provided for each active region of a server-side image map.

(f) Client-side image maps provided instead of server-side image maps except where the regions cannot be defined with an available geometric shape.

(g) Row and column headers identified for data tables.

(h) Markup used to associate data cells and header cells for data tables that have two or more logical levels of row or column headers.

(i) Frames titled with text that facilitates frame identification and navigation.

(j) Pages designed to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.

(k) A text-only page, with equivalent information or functionality, shall be provided to make a web site comply with the provisions of this part, when compliance cannot be accomplished in any other way. The content of the text-only page shall be updated whenever the primary page changes.

(l) When pages utilize scripting languages to display content, or to create interface elements, the information provided by the script shall be identified with functional text that can be read by assistive technology.

(m) Web page requiring that an applet, plug-in or other application on the client system to interpret page content, must provide a link to a plug-in or applet that complies with §1194.21(a) through (l).

(n) Electronic forms designed to be completed on-line shall allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

(o) A method shall be provided that permits users to skip repetitive navigation links.

(p) When a timed response is required, the user shall be alerted and given sufficient time to indicate more time is required.

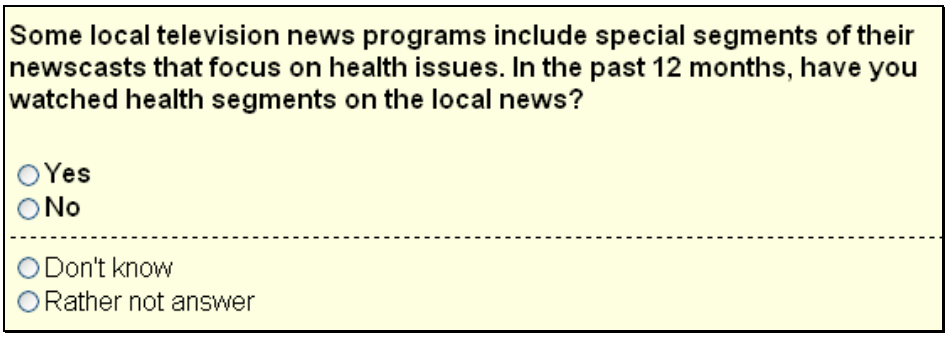
7. Accessibility testing of Blaise IS Surveys

In the first stage of our work on accessibility and Blaise IS, we focused on automated software to analyze web pages and report whether the Section 508 standards are being met. There are a number of free services on line and commercial systems that will check how well a web page meets the 508 web standards. W3C has a list of such tools[7].

[7] http://www.w3.org/WAI/ER/tools/

Because IS surveys may have hundreds of pages, we looked for an application able to test many pages in a batch. We selected AccVerify (www.hisoftware.com). Next, to capture the Blaise IS pages for a survey session we adapted the standard IS ASP page handler (biPagHan.asp), adding code to save every page to a file.

Our first test was of a sizeable web survey that had been completed. In the test session 182 web pages were saved and then processed through AccVerify. The testing reports identified that every page with an enumerated question "failed" because the radio button graphical display did not include an "alt=" string.



We discussed the issue with the Blaise team and agreed on the text to put in the alt= string. The XSL interview style sheet was revised to add this. We recorded the survey again and passed the recorded pages to AccVerify. No problems were reported.

We repeated the cycle a few more times—testing web surveys, processing pages with AccVerify, finding failures of different types, conferring with the Blaise team, receiving a revised interview XSL styles and retesting to confirm the change worked.

This process clearly illustrated the flexibility and power of the Blaise IS XML/XSL architecture in that significant enrichment of the system for accessibility could be implemented simply and quickly. At the same time we learned of the limitations of the automated testing of accessibility and the importance of testing with a screen reader.

8. Screen Readers

Screen readers are the essential assistive technology for blind users. Screen readers applied to a browser read the html document's text, interpret the tags and other elements, and convert the "readable" material into synthesized speech. From testing pages on the JAWS system and discussions with experienced users it's clear that

- Screen readers handle properly some things AccVerify and similar 508 standards checkers fail, e.g. radio buttons.
- Conversely, screen readers can be stumped by survey elements that meet the specific 508 standards.
- The true test for accessibility for blind users is whether the screen reader is able to make the page sensible for the user.

Another significant issue is that using a screen reader can be difficult for the inexperienced. Screen reader users rely heavily on key-combinations to navigate about the page. JAWS, the mostly widely used product in North America, has

about 66. So a sighted person with limited JAWS skills may struggle to determine if a problematic read of a page is the fault of the code or the JAWS user.

9. Screen Reader Testing of Blaise IS

We conducted two tests with JAWS of Blaise IS surveys included among the samples provided with the system. These were Expenses and NSSE2003. One tester was a staff member with normal vision and a few hours experience with JAWS. The second was an accessibility consultant who is blind and experienced with JAWS and other screen readers.

Figure 2. Tests of IS surveys with JAWS screen reader

Test 1: sighted Blaise developer

- Reads header but not footer (Prev, Next buttons)
- Items with preface text—not reading preface
- At times not reading enumeration sets past first empty selection
- Tables—column labels not read
- Items with icons to left (Help & Comment buttons) are not read properly

Test 2: (blind consultant)

- Overall 70-80% of full accessibility
- Need explicit titling of columns with <TH> tag
- Form elements need a <Label for="idname">
- Recommends alt-key combinations for key navigational elements, <Prev>, <Next> etc.

Of the items, only one was identified by both tests—labels for table columns. For the rest of the items mentioned by the sighted JAWS user, the blind user didn't report a problem. This effort clarified two issues: first, inexperienced JAWS users have limited value as testers and, second, the ad hoc approach of finding problems here and there and fixing them is insufficient. The number of issues that need to be considered is larger than initially believed.

10. A Broader Approach to Web Survey Accessibility

Interpretation and implementation of the 508 standards has been evolving as Federal agencies and others analyze the issues, develop interpretations, specify requirements and test products. The most detailed and well organized that we have found is from the U.S. Social Security Administration (SSA). As an agency with the elderly and disabled as major constituencies, SSA has taken the lead in developing processes and materials to communicate to government officials, software developers and vendors how to apply Section 508.

SSA's Accessible Solutions and Usability Branch has produced a significant set of materials related to accessibility. In the "Guide to Applying Section 508 Standards", the standards are expanded upon with SSA's interpretation of the standard and the specification of one or more requirements to meet the standard. For each requirement the assistive technology to be supported, recommended evaluation criteria and the testing method to judge whether the requirement is met are all provided. Figure 3 presents the SSA information on the first of fifteen elements of the Web-based intranet and Internet information and applications standard Sect 1194.22.a.

Figure 3: SSA Elaboration of Section 508 Section 1192.22.a Standard[8]

Standard 1194.22.a	**A text equivalent for every non-text element shall be provided (e.g. via "alt", "longdesc", or in element content).**
SSA Interpretation	AT users shall be able to access all meaningful non-text elements.
Requirement (a.1)	**Screen readers shall be able to speak all alt text or equivalent elements.** *Supported By*: JAWS *Recommended Evaluation Criteria*: All alt text must be meaningful and speak when accessed. *Testing Method*: Tab through controls to ensure that they speak and are meaningful. Arrow through images to see that they speak and are meaningful.
Requirement (a.2)	**Keyboard and voice input users must be able to access all text equivalent alternatives for non-text elements.** *Supported By*: JAWS, MAGic, Dragon *Recommended Evaluation Criteria*: All alt text or any equivalent must be exposed by the keyboard or voice input. *Testing Method*: Navigate by keyboard to see whether alt text or tool tips are exposed by using the keyboard or voice. ASB has developed a JavaScript capability to accomplish this.
Requirement (a.3)	**If textual links are not possible, then the screen name of the non-text element will be at the beginning of the alt-text tag, long description, title attribute, etc.** *Supported By*: JAWS, MAGic, Dragon *Recommended Evaluation Criteria*: All text links must have alt text beginning with the screen text shown. *Testing Method*: Use AT to expose alt text using screen readers, keyboard, or voice input.

The SSA Guide to Applying Section 508 Standards covers all six standards. Two of the six standards relate to survey applications including Blaise IS:

- 1194.21 Software applications and operating systems, 11 sections (a-l) and 56 requirements;
- 1194.22 Web-based intranet and Internet information and applications, 15 sections (a-p) and 54 requirements.

[8] Section 508 Accessibility for Electronic and Information Technology for People with Disabilities: Guide to Applying Section 508 Standards: U.S. Social Security Administration (undated).

Of the 110 total requirements for the two standards, the following appear relevant to Blaise IS and accessibility with JAWS.

Figure 4: Requirements most relevant to Blaise IS and use of screen readers.

Section	Requirements
21a	1, 2: Hotkeys must be assigned to commonly used control elements or notebook tabs to provide comparable mouse access. Tabbing which results in excessive keystrokes (more than 3) to activate an element is considered non-compliant. Hotkeys must be assigned to links used repeatedly on more than one screen or identical links used as templates for multiple applications. 3, 4: Tab indices must be assigned to significant text information, directional cues, and error text information, or a comparable access method must be attained. Must be a logical tab order, generally left to right, top to bottom. 6: Navigation to screen elements with the keyboard/voice must be available in a comparable manner to a mouse. 8: Hotkeys must not conflict with the browser. 10: Keyboard must be used to open list boxes, open and close tree structures, and navigate to the next logical item in proper order. 11: Navigation to radio buttons using arrow keys, items in a combo box, list views, or check boxes must not automatically select the item and/or change focus. 12: When focus changes visually to a new frame/pane or a logical place on the screen, a keystroke must be available to move focus if a well-defined visual focus does not occur. 13: When embedded links are within sentences or paragraphs, the link must stand on its own to be meaningful *and* must have a title attribute that is meaningful within the context of the sentence.
21c	1: AT users shall be able to use the keyboard or voice commands to move to logical points of focus in order to attain focus on selected items.)
21d	1, 2, 3: Navigation to controls must result in appropriate speaking of labels, data and cues; a well-defined visual focus; and track with screen magnifiers. 4: Interface elements must be exposed to voice recognition technology so users can access all controls.
21f	1: Screen readers should speak all significant text displayed
1l	1: Screen readers must speak all controls, labels, directions, and cues in a logical order. 2: Keyboard users must get focus to all controls, directions, and cues.

	3: Errors can be handled by the use of pop-ups, "red balls" or any other means. Errors can also be displayed after inputting a field or submitting a page. Access to errors must be keyboard accessible and comparable to a mouse user. 5: The information within the pop-up should speak immediately through a screen reader. 6: The pop-up should receive focus for screen magnification software. 9: A list of errors must be identified at the top of the page and receive focus. Links to each error must be displayed at the top of the page. 10: Each error on the page is associated with the element in error by some indicator that is identified by speech or text. 11: The user must have the ability to navigate precisely to each identified error on the page and know precisely where each error is without having to navigate through an entire form.
22a	1, 2, 3: Text equivalent for all non-text elements
22g	1: Row and column headers identified for data tables
22n	1: Screen readers must speak all controls, labels, directions & cues in a logical order. 2: Keyboard users must get focus to all controls labels directions cues. 3, 5, 6, 7: Errors can be handled by pop-ups "red balls" or other means. A list of errors shall be identified at the top of the page and receive focus. The user shall have the ability to navigate precisely to each identified error and know precisely where each error is without entire form navigation.
22o	1, 2: Provide links/methods to skip over redundant links.

These 37 requirements group roughly in five areas:

- Speaking all controls, labels, directions, non-text elements, and in a logical order
- Hot keys and tabbing
- Navigation
- Tables
- Error handling

Some of these elements are currently in the system. But others may be challenging to implement. Fortunately, the XML, XSL architecture of the Blaise IS system provides a strong platform for extending the system to meet these challenges.

As well, the information available from the Social Security Administration provides valuable direction and support for the effort. Not only are the issues and requirements specified, but evaluation criteria and testing methods are clearly defined.

With this and the skill and dedication of the Blaise team, there is reason to be optimistic that Blaise IS's current "70-80%" level of accessibility for the visually

impaired can be raised to the 9x% level. Both the Blaise team and Westat will be working on this effort in the coming period.

Session 2.4: Challenges at Statistics Netherlands

The Dutch Annual Business Inquiry: Developing and Testing an Electronic Form
Ger Snijkers, Evrim Onat, Jo Tonglet, Rachel Vis & Robert 't Hart
(Statistics Netherlands & Metaform, Amsterdam)

CASI questionnaires with Blaise 4.7
Gerrit de Bolster
(Statistics Netherlands)

Basil, A New Tool for CASI in Blaise
Roger Linssen & Jo Tonglet
(Statistics Netherlands)

The Dutch Annual Business Inquiry: Developing and Testing an Electronic Form

Ger Snijkers, Evrim Onat, Jo Tonglet, Rachel Vis (Statistics Netherlands) & Robert 't Hart (Metaform, Amsterdam)

1. Introduction

A major issue in Dutch governmental policy as to data reporting in general is reduction of response burden. As a consequence, Statistics Netherlands strives for reduction of data reporting for individual businesses, as well as making data reporting as efficient and easy as possible. One way to do that is providing electronic questionnaires via the internet (Haraldsen, 2004; Hedlin et al., 2005; Dowling, 2005).

In 2004, a project was started to redesign the paper questionnaire for the Dutch Annual Business Inquiry. First of all, the questionnaire was stripped to items necessary with regard to output demands. Secondly, the form was redesigned with regard to the structure (sections of items), instructions and wording. And thirdly, the visual design of the form was restyled. This redesign project created the opportunity to develop an electronic version of this complex questionnaire.

The electronic form was developed and tested in a number of steps. Functional issues of the form were investigated in a small usability test, using a draft version that was very similar to the original paper form (Snijkers, Tonglet & Onat, 2005). This test also resulted in the identification of navigational issues, edit rules and visual design issues that make an e-form different from a paper form. The result of this test was a prototype of the e-form. In a second step the prototype was tested again, in order to find out how the new form works in practice (Onat & Vis-Visschers, 2005). In this paper both usability tests will be described. The last step will be a field pilot in which the usability and data collection process as a whole will be tested.

2. The Dutch Annual Business Inquiry

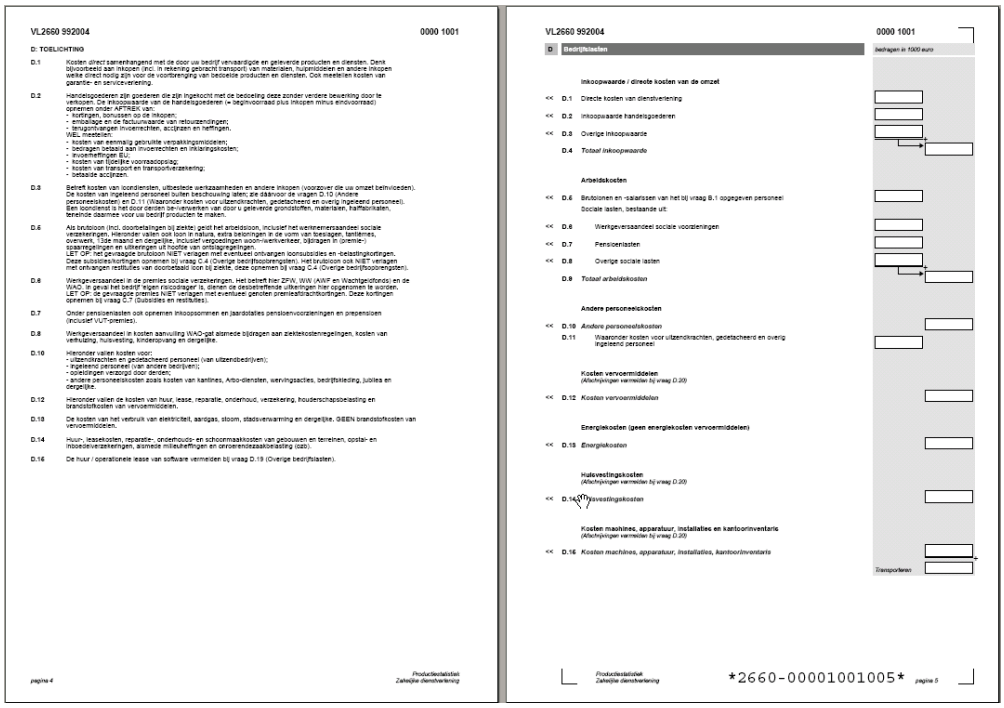
In the Dutch Annual Business Inquiry, businesses are requested to provide information on benefits and losses, i.e. their business accounts. In 2005 a sample of about 70.000 businesses was drawn for the 2004 Inquiry. These businesses received a paper form including an advance letter, saying that – among other things – this inquiry is mandatory. About 45% of the sample is self-selecting, meaning that they receive the questionnaire every year. This concerns the larger establishments with 50 and more employees.

The longest questionnaire for large establishments consists of up to 40 pages. A questionnaire may be seen as a booklet of A4 pages with the items on the right page and the instructions and explanations on the left page (see figure 1). The items are grouped into sections, which may be over more than 4 pages long. Sections concern issues like employees, benefits, costs, and business results. The questionnaire is characterised by many and voluminous instructions and explanations, because of differences in definitions as used by businesses.

The questionnaire is complicated and hard to complete. This has to do with the fact that a lot of detailed information is requested. Furthermore, the information has to be collected from various departments, and most of the definitions and the order of

the items on the questionnaire do not match those of the administrations. These aspects make the completion process of the form very cumbersome and time-consuming, resulting in measurement errors. Also the lay-out of the paper questionnaire caused measurement errors. These results stem from a detailed evaluation study of the paper questionnaire (Giesen, 2004, 2005).

Figure 1. The original questionnaire of the Annual Dutch Business Inquiry.



VL2660 992004 0000 1001

D: TOELICHTING

D.1 Kosten direct samenhangend met de door uw bedrijf verkaardigde en geleverde producten en diensten. Denk bijvoorbeeld aan inkopen (incl. in rekening gebracht transport) van materialen, hulpmiddelen en andere inkopen welke direct nodig zijn voor de voortbrenging van bedoelde producten en diensten. Ook meetellen kosten van garantie- en service-verening.

D.2 Handelsgoederen zijn goederen die zijn ingekocht met de bedoeling deze zonder verdere bewerking door te verkopen. De inkoopwaarde van de handelsgoederen (= beginvoorraad plus inkopen minus eindvoorraad) opnemen onder AFTREK van:
- kortingen, bonussen op de inkopen;
- emballage en de factuurwaarde van retourzendingen;
- terugontvangen invoerrechten, accijzen en heffingen.
WEL meetellen:
- kosten van eenmalig gebruikte verpakkingsmiddelen;
- betaalde en betaald invoerrechten en inkoopkosten;
- invoerheffingen BTW;
- kosten van tijdelijke voorraadopslag;
- kosten van transport en transportverzekering;
- betaalde accijzen.

D.3 Betreft kosten van loondiensten, uitbestede werkzaamheden en andere inkopen (voorzover die uw omzet beïnvloeden). De kosten van ingehuurd personeel buiten beschouwing laten; die dient/vindt de vragen D.10 (Andere personeelskosten) en D.11 (Waaronder kosten voor uitzendkrachten, gedetacheerd en overig ingeleend personeel). Een loondienst is het door derden (se-)verwerken van door u geleverde grondstoffen, materialen, halffabrikaten, tenminde daarmee voor uw bedrijf producten te maken.

D.6 Als brutoloon (incl. doorbetalingen bij ziekte) geldt het arbeidsloon, inclusief het werknemersaandeel sociale verzekeringen. Hieronder vallen ook loon in natura, extra betalingen in de vorm van toeslagen, tantièmes, overwerk, 13de maand en dergelijke, inclusief vergoedingen (oor-)werkverkeer, bijdragen in (premie-) spaarregelingen en uitkeringen uit hoofde van eigenrisicoregelingen.
LET OP: Hieronder mogen brutoloon NIET verlagen met ontvangen bonussubsidies en -loestellingkortingen. Deze subsidies/kortingen opnemen bij vraag C.4 (Overige bedrijfsopbrengsten). Het brutoloon ook NIET verlagen met ontvangen restituties van doorbetaald loon bij ziekte, deze opnemen bij vraag C.4 (Overige bedrijfsopbrengsten).

D.8 Werkgeversaandeel in de premies sociale verzekeringen. Het betreft hier ZFW, WW / AWF en Waz/ingeschonds en de WAO, in geval het bedrijf eigen risicodrager is, dienen de desbetreffende uitkeringen hier opgenomen te worden.
LET OP: de gevraagde premies NIET verlagen met eventueel genoten premieafdrachtkortingen. Deze kortingen opnemen bij vraag C.7 (Subsidies en restituties).

D.7 Onder pensioenlasten ook opnemen inkoopsommen en jaardotaties pensioenvoorzieningen en prepensioen (inclusief VUT-premies).

D.8 Werkgeversaandeel in kosten aanvulling WAO-gat alsmede bijdragen aan ziektekostenregelingen, kosten van verhuizing, huisvesting, kinderopvang en dergelijke.

D.10 Hieronder vallen kosten voor:
- uitzendkrachten en gedetacheerd personeel (van uitzendbedrijven);
- ingeleend personeel (van andere bedrijven);
- arbeidsgen verzorgd door derden;
- andere personeelskosten zoals kosten van kantines, Arbo-diensten, wervingsacties, bedrijfsfeesting, jubilea en dergelijke.

D.12 Hieronder vallen de kosten van huur, lease, reparatie, onderhoud, verzekering, houderschapsbelasting en brandstofkosten van vervoermiddelen.

D.13 De kosten van het verbruik van elektriciteit, aardgas, stoom, stadsverwarming en dergelijke, GEEN brandstofkosten van vervoermiddelen.

D.14 Huur-, leasekosten, reparatie-, onderhouds- en schoonmaakkosten van gebouwen en terreinen, costal- en inboedelverzekeringen, alsmede milieuheffingen en onroerendezaakbelasting (ozb).

D.16 De huur / operationele lease van software vermelden bij vraag D.19 (Overige bedrijfslasten).

pagina 4 Productiestatistiek Zakelijke dienstverlening

VL2660 992004 0000 1001

D Bedrijfslasten bedragen in 1000 euro

Inkoopwaarde / directe kosten van de omzet

<< D.1 Directe kosten van dienstverlening
<< D.2 Inkoopwaarde handelsgoederen
<< D.3 Overige inkoopwaarde
D.4 *Totaal inkoopwaarde*

Arbeidskosten

<< D.6 Bruto/onen en -salarissen van het bij vraag B.1 opgegeven personeel
Sociale lasten, bestaande uit:
<< D.6 Werkgeversaandeel sociale voorzieningen
<< D.7 Pensioenlasten
<< D.8 Overige sociale lasten
D.9 *Totaal arbeidskosten*

Andere personeelskosten

<< D.10 *Andere personeelskosten*
D.11 Waaronder kosten voor uitzendkrachten, gedetacheerd en overig ingeleend personeel

Kosten vervoermiddelen
(Afschrijvingen vermelden bij vraag D.20)

<< D.12 *Kosten vervoermiddelen*

Energiekosten (geen energiekosten vervoermiddelen)

<< D.13 *Energiekosten*

Huisvestingskosten
(Afschrijvingen vermelden bij vraag D.20)

<< D.14 *Huisvestingskosten*

Kosten machines, apparatuur, installaties en kantoorinventaris
(Afschrijvingen vermelden bij vraag D.20)

<< D.16 *Kosten machines, apparatuur, installaties, kantoorinventaris*

Transporteren

Productiestatistiek Zakelijke dienstverlening *2660-00001001005* pagina 5

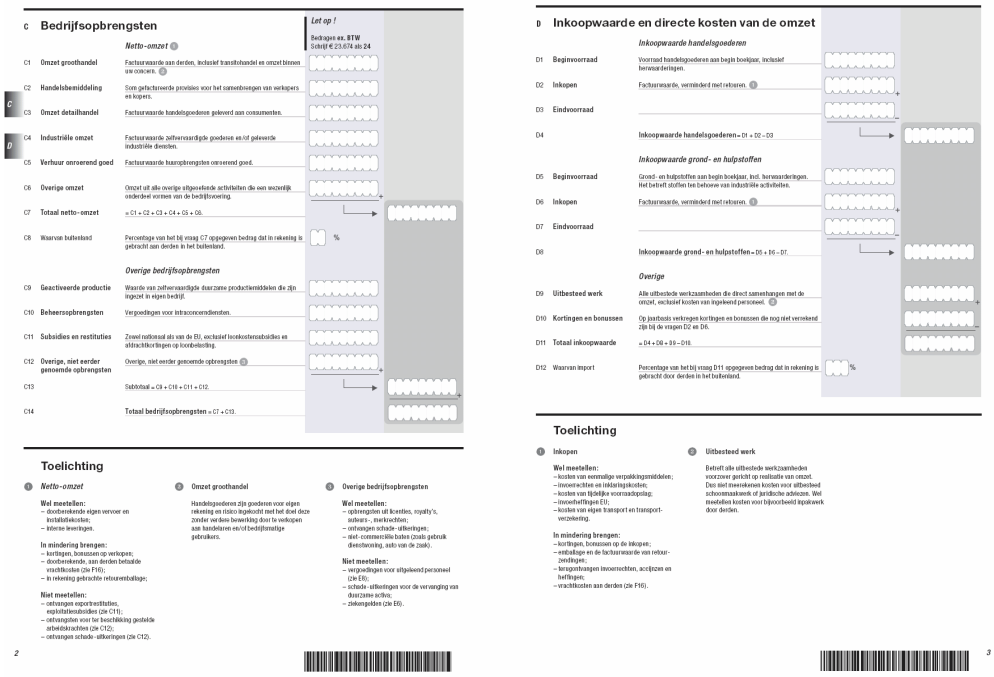
With these results in mind the structure and the lay-out of the questionnaire was redesigned. This came down to:

- breaking the questionnaire down into smaller sections of no more than one page, resulting in a better overview for each section, and less calculation errors;
- a strict order of [item label] - [short instruction] - [answer space], thus connecting – in reading order – items and answer spaces;
- locating additional instructions and explanations at the bottom of a page, like footnotes;
- restricting instructions and explanations to the most essential information, resulting in short and readable notes.

In figure 2 two pages of the newly designed paper form are presented. To get this result a professional designer[1] was consulted.

[1] Robert ‘t Hart (Metaform, Amsterdam).

Figure 2. The redesigned questionnaire of the Annual Dutch Business Inquiry.

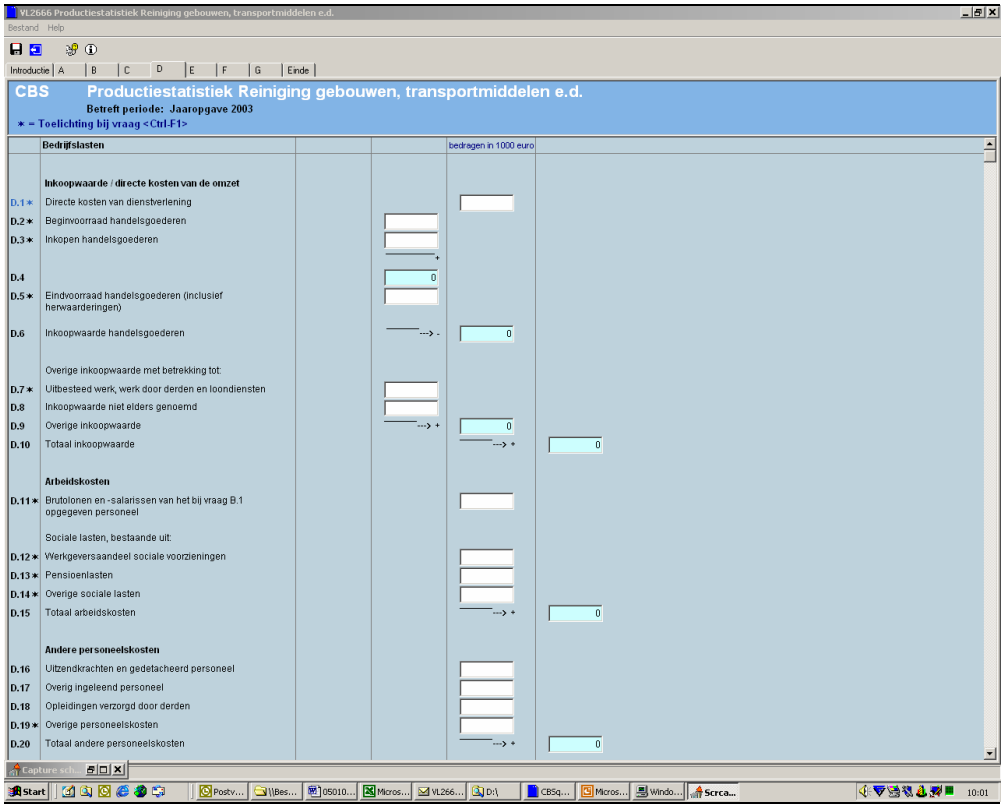


3. Testing the electronic form

The results of the evaluation of the paper form helped in thinking about the visual design of the electronic questionnaire. Because of this study, we already had a clear view on the response process with regard to this questionnaire (Giesen, 2004, 2005; see also Willimack et al., 2004). However, some research issues still had to be answered. We had to find out how the electronic form would work-out in practice, and what features had to be included in the e-form in order to make it easy to use. Another important issue was whether the paper and the electronic forms had to be designed in the same way. To research these issues a usability and test study was carried out.

At the time we started thinking about the electronic form, the results of the evaluation study of the paper form were not yet available. So, the old form (figure 1) was used as a basis. This form was programmed in Blaise. The use of Blaise set the layout conditions. This form is presented in figure 3. The original sections are the tabs in the e-form, located at the top of the screen. The tabs were labelled A, B, C, etc., corresponding to the sections of the paper form. Furthermore, the form is characterised by pages that need scrolling. To help the respondent complete the form, edit rules like calculations and checks were added to the e-form. To obtain explanations to items the key combination <Ctrl><F1> had to be pressed; this was indicated by '*' at the left hand side of an item.

Figure 3. The e-form of the Annual Dutch Business Inquiry, first version.



This form was tested in three waves, as is indicated in table 1. In the first wave an on-line version of this form was tested by 15 colleagues of Statistics Netherlands. These colleagues were familiar with the paper questionnaire, and comprised testers from the CBS cognitive lab, business interviewers, questionnaire developers, and helpdesk employees. The designer who redesigned the paper form was involved in this test wave.

In the second wave, this form was tested in the field by 6 CBS business interviewers and involved 37 businesses. These businesses were selected in such a way that a variety of branches of industry were included in the study: Software development, road transportation, accountant bureaus, and building cleaning. These businesses were middle to large businesses as to number of employees, i.e. 50 and more. In this way, the completion process would be complex and vary among businesses, in the sense that smaller and larger business administrations were involved and perhaps even more than one administration had to be consulted.

The questionnaire was loaded from a CD-rom onto the laptop of the interviewers and completed by the interviewers at the offices of the businesses. The interviewers were instructed in the use of the electronic questionnaire, since they had to know how to operate the e-form in the field.

In the third wave, in-depth interviews were carried out with 6 business respondents at their office. Like in the second wave, a variety of branches of industry was included: software development, road transportation, and building cleaning. Again, the sizes of the businesses varied, from 20 up to 100 employees.

These interviews were carried out by testers from the CBS cognitive lab in cooperation with business interviewers. The business interviewers are experts on this form, and can identify errors in the completion process. The interviews were video taped. In this wave the respondents had to download the questionnaire from a CBS server via the internet (at a https-address), log-in to the questionnaire with a

username and a password, complete it off-line, and send the data back via a secure internet connection to a CBS server. The respondents were not familiar with the questionnaire.

Table 1. An overview of the test waves

	Wave 1	Wave 2	Wave 3
Period of testing	August 2004	October 2004	November/December 2004
Questionnaire	Blaise	Blaise	Blaise
On/off-line	On-line via internet	Off-line, installed from CD-rom on laptop of business interviewer	Off-line, downloaded via internet
Number of completions/ interviews	15 CBS employees	37 businesses	6 businesses
Tested by:	CBS cognitive lab testers, business interviewers, helpdesk employees, questionnaire developers, designer	6 CBS business interviewers	2 cognitive lab testers, and 2 CBS business interviewers
Results	Major difficulties with on-line version	Experiences from interviewers	Most results from inexperienced respondents

4. Results of the first test study

In this section the results of the three test waves will be described. The results will be presented in the order of the response process, i.e. i) retrieving the questionnaire, ii) starting up the questionnaire, iii) introduction to the questionnaire, iv) filling-in the questionnaire, v) transmitting the data and vi) deleting the questionnaire.

4.1. Retrieving and installing the questionnaire

In the test, three ways of distributing the questionnaire have been used. The on-line version, in wave 1, was characterised by long sending and receiving sessions. Even for fast data connections, the time needed to receive a new page lasted much more than 5 seconds. This was due to the length of the questionnaire and the included edit rules. Distributing the questionnaire via CD-rom, as was used in wave 2, on a large scale is relatively expensive.

In testing wave 3 (with the downloadable version), it was observed that the problems regarding retrieving and installing the questionnaire were mainly getting the https-address right. In this wave respondents had to type in a long and unusual https-address from a letter. This gave rise to typing errors. Also the fact that a secured address was used (https) brought about errors in the address: people did not see the ‘s’. After several tries, respondents needed help from the interviewers to continue.

Once the respondent had logged-in on the CBS-server the questionnaire could be downloaded and installed. The installation procedures did not cause major

problems. One respondent could not download the questionnaire because of a firewall; the questionnaire was installed from a CD-rom.

4.2. Starting-up the questionnaire

Once the questionnaire had been installed on the computer, it could be opened. In wave 3, however, respondents first had to log in to the questionnaire by use of a username, a password and an additional security code. This procedure did not cause any trouble, except for the use of the security code, which was supposed to prevent unauthorised logging in to the questionnaire with a hacked username and password, especially for on-line questionnaires. This code was generated by the computer, presented on the screen, and had to be copied. The meaning of this code was not clear and resulted in many questions: "why is it there?" "Since it is on the screen, it makes no sense, or does it?" Respondents were reluctant to continue, until the meaning was explained.

4.3. Introduction to the questionnaire

After the respondent had logged in to the questionnaire, a first page with information regarding the questionnaire was presented. This page contained information on the sections of the questionnaires and their order, as well as information on how to navigate, get to instructions, and transmit the data.
This was done in wave 3. In the previous waves, the opening page was the first page with data boxes. In these waves, the 'respondents' did not know what to do, although they were familiar with the paper questionnaire.

4.4. Filling in the questionnaire

Completing a questionnaire like the Annual Business Inquiry is a very complex and long process, which puts a heavy burden on the respondent (Giesen, 2004, 2005). During the test interviews, respondents sat in front of their computer, surrounded by piles of administration papers, notepads and a calculator. The filling-in process involved their turning from the questionnaire to the administration papers, finding a pen to make notes, returning to the questionnaire on the screen, opening the explanation window to the item using their mouse, going in search of additional administration papers, typing in numbers in the calculator, etc, and finally entering the data in the questionnaire. Respondents were also disturbed by the phone ringing, and by colleagues coming in asking for information. Also, in practice, respondents may need several sessions to complete the questionnaire, and they need to contact colleagues from other departments to get the requested information. It is hardly surprising that, during such a process, respondents became easily confused and got lost when completing the questionnaire.

With regard to a survey like this one, we have to take into consideration that for businesses there are no benefits in participating. To them the completion of the questionnaire only brings about costs. As Willimack et al. put it (2002, p. 225): "Survey participation is considered a non-productive activity, resulting in a cost to the business that does not generate profit." This means that business respondents will not be motivated to complete the questionnaire.

Because of the complex completion process and the lack of motivation, the usability and the user friendliness of the e-form are of great importance (Hedlin, Dale, Haraldsen & Jones, 2005). Our assumption is that when respondents have difficulties with the usability, they become irritated and will not complete the questionnaire (resulting in (item-)nonresponse) or will choose a satisficing response strategy, i.e. try to complete the questionnaire as quickly as possible e.g. by estimating answers (very likely resulting in measurement errors; Krosnick, 1991). Jansen and Steehouder (see d'Haens and Steehouder, 2000) call this the

kick-and rush behaviour2, meaning that respondents go straight for the tasks; they read as little as possible, read scanning looking for keywords, and seek an efficient response strategy. This behaviour will even be stronger when they are not motivated to participate.

The focus of the test was on the usability and the user friendliness of the questionnaire. Aspects concerned with usability are the visual lay-out and navigation. Some ideas to improve usability were mentioned by the respondents during the interviews. These ideas concerned printing, searching, calculating, carrying-over, explanations, and progress indication. These issues will be discussed in this subsection.

4.4.1. Visual lay-out and navigating

The characteristics of the e-questionnaire, as presented in figure 3, are described in section 3. In the test interviews in wave 3, we found that the tabs were not identified as tabs, and as such did not help respondents to navigate through the questionnaire: they did not realize that they could skip from one section to the next by clicking the tabs.

Because respondents did not identify the tabs as the separate sections of the questionnaire, they had no overview of the questionnaire and got lost. After completing the items in the first tab, and pressing <enter> to the last item, they skipped automatically to the next tab. All of a sudden a screen with empty answer spaces was presented. This confused the respondents, who then wondered where the answers had gone. They had not noticed that a new section was presented. The interviewer had to instruct the respondent on the use of the tabs. One respondent remarked that a structure like the Windows Explorer would be more logical, with all sections listed in an index at the left site of the screen.

Furthermore, some tabs were comprised of long pages, corresponding to long sections on the paper questionnaire. This made scrolling necessary, resulting in a bad overview of the section.

An e-form should be clear and user friendly, like every questionnaire (Dillman, 2000; Fowler, 1995). The lay-out should be functional in the sense that it should help the respondent in finding his way through the questionnaire. Furthermore, respondents want to know what sections of the questionnaire are completed and what still has to done. The visual lay-out of the tested e-form did not meet these needs. During the test, this resulted in a lot of questions by the respondents, on how to continue and where to go next. Because of unexpected skips, some respondents got lost in the questionnaire.

4.4.2. Printing

Respondents requested the possibility to print the questionnaire. In the tested version, no printing option was available. We found that respondents wanted to make a paper copy for several reasons. First of all, while completing the questionnaire, they want to know where they are and what data they have already entered. Secondly, when other departments have to be consulted, separate sections of the questionnaire can be passed on in hard copy. After the questionnaire has been completed, respondents want to check the data on paper and obtain authorisation to send the data to Statistics Netherlands. And finally, they want a paper copy for their archives.

[2] In case of electronic questionnaire we can speak of click-and-rush behaviour.

4.4.3. Help window for calculations

In addition to a print function, respondents requested for a 'fill-in window'. This is a window that can be opened with an item, offering the possibility to fill in the items from the administration papers that match the item and the corresponding numbers. In this window, all numbers will be automatically counted up, and the result will be put in the corresponding response space.

After completing the questionnaire, respondents did not know anymore with what items on the questionnaire specific items from the administration papers were listed. This aid would be helpful, when at the end corrections had to be made to the entered numbers, and respondents had to go over the questionnaire again. Also, the windows had to be printed with very item used. Respondents felt that such an aid would be helpful to keep track of the completion process.

4.4.4. Entry search

While completing the questionnaire, some respondents ticked off the items in their administration papers that were already entered in the questionnaire. At the end of the questionnaire, they noticed that not all items had been checked off. However, they did not know where to put these items. Up to this point, the questionnaire was leading in the response process, meaning that they searched for the items in their administration papers that matched the definitions on the questionnaire (or at least, what they thought would match). From this stage on, however, the administration papers became leading. Now, they had to find the entries in the questionnaire that matched the items in the administration papers. At this point, respondents would find it useful when they could search for labels, instead of having to browse through the questionnaire and hoping to find the correct item. Here, respondents indicated that an entry-search facility would be helpful.

4.4.5. Automatic calculations

In the tested questionnaire, items were automatically added or subtracted. This is a major feature of computer-assisted data collection (Couper et al., 1998; De Leeuw, Hox & Snijkers, 1995). Respondents were positive about this feature; as a matter of fact they expect the computer to do the calculations (Murphy, 2005). In some cases, however, the results of the calculations were unexpected or it was unclear were the numbers came from. This was the case when the calculated numbers were not logical, or when the results were put at the end of the page and respondents had to scroll to find them.

4.4.6. Carrying-over (imputations)

In the paper form, many relations between items exist, for example, the summation at the end of a page has to be carried over to the next, or the summation of a section is used in another one. On paper, respondents have to be instructed in these matters (as is the case with addition and subtraction). In the e-form, these rules had been computerised. We believed that this would make the form easier to complete, since on paper many respondents had shown difficulties with these rules (Giesen, 2004, 2005). It was discovered, however, that respondents became confused when, for example, the computerised imputation rules were not logical to them, or when it was not clear were the imputed number came from. Furthermore, they could not edit the imputed numbers, since these answer spaces were locked.

4.4.7. Instructions and explanations

In the tested questionnaire, explanation windows could be opened by pressing <Ctrl><F1>. Explanations were indicated by '*' with items, but respondents did not notice this mark. Therefore, they were not aware that help was available. Even when they did notice the asterisk, they did not know what it meant. Only after they

were told that an explanation window could be opened and how it could be done, did they use it.[3]

4.4.8. Progress indicator

Since respondents cannot easily browse through an electronic questionnaire, as is the case with paper forms, respondents indicated that a progress indicator is desirable. This indicator gives feedback to the respondent as to what parts have been completed and what still has to be done.

4.5. Transmitting the data

To send back the data, respondents had to log in to the internet. Before doing so, the respondent had to confirm that all relevant items had been completed. After this had been done, a pop-up window appeared asking whether the data should be transmitted now.

In this process a number of problems appeared. First of all, after the filling-in of the questionnaire had been confirmed, respondents had to press <enter> to continue; the interviewer had to inform them of this. Secondly, in a number of cases, respondents didn't manage to log in to the CBS server because of technical problems. Also, respondents were confronted with messages from the computer saying that 'manipula.exe' was trying to connect to a remote server. This computer program was part of the e-form, but since respondents were unaware of this hidden part, they did know what to do.

After the data had been transmitted, a confirmation was received, thanking the respondents for the data. Respondents were positive about this message. However, after this message window was closed, the window saying that the data are ready to be sent appeared again on the computer screen. This was very confusing. Some respondents thought that the data had to be sent again, although they had received a confirmation.

4.6. Deleting the questionnaire

After the response process was completed, respondents might have wanted to delete the questionnaire from their computer. In this study, respondents did not indicate that they would like to do this.

5. Recommendations and discussion after the first tests

Based on the results of our research the following recommendations are proposed with regard to the electronic questionnaire of the Dutch Annual Business Inquiry.

i) Retrieving and installing the questionnaire

- As to this long and complex questionnaire, we recommend a downloadable questionnaire to be installed on the computer, and completed off-line. This recommendation is based on our experience in this study and the evaluation of the paper form (Giesen, 2004, 2005). A rule of thumb (as used by the Dutch Tax Office) is that questionnaires of over 25 items should be off-line versions. This questionnaire may be completed in several sessions, and by several people from several departments. A downloadable form makes it possible to stop and

[3] This does not necessarily mean that the explanations are useful to respondents. From the evaluation of the paper form (Giesen, 2004, 2005), we know that they only read explanations when they need help. When they read the explanations, they may not be very useful, since they do not provide the required help. Also, help may be missing where it is wanted.

start again at any given moment. Also, all information concerning the questionnaire and entered data is available, making it possible to browse through the questionnaire while keeping an overview. Furthermore, the time to be on-line is relatively short, in comparison to an on-line version.

A drawback of the off-line version is that businesses with firewalls may not be able to retrieve the questionnaire. We expect, however, that in practice this may not be a big problem. This is based on the experiences with the Dutch Tax Office. Since the beginning of 2005, businesses are compelled to use electronic tax forms that have to be downloaded via the internet.

- Downloading and installing should be clear and simple. This could be done by providing an internet site with a simple http-address (like e.g. www.mycbs.nl). When this site is a personal site, this site should be secured by a username and a password, for example. The download and install procedures should preferably be in accordance with known conventions as used by MS-Windows.

ii) Starting-up the questionnaire

- The questionnaire may start with a log-in procedure. Since the respondent may feel that the questionnaire contains confidential data, a log-in procedure may be needed. A respondent will then be asked whether he would like to protect the questionnaire with a username and a password so that unauthorised personnel cannot open it. This procedure may be optional.

iii) Introduction to the questionnaire

- The test study shows that respondents need a clear introduction to the questionnaire. After having logged-in, the questionnaire should open with this page, listing information on the structure of the questionnaire, how to proceed, navigate, obtain explanations, fill in and transmit the data. This page should, preferably, not exceed one screen.

iv) Filling in the questionnaire

- The visual design should be functional in the sense that it should help the respondent find his way through the questionnaire, and provide information on what has been completed already and what remains to be done. Furthermore, as the evaluation of the paper form shows (Giesen, 2004, 2005), the questionnaire should be composed in a consistent way; every lay-out element that is not consistent may confuse respondents.

  The composition of the questionnaire should be instantly clear and simple. This should also be true for navigation. The tabs and the long pages (making scrolling necessary) did not provide an overview of the questionnaire. A set-up that people are used to is a design with the sections listed at the left side of the screen, as is shown in figure 4. Punselie (2004) recommends that such a structure should not be too deep. According to him, a structure with more than 4 levels results in people loosing overview. Also, each level should consist of no more than 7 sections.

  In the literature on Web questionnaires, a lot of attention is given to the visual design and navigating (see e.g. Best & Krueger, 2004; Punselie, 2003; Schonlau et al., 2002; Vroom, 2002; Van der Geest, 2001; Dillman, 2000; Dillman et al., 2005).This indicates that these issues are important regarding usability.

- Scrolling should be avoided as much as possible. Each section should be made to fit on a computer screen, i.e. paging instead of scrolling.

- The questionnaire should be composed out of small, clear sections. Here, the redesign of the paper form (as discussed in section 2) helped in designing the electronic form.

- The questionnaire should have a printing function. This may be a function which asks what section of the questionnaire should be printed, e.g. this section (empty), this section (including answers), the whole form (including answers), or an empty form.

- A fill-in window and an entry-search facility would help in obtaining a better match between administration papers and the questionnaire items, in stead of matching on face value and what comes first. This facility would help in making completion easier as well as reducing measurement errors (i.e. listing administration items at the wrong entry).

- Edit rules with regard to calculations and carrying-over (imputations) should be implemented in the form. Respondents expect the computer to be more than a passive receiver of data: "They expect the computer to perform calculations for them and to help them keep their data consistent by running edit checks for valid data.", as Murphy points out (2005, p. 10). However, these rules should be clear and logical to the respondents.

  Although not tested in this study, we state that the same is true for consistency and range checks. Dowling (2005) reports that respondents like edit checks because they want to get it right. Also, they feel that it is efficient in the sense that it "stops me getting a phone call later on", when the Statistical Institute calls back for data cleaning.

  However, experience with computer-assisted data collection (Haraldsen, 2004; Couper et al., 1998; De Leeuw, Hox & Snijkers, 1995) shows that edit checks should be implemented with care and tested carefully. Too many interruptions and error messages may frustrate the response process, and irritate respondents. When occurring, clear error messages should indicate the error to the respondent. Murhphy (2005) found in usability studies of electronic questionnaires that error messages are "showstoppers". Even if respondents were free to go on, "they would not continue to the next question unless they could resolve the edit message."

- Instructions and explanations should be presented in a clear way. It should be clear to respondents immediately that explanations to items are present, and that they can be viewed simply by clicking a button. This button, as well as short instructions, should be presented in such a way that they will attract attention. Since respondents are not likely to read long texts, especially not on the screen, and since they read scanning, the explanations themselves should be clear and short, as are the instructions to the new paper form (see section 2). This is also true because in web data collection no interviewers are present to provide assistance.

- While completing an electronic questionnaire, respondents need feedback on their progress. A clear progress indicator should be implemented.

v) Transmitting the data

- Before transmitting the data, a confirmation of the completeness and accuracy of the data should be requested. This may not only be interpreted as a check to whether the questionnaire is completed, but also whether the data are correct.

- Technical problems with regard to the transmission process should be avoided: this process should be tested carefully.

- The transmission of the data should be confirmed with a "thank you" message, indicating that the data have been received. After this message has appeared on the respondent's screen, the send-button should not be presented anymore.

vi) Deleting the questionnaire.

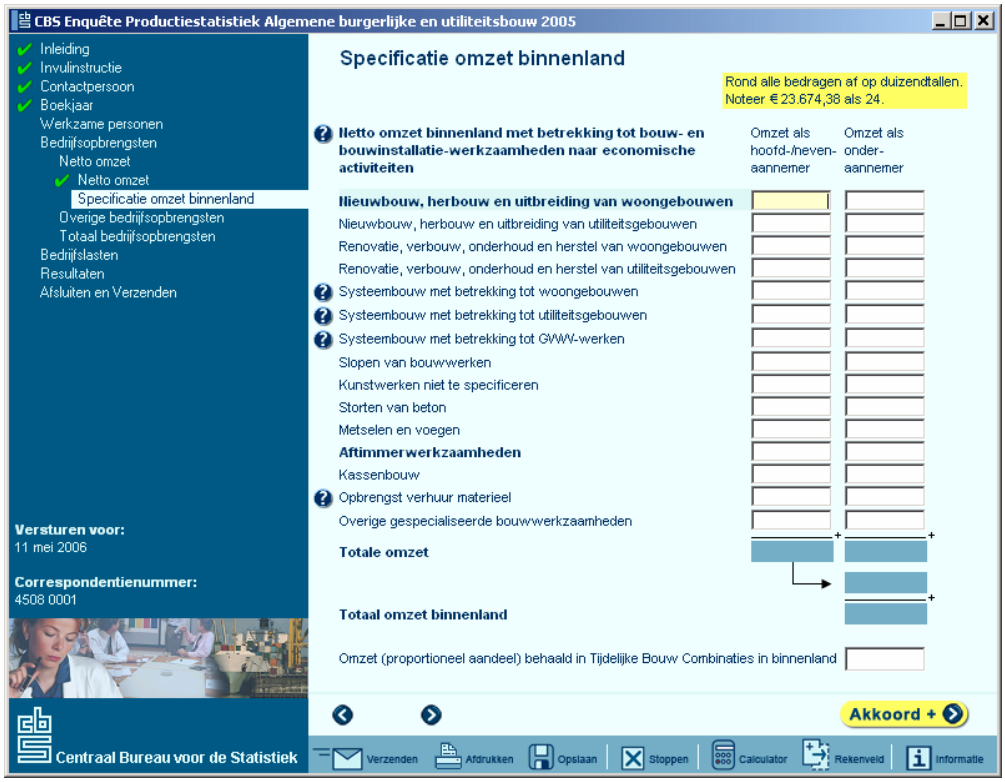
- Although this study did not provide any data on the need to delete the questionnaire afterwards, we feel that this option should be implemented in the system. In most software programs this is a default option.

With these recommendations in mind and following guidelines as presented in the literature on internet surveys (see e.g. Dillman et al., 2005; Best & Krueger, 2004; Haraldsen, 2004; Punselie, 2003; Schonlau et al., 2002; Vroom, 2002; Van der Geest, 2001; Dillman, 2000), this questionnaire was redesigned. The result is presented in figure 4.

6. Testing the new version of the downloadable electronic form

Following the above mentioned recommendations, a new electronic form for the Annual Business Inquiry had to be developed. This was not immediately possible, because the recommended visual design was rather difficult to program. The informatics department of SN developed a new computer program, based on Blaise that could build such an advanced visual design. The working title of this program was TEPS (tool for the electronic business inquiry), later it was named Basil. For this test round four different questionnaires were programmed in Basil. Figure x gives an example of one screen of the questionnaire.

Figure 4. Example of a screen of the questionnaire



The visual design of the form is similar to the electronic forms of the Dutch tax authorities, and as such respondents are familiar with the visual design. In the

frame on the left the different sections of the questionnaire are presented in an index. This index can be used to browse through the questionnaire. At the bottom of the screen there is a menu bar, with buttons for sending, saving, quitting and printing the questionnaire; a button for the calculator, and the help window for calculations and a help-button. Above the menu bar the navigation buttons are located. Additionally there is a yellow affirmation button, with which each screen must be completed. Finally there is the content frame in which the questions are presented.

The questionnaires were tested with ten businesses. The test interviews were conducted as follows: an SN consultant and an interviewer from the Questionnaire Laboratory carried out the interviews. The aim of the interview (i.e. testing the questionnaire) was explained to the respondent. Next the respondent received a letter with the internet address of the questionnaire in it. The consultant and the interviewer then observed how the respondent filled in the questionnaire. Afterwards they evaluated the task and the respondent could give his opinion and recommendations. The results of the test are presented next.

6.1. Results of the test

6.1.1. Downloading and installing the questionnaire

To download the questionnaire the respondent had to copy a lengthy internet address from the letter into the internet browser. This caused several problems, though the mistake of forgetting the "S" (indicating the secured connection) in "httpS", was not made due to the warning in the letter. Still it was a lengthy address with a long string of numbers, e.g. "https://.../53508001.exe". Several respondents checked and rechecked whether they had copied the right number. It made them uncertain.
Installing the questionnaire caused no real problems. Though for one respondent it was not possible to download an '.exe-file' on his computer. He had to ask a system manager to download and install the questionnaire on his desktop for him.

6.1.2. Navigation

As was said, there were two ways to navigate through the questionnaire, the index in de left frame and the navigation buttons. The respondents used both ways, though the buttons were used to browse chronologically and the index was used to skip quickly to a specific screen. The ways to navigate through the questionnaire was clear and needed no explanation. The only thing that might be mentioned was that respondents initially did not understand the function of the affirmation button, and used it to browse through the screens and erroneously affirmed incomplete screens. Still, they found out quickly how to use the buttons correctly.

6.1.3. Progress indication and the affirmation button

In the electronic forms of the Dutch tax authorities there is an affirmation button to validate the answers are given on a specific screen. By affirming the answers the screen is completed and "closed". In the questionnaire of the Annual business Inquiry all screen have to be "closed" before the questionnaire can be returned to the office. Yet, it cannot be determined in advance which questions have to be answered and which will be left open. It is possible that a business does not have certain costs even though they are in the questionnaire. Such questions can be left unanswered. By affirming that a screen has been filled in completely the respondent indicates that he/she consciously and correctly has left certain questions unanswered.

When a screen has been "closed" a green check mark appears before the section in the index. These check marks also functions as a progress indicator, then. A

respondent can see which sections he/she has completed and which sections still have to be filled in.

6.1.4. Explanatory notes

For many questions there are additional explanations available. In these explanatory notes for example precise definitions are given, or instructions on how to calculate a certain item. Again the way to access the explanations is similar to the electronic forms of the tax authorities: a button with a question mark before the question: . During the test we saw that the respondents did not know how to access the explanations. They did not "see" the button. Later the colouring of the button was changed and this had a positive effect: . Now, the respondents noticed the buttons and accessed the explanations more often.

6.1.5. Help window for calculations

The help window for calculations is a completely new feature in the electronic form. It was developed as a result of the remarks of the respondents in the previous test. In this questionnaire it is possible that for one item in the questionnaire several items from the respondent's administration have to be added up. To keep a record of which items from the administration have been used in the questionnaire it is possible to save the calculation in a special help window.

During the test no respondent used this function spontaneously. It was not immediately clear how it would work. Only after hints from the interviewers respondents noticed the button and tried to use it. Remarkably, once the respondents were aware that the function existed they were enthusiastic about it; they just had to be pointed towards it.

6.1.6. Returning the questionnaire

The questionnaire can be returned to SN in two ways, via the internet or via an e-mail. When a respondent clicks on the send button in the menu bar, several checks are carried out. Firstly it is checked whether all screens have been "closed" with the affirmation button. Next, it is checked whether a couple of essential items have been answered (i.e. some profit and some cost items have to be filled). Then the respondent is presented with the choice whether he/she wants to send the questionnaire via the internet or in an e-mail.

There were no changes with the previous version of the questionnaire on this point. So the observed problems there returned here.

6.2. Discussion

After this second test we can say that the user friendliness of the electronic questionnaire for the Annual Business Inquiry has been greatly improved. We noticed that the respondents enjoyed working with the questionnaire, even though the task had not changed compared to the paper questionnaire. Yet the "fun factor" of the task was higher.

The experiences in this test show that the respondents can handle the questionnaire and like to work with it. They still loathe the complicated task of filling in the questionnaire, though.

7. Conclusions

Completing the Dutch Annual Business Inquiry questionnaire is a very laborious and complex process (Giesen, 2004, 2005). Business respondents are poorly motivated to complete such questionnaires: they see no benefits, only costs (Willimack, 2002). This results in kick-and-rush (or click-and-rush) behaviour (d'Haens & Steehouder, 2000) and satisficing (Krosnick, 1991): respondents rush through the questionnaire, misread it, and provide the answers that are easiest for them (satisficing). When the questionnaire is badly designed, this behaviour will even be stronger, resulting in bad data. Ultimately, respondents will stop responding.

The following example illustrates this kick-and-rush behaviour. In the instructions to the questionnaire, respondents read that a help window for calculations was available. Yet, they did not use it spontaneously. Only after being hinted by the interviewer, they would remark: "Oh yes, you're right, I have read about that help window."

In order to tailor the questionnaire to the click-and-rush behaviour (and preventing non-response and measurement errors from occurring), the first version of the electronic questionnaire needed a lot of improvement. As a result of the test, a new version of the electronic questionnaire was developed in which most recommendations have been integrated.

In general, these recommendations involve making the electronic questionnaire clear and logical in every way. This means that the questionnaire should be simple, intuitive and consistent with regard to the visual design and its features. Also the structure of the questionnaire should be logical to the respondent, and should help to maintain overview. Breaking down the questionnaire into small parts and small tasks may assist in its step-by-step completion. Also a number of features should be built-in to assist the respondent in completing the questionnaire, like calculations, imputations, edit checks, a printing facility, a fill-in window, an entry-search facility, and a progress indicator. These built-in features should however be transparent: hidden rules and features may confuse the respondent and make them feel uncertain, even when they are familiar with the paper form. Instructions and explanations should be immediately clear.

To paraphrase Van der Geest (2001): Web questionnaire design is communication design. At all times the questionnaire should be intuitive and provide instant answers to questions like (see also Dillman et al., 2005):

- What I am supposed to do (next)? How should I continue?
- What will happen when I press this button?
- How is the questionnaire built up?
- Where am I? What did I do so far?
- Where can I put these data?
- Where can I get help?

These test show that completing a questionnaire on the computer and reading from the PC screen is very different from completing a paper questionnaire. These differences include the facts that a web questionnaire is not just a passive measuring instrument, and respondents expect it to help them. Also reading from the PC screen is different than from paper, as is navigating and obtaining an overview. With a computer people are less patient than when reading from paper. This is also concluded by Haraldsen (2004) and Dillman (2000). In order to make the questionnaire work well, the visual design (including character fonts) and its features have to be adapted to the chosen medium. As a consequence, the electronic questionnaire should be designed differently than the paper form.

However, we feel that to respondents both paper and electronic form should have the same look-and-feel.

Businesses ask for web questionnaires. In 30 interviews with businesses on web data collection, Dowling (2005) found that the internet is considered an efficient and respondent friendly way of data reporting. But, to be an efficient alternative for data reporting, electronic questionnaires should be easy to use, i.e. respondent friendly (Dowling, 2005, Hedlin et al., 2005; Snijkers, 2002), and, the visual design and its features should support the response process. Dowling concludes that when the web doesn't work, respondents will return to paper. Here, the web includes the whole process, from logging in to the internet, downloading the questionnaire, completing the questionnaire, to transmitting the data. Once respondents have returned to paper, it is our firm belief that it will be very hard to convince them to use the web again. That is the reason why we put a lot of effort in the development of this e-form. The next step is putting the questionnaire to the test in the field.

8. References

Best, S., and B. Krueger (2004). Internet Data Collection. Sage (Quantitative Applications in the Social Sciences Series, Nr. 141), Thousand Oaks, Ca.

Couper, M.P., R.P. Baker, J. Bethlehem, C.Z.F. Clark, J. Martin, W.L. Nicholls II, and J.M. O'Reilly (eds.) (1998). Computer Assisted Survey Information Collection. Wiley, New York.

De Leeuw, E., J. Hox, and G. Snijkers (1995). The Effect of Computer-Assisted Interviewing on Data Quality. Journal of the Market Research Society, Vol. 37, No. 4, pp. 325-344.

Dillman, D. (2000). Mail and Internet Surveys. The tailored Design Method (second edition). Wiley, New York.

Dillman, D., A. Gersteva, and T. Mahon-Haft (2005). Achieving Usability in Establishment Surveys through the Application of Visual Design Principles. Journal of Official Statistics, Vol. 21, No. 2, pp. 183-214.

Dowling, Z. (2005). The Challenges of Implementing Web Data Collection for Mandatory Business Surveys: The Respondent's Perspective. Paper presented at the First EASR Coneference (European Association for Survey Research), 18-22 July 2005, Barcelona, Spain.

Fowler, F.J. (1995). Improving Survey Questions. Design and Evaluation. Sage (Applied Social Research Methods Series, Nr. 38), Londen.

Giesen, D. (2004). Evaluating the Annual Business Inquiry Questionnaire: Results and recommendations. (In Dutch: Evaluatie vragenlijst productiestatistiek: Resultaten en aanbevelingen.) Statistics Netherlands, Methods and Informatics Department, Heerlen.

Giesen, D. (2005). Results from the Dutch Annual Business Inquiry 2005 Pre-test. (In Ducth: Resultaten Pre-test PS2005.) Statistics Netherlands, Methods and Informatics Department, Heerlen.

Hedlin, D., T. Dale, G. Haraldsen, and J. Jones (2005). Developing Methods for Assessing Perceived Response Burden. Research report. Statistics Sweden, statistics Norway, and UK Office for National Statistics.

Haraldsen, G. (2004). Identifying and Reducing Response Burdens in Internet Business Surveys. Journal of Official Statistics, Vol. 20, Nr. 2. (Special Issue on Questionnaire Development, Evaluation and Testing Methods), pp. 393-410.

Krosnick, J.A. (1991). Response Strategies for Coping with the Cognitive Demands of Attitude Measures in Surveys. Applied Cognitive Psychology, 5, pp. 213-236.

Murphy, E.D. (2005). Usability and Accessibility Testing in Support of Internet Reporting. Paper presented at 53rd Conference of European Statisticians, Geneva, 13-15 June 2005; organised by the Statistical Commission and Economic Commission for Europe of the UN Economic and Social Council.

Onat, E., and R. Vis-Visschers (2005), Usability test with the electronic questionnaire of the Dutch Annual Business Inquiry. (In Dutch: Usability test e-PS: Eindrapportage.) Statistics Netherlands, Division of Business Statistics, Development and Support Department, Heerlen.

Punselie, R. (2003). Making Web sites work. (In Dutch: Websites die werken.) Kluwer, Alphen aan den Rijn, Netherlands.

Schonlau, M., R.D. Fricker, M.N. Elliott (2002). Conducing Research Surveys via E-Mail and the Web. Rand, Santa Monica, Ca.

Snijkers, G. (2002), Cognitive Laboratory Experiences. On Pre-testing Computerised Questionnaires and Data Quality. Ph.D. Thesis. Utrecht University, Utrecht / Statistics Netherlands, Heerlen.

Snijkers, G., J. Tonglet and E. Onat (2005), Towards an electronics questionnaire of the Dutch Annual Business Inquiry. (In Dutch: Naar een elektronische vragenlijst voor de Productiestatistiek.) Statistics Netherlands, Division of Business Statistics, Development and Support Department, Heerlen.

Van der Geest, Th. (2001). Web Site Design is Communication Design. John Benjamins (Document Design Companion Series, Vol. 2), Amsterdam.

Vroom, B. (2002). Checklist for well-designed web sites. (in Dutch: Checklist voor goede websites. Kluwer, Alphen aan den Rijn, Netherlands.

Willimack, D.K., L. Lyberg, J. Martin, L. Japec, and P. Whitridge (2004). Evolution and Adaptation of Questionnaire Development, Evaluation, and Testing Methods for Establishment Surveys. In: Presser, S., et al. (eds.). Methods for Testing and Evaluating Survey Questionnaires, pp. 385-407. Wiley, New York.

Willimack, D.K., E. Nichols, and S. Sudman (2002). Understanding Unit and Item Nonresponse in Business Surveys. In: Groves, R.M., et al. (eds.). Survey Nonresponse, pp. 213-227. Wiley, New York.

CASI questionnaires with Blaise 4.7

Gerrit de Bolster (Statistics Netherlands)

1. Introduction

With the introduction of Blaise 4.7 new powerful features came available. The possibility to define your own menu options and buttons and to invoke through these options new processes like Maniplus set-up's including data interchange allowed us at Statistics Netherlands to create multi-layer questionnaires. This paper describes the structure of these multi-layer questionnaires for the Production Price Index survey and for the Inland Shipping survey created for our CASI-tool EDR and will focus on the new features of Blaise 4.7 applied herein. Also included is a description of the electronic questionnaire for the EU-survey on Continuing Vocational Training applying some of the new functionalities introduced with Blaise 4.7.

Besides the possibility of creating multi-layer questionnaires a lot of other improvements were made in Blaise 4.7. Many of them were gratefully used in the newly created CASI questionnaires for EDR.

2. The requirements

2.1. Form-based actions

Already for some time the subject matter departments using electronic questionnaires for CASI were asking for the possibility to start-up actions directly from an opened form. Actions e.g. like printing the form. In our CASI-tool EDR we generate a RTF-file and invoke the current text editor (mostly MS-Word) to show and print that generated file. The RTF-file is generated with a Maniplus set-up and populated with the data from the form. Until the birth of Blaise 4.7 this set-up was invoked by a menu option in the main (Maniplus) set-up using the stored data from the form. For a respondent it was not always clear that he should first close (and store) the form before he could print it. Although before version 4.7 it was also possible to activate a process from a menu option in the form (but not using a button!) there was no possibility to pass on the data from the current form. Not as a record and also not as parameters.

Similar to printing it was also requested to create for the respondent the possibility to send the current form (using CBS encryption and communication software included in the CASI-tool EDR) after filling in the last answer. For the reasons explained before the respondent had to close the form first and send it using the main menu. In this case it was even necessary to for fill some conditions (like filling in the mandatory questions) before this function should be enabled.

2.2. Multi-layer questionnaires

More than a year ago the department responsible for the Producers Price Index survey (PPI) came with a request to create an electronic form. Under the pressure for being more cost-effective they saw this (and rightly) as the only possible way. Several unsuccessful attempts were made to create a form using Blaise 4.6 to meet this need. It appeared to be impossible to solve the main problem regarding the PPI: the need for a master-detail solution. The PPI survey is asking the enterprises to fill in their prices for a set of products produced or imported by them. For that purpose an enterprise receives 1-3 slightly different forms (sales interior, sales

abroad and import) depending on its activities. The main part of the form is a list of the products (max. 60) for which they have to fill in their prices. In the paper questionnaire that is the ONLY form. Every product is defined by 9 attributes (slightly different between the 3 forms) like product name, quality and quantity and so on. The respondent must be able to change each of the 9 attributes to introduce a new similar product if the old one is not produced anymore. Also he must be able to add new products and to de-activate (temporarily or for ever) a product. On the paper questionnaire it was done by the most powerful tool a human being ever created: the pen! Just by scratching on the paper and writing what ever he wants next to it he could for fill his task. In the electronic form the 9 attributes appeared to be too extended to place them as question fields on the main form e.g. in a table. The respondent would loose the overview. Several attempts were undertaken to apply the Rules section to switch between a main form with only a summary of the attributes and a parallel with more detailed fields. All attempts remained unsuccessful. There was no way we could create something like an auxiliary form.

At the same time the department responsible for the surveys on Traffic and Transport came with a similar problem. They received all kind of paper forms from lockkeepers and bridgemen reporting about inland shipping movements. These movements were grouped by a so-called event: the lockage (of the lock) or the opening of the bridge. Information should be provided as well on the level of an event as on a ship (or group of ships). They were also looking for 2 electronic forms to support this two level (or multi-layer) questionnaire. As always, the paper forms used were very “flexible”. Mainly one level (very large) matrixes in which the event information was written only on one line while the data about the ships were spread over several lines. Of course, additional (but crucial) information was written everywhere. Only a (costly) human being could understand the information written on these paper forms.

2.3. The adjustable index

Finally a third “customer” reported itself: the department responsible for the EU-survey on Continuing Vocational Training (CVTS). As it was known that this questionnaire with rather complicated questions could not be filled-in in one session by the respondent, it was decided to apply our CASI-tool EDR. The main issue in this questionnaire was the need of dynamic routing. The questionnaire consists of 7 different parts. Depending on the answers given in the second part rules determine which other parts should be answered by the respondent. As it was meant to be a CASI questionnaire it was very important that the respondent knew at all time at which point he was in the questionnaire. The length of the questionnaire made it not suitable for a sequential solution in which the active parts followed each other as the respondent was filling in the questionnaire. As methodologists in Statistics Netherlands disapproved the use of tags (with parallels) in case there were a number of it (as in this case), an alternative with a main adjustable index screen was created. The navigation through this questionnaire was made easy thanks to the new functionality of Blaise 4.7.

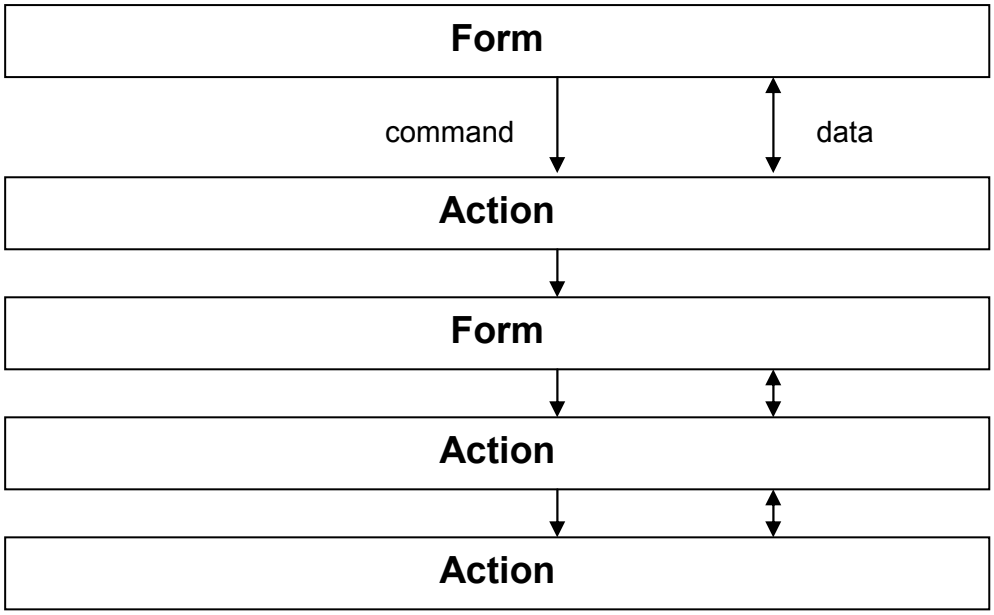
3. The solutions

3.1. The form/action architecture

The main improvement in Blaise 4.7 was the possibilty that from within an opened form an action can be invoked. Not only can the action be invoked but also the data in the current form can be transferred to that invoked Maniplus set-up. This set-up on its turn can start up another (sub-) form pre-populating it with data from the parent form. And finally it works two-ways: the data from the sub-form, after editing, can be transferred back to the parent form. And even more: from within a

Maniplus set-up another set-up could be invoked transferring (to and from) the current record of a temporary file.

This functionality makes it possible to create an architecture like this:



Any set of "layers" is possible. Between two forms there is always an "action" (set-up) to transform data to the other model.

This architecture was applied in the three questionnaires I mentioned before. In the next paragraphs I will go in more detail for each questionnaire to show how the multi-layer model was solving the typical problem.

3.2. The Producers Price Index (PPI) questionnaires

The 3 questionnaires of the PPI survey are completely the same with the exception that only a few attributes of the products are different. The following description is therefore valid for all three the questionnaires.

The main form of the questionnaires is the list of the products (or price items) for which the prices should be given. This main form consists of a data model containing a TABLE of (maximum) 60 rows. Every row refers to a BLOCK with contains the fields for the product attributes and the related price. Furthermore it contains two auxiliary string fields. These auxiliary fields are filled with a selection of the values of the fields with the product attributes. Only these auxiliary fields and the price field are shown together with a field for the status of the price item and the currency used for the price. Except for the price field all the other fields can not be edited in this form.

The initial set of price items is sent with the form to the respondent. If a respondent wants to change the attributes of a price item he focuses on that row and activates through a menu option or a button a Maniplus set-up. Which row is focused is determined by the invoked set-up using the ACTIVEFIELD statement. This statement produces the name of the focused field including the index of its occurrence in the table. This index is derived from its name.

In the invoked set-up a Blaise file is opened in a temporary folder using a separate data model with fields for all the attributes of a price item. A record is written filled with the data from the fields from the focused row of the table of the current record

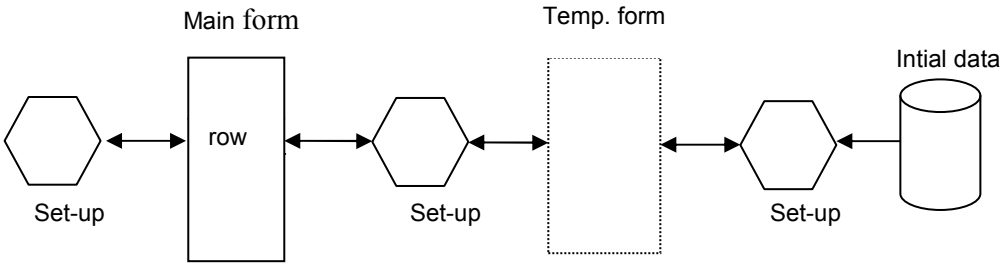
of the parent form. The current record is available through a temporary file with the INTERCHANGE=TRANSIT settings. Applying the EDIT statement the resulting form is shown on the screen and the data can be edited. (Because in 4.7 a temporary file can not be used in an EDIT statement a real Blaise file had to be created on the spot. After finishing the action it is deleted again. In Blaise 4.8 this is not necessary anymore as a temporary file can also be used in an EDIT statement.)

After closing the form the Maniplus set-up is checking the HISTORY field to see if the form was changed. If that is the case the data is copied back in the temporary file to pass it on to the parent set-up. The SETUSERCHANGE statement is used to inform the parent form that it has been changed.

In a similar way a new price item can be added creating an empty record in stead of a pre-populated record. This is done by invoking the same Maniplus set-up with another value for a parameter altering its process. The new price item is added at the next empty row in the table in the parent form. If there are 60 price items (the maximum) stored the “add” option is disabled.

The value of the status field of the table row is changed using a Maniplus dialog in stead of a form. Again it is done by invoking the same Maniplus set-up with another parameter value. Changing the status the respondent can communicate that the price item is temporarily or definitive inactive. He is not allowed to remove a price item initially provided by Statistics Netherlands. This is done by the survey department based upon the status information provided by the respondent, when a new set of price items for the next period (can be a month, a quarter or even 6 months) is sent to the respondent. The respondent can however remove a price item that was added by him before.

Furthermore the attributes of a price item can be set back to its original values (as was initially provided by Statistics Netherlands) invoking another action. If it is done while the respondent has the sub-form with the detailed attribute fields opened another Maniplus set-up is invoked to perform this action. In fact we can speak of 5 layers: (EDR-menu ->) Set-up1 -> Form1 -> Set-up2 -> Form2 -> Set-up3.

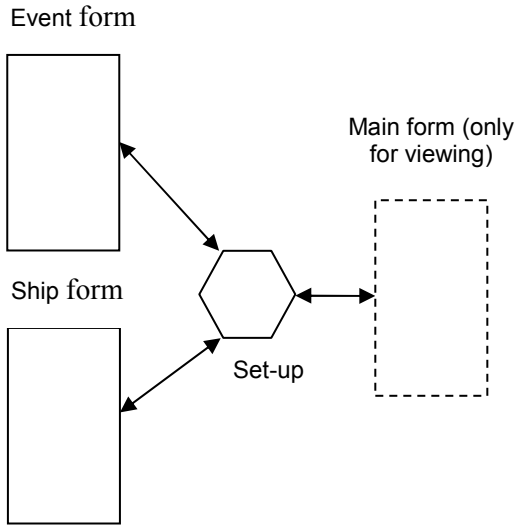


The 3 different PPI CASI-questionnaires were put in action in February of this year and according to the PPI department quite successfully.

3.3. The questionnaire for locks and bridges

Different from the PPI questionnaires the main form in the (monthly) questionnaire for the ship movements at locks as well as in the one for the bridges does not contain a field that can be filled. Thanks to new functionality in Blaise 4.7 a user is still able to select (focus) a line in the table. Both questionnaires have the same architecture, only some fields in the forms are different.

The main form is a TABLE of (maximum) 1000 rows to show the events and their ship movements. It only contains a view on the events and ship movements; the real data is stored in a separate Blaise files. One for the events and another for the ship movements. In the main form the data of an event as well as of a ship movement in stored in a string field. The ship movement data is slightly indented giving a hierarchical view. For every action like editing, adding or removing an event or ship movement a Maniplus set-up is activated. In the same way as in the PPI questionnaire this set-up detects with row is focused. However, in this questionnaire the real data is stored in separate Blaise files and, depending on the requested action, these files are edited. Closing an action the table in the main form is rebuild by this set-up deriving the necessary data from the separate Blaise files. In the event file a toggle field is present which value indicates if the related ship movements should be added to the main form or not. The value of this field is also changed using a button (or menu action) in the main form. The effect is that the respondent can show or hide the ship movements for one or more events.

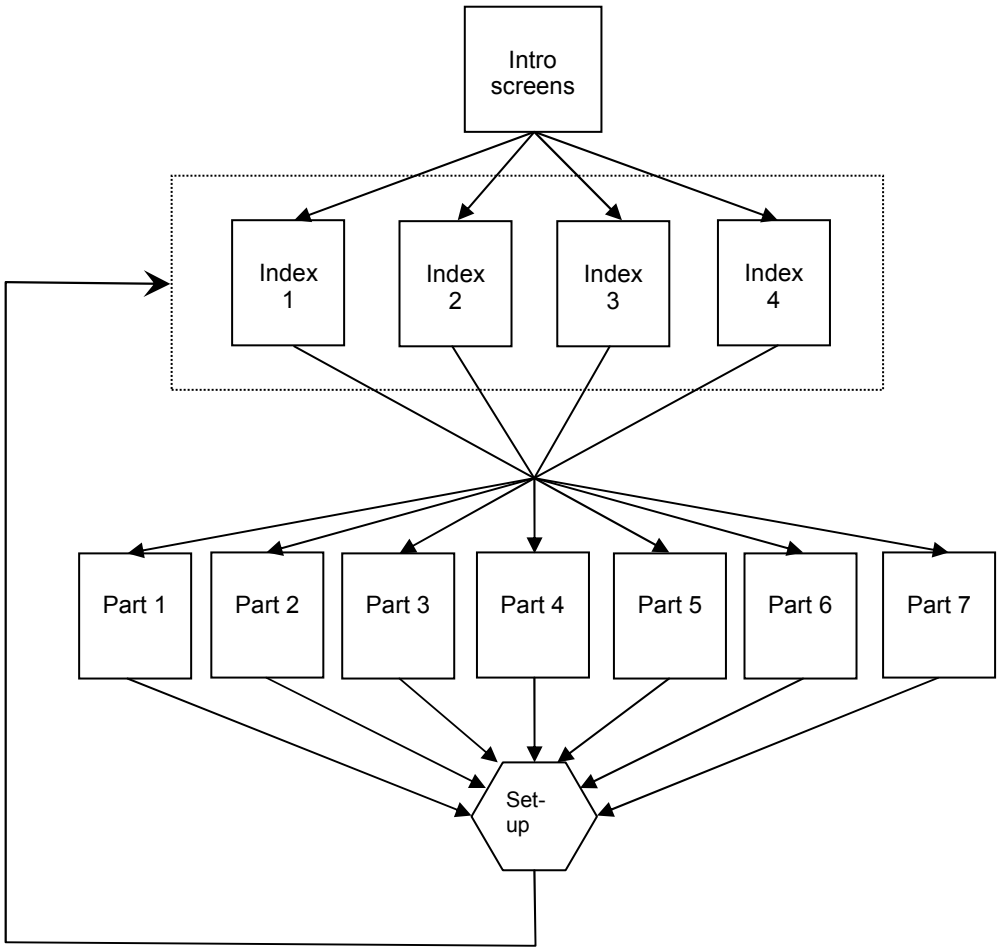


During the development it became clear that the majority of the questions in these questionnaires were not used by Statistics Netherlands but by the Ministry of Traffic and Transport. Due to that and some other discussions about maintenance the survey department decided not to go on with these questionnaires.

3.3. The CVTS questionnaire

In the CVTS questionnaire the 7 different parts are created as 7 BLOCK's in the main form. Additionally a BLOCK containing the Index as a enumerated question is added. In fact there are 4 different Indexes. Depending on the values in the fields of part 2 one of these indexes is put on the route. The first two parts are always included. By selecting a value in the Index the related part is put on the route just after the Index. All the other parts are then off-route (although their data is not lost as they are still on the route through KEEP statements). As a consequence the respondent is jumping directly to the first screen of the selected part.

At the end of the active part the respondent also reaches the end of the questionnaire. In Blaise 4.6 he had to browse back to the Index screen by screen (as a part normally contains more screens). Thanks to Blaise 4.7 we can now apply a button or menu option which activates a Maniplus set-up transferring the data. With the SETACTIVEFIELD statement we change the active field to the active Index field and return to the form. As a result the respondent jumps directly to the Index, were ever he was positioned in the form.



It is planned that this questionnaire will be operational in the second half of April.

5. Concluding remarks

With the release of Blaise 4.7 we were given the opportunity to create CASI questionnaires that were not possible before. As we have proven already with the PPI questionnaires they for fill the needs of our statistical departments as well as of our respondents. The on-going integration of DEP and Maniplus will allow us to create with Blaise in the near future without extensive resources any kind of powerful questionnaire that most certainly cannot be created with any other language or tool.

Basil, A New Tool for CASI in Blaise

Roger Linssen & Jo Tonglet (Statistics Netherlands)

1. Introduction

In the past decade, computer assisted self interviewing has become more and more popular. CASI is a cheap way to get data from respondents: you don't need expensive call centres or interviewers.

Statistics Netherlands is experimenting with different types of CASI surveys. One of the experiments was the use of a downloadable program that handles the survey and sends the data back to Statistics Netherlands through the internet.

2. Requirements

There were several user requirements for such a downloadable program, listed in Appendix A. A couple of important requirements:

- Size < 2 MB
- No reboot after installation
- Reusable for different surveys
- Data transmission through internet or e-mail
- Lots of specific layout requirements (see also Appendix B)
- Time to complete the project: 5 months

3. Project Decisions

After carefully examining the requirements, it was decided that the program should be programmed in Delphi. Blaise is programmed in Delphi, and therefore, all Blaise functionality could be used in the program. Also, Delphi can produce stand-alone executables, which don't require any run-time libraries, and therefore an easy installation is guaranteed.

The next decision was to handle all layout requirements within the Blaise language. It was not possible to change the Blaise compiler within such a short period of time, but it was possible to use the Blaise language concept to store layout information. Another important decision was to use the Blaise paging mechanism, instead of creating a new algorithm. An 'index' control, which Blaise doesn't have by default, had to be created, and linked to the paging mechanism.

It was soon clear that some form of event handling was necessary: the layout requirements included buttons, hyperlinks, help icons, etc. It was decided to use Manipula (or rather, Maniplus) to facilitate event handling. This decision had major consequences: all event handling had to written by the programmer of the data model. However, this way TEPS (Tool for Electronic Production Statistics) could be a generic program, which could be used for all sorts of data models. Dialogs, error handling, printing and even transmission of data could be handled by Maniplus.

4. Prototyping

In the Netherlands, people can fill out their tax form using a program similar to TEPS (see Appendix C). This Tax Disc can be downloaded and installed on a PC, handles the tax 'questionnaire', and sends the data back to the tax department. All in all, a perfect prototype.

Typically, the tax questionnaire has an index and an image on the left side of the screen, and a content area on the right side of the screen. The bottom part of the screen contains a navigational area and a menu area.

4.1 Application

The first step was to define an 'application definition', in which a specification of the global layout of the instrument could be made. This specification is XML-based. The Blaise datamodel description was the ideal place to put this definition:

```
DATAMODEL Test
"<application width=800 height=600 Title='Test Application'>
   <panel align=left width=100 color=blue>
     <panel align=bottom height=100>
       <img src='LOGO' left=0 top=0 width=100 height=100>
     </panel>
     <panel align=client>
       <index-area>
     </panel>
   </panel>
   <panel align=client>
     <panel align=bottom height=25 color=blue>
       <button left=0 top=0 height=25 src='EXIT'>
     </panel>
     <panel align=client color=green>
       <content-area>
     </panel>
   </panel>
 </application>
"
...
ENDMODEL
```

This code snippet shows how to define an instrument, divide it into areas, and put controls (image, button, index and content) on the areas. It is possible to divide areas (panels) into sub-areas.

4.2 Content area

Next, the content-area needed to be filled with (Blaise) questions. To achieve this, a question layout definition was needed. A question can be seen as a container, which holds one or more elements. The question text part of a question definition is used to describe the layout of a question:

```
FIELDS Q
"<question>
   <label left=0 top=0 width=100 height=25 text='Name?'>
   <input left=150 top=0 width=75 height=25>
 </question>
" : string[40]
```

This code will define a question with two display elements: a label and an input control. The type of input control depends on the type of the Blaise field. In this case, an edit box will appear, because the type of the Blaise field is string. If the Blaise type was an enumeration, the input control would have been a radio button group. It is possible to change the look and feel of the default control by setting its properties, but it is also possible to use a different control. For an enumeration, for

instance, it is possible to use a dropdown control, a checkbox (useful if there are only 2 options in the enumeration), or a button.

4.3 Index area

The Index area was especially tricky. It should be possible to define the index in the Blaise language, but Blaise has no concept of an index. It was decided to use Blaise block field descriptions to represent the index. Nested blocks can then be used to add structure to the index.
The following piece of code shows how to define an entry in the index:

```
BLOCK BBlock
  {Field Definitions}
  ...
ENDBLOCK

FIELDS
  B "<index>
       <img left=0 top=0 width=16 height=16 src='ICON'>
       <label left=20 top=0 width=40 height=25 text='Section 1'>
     </index>
    " : BBlock
```

The index element consists of an image and a label. By clicking on the index element, the first eligible field of BBlock will be focused.

4.4 Event handling

The next challenge was to add event handling to the data model: when certain events occur, a piece of (procedural) code should be executed. For instance, by clicking on a button, data could be saved to disk, or sent to a web server. It should also be possible to create dialogs in an event handler, which present a user interface to perform certain tasks. For instance, when the user tries to leave the program, a custom dialog could pop up which asks the user if he's really sure he wants to exit.
Blaise does not support event handling. However, one of the Blaise tools, Maniplus, has all the characteristics to be used for event handling: it is a procedural language, and it can create user interfaces.
In the application section, one can define the Maniplus setups that are used for event handling

```
DATAMODEL Test
"<application width=800 height=600 setups='Events.msu'>
   ...
 </application>
"
```

The Maniplus setups that are specified in the application section are preloaded during initialization of the program, and are kept in memory after that. This way, the performance is sped up significantly.

The actual event handler, written in Maniplus, could look something like this:

```
PROCESS Events
USES
  dmTest 'Test'

TEMPORARYFILE tfTest : dmTest
SETTINGS
  INTERCHANGE = SHARED

DIALOGBOX DlgSave "Save"
  ESCAPE = NO
  SIZE   = (370, 100)
  COLOR  = '239 255 255'
  TEXT   = ('Press the OK button to save your data', 10, 10)
  BUTTON
    CAPTION = 'OK'

PROCEDURE SaveData
  dlgSave
  tfTest.WRITE
ENDPROCEDURE
```

This event handler will show a dialog first, after which a record is written. To use this event handler, for instance when a button is clicked, assign the event handler in the data model:

```
DATAMODEL Test
"<application width=800 height=600 setups='Events.msu'>
   <panel align=bottom height=50>
     <button left=0 top=0 width=75 height=25 src='SAVE'
             onclick='Events.SaveData'>
   </panel>
   ...
 </application>
"
```

4.5 Variables & Expressions

Blaise has the possibility to put variables (fills) in question texts. This way, a piece of text can vary depending on the value of a field. TEPS extends this concept by allowing two other fill types to be used in texts: $-fills and %-fills.

Both new fill types require the fully qualified field name:

```
BLOCK BBlock
  FIELDS
    Q1 : string[30]
    Q2 "<question>
          <label left=0 top=0 width=100 text='^Q1'>
          <label left=0 top=50 width=100 text='$(B1.Q1)'>
          <label left=0 top=100 width=100 text='%(B1.Q1)'>
          <input left=0 top=150 width=100 height=25>
        </question>
       " : integer[1]
ENDBLOCK

FIELDS
  B1: BBlock
```

This will create an area with three labels and an input control. Each label will have the value of Q1 as text. The normal fill (^Q1) does not require the fully qualified name, whereas the other two fills do.

If the structure of the question (e.g. the components it is composed of) doesn't change, then the %-fill is preferable. If a $-fill is used in a question text, the structure of the question is rebuilt every time fills are calculated. This means that the screen may flicker.

Notice the (optional) parenthesis around the fully qualified fieldname. This allows you to add special characters directly after the fieldname:

```
<label left=0 top=0 text='Year:$(Year).'>
```

This way, the dot will not be interpreted as being part of field Year.

The $ and %-fills also support expressions:

```
FIELDS
  NrOfPeople: integer[2]
  Q "<question>
       <label left=0 top=0 visible='%NrOfPeople>0'>
       <input left=0 top=150 width=100 height=25>
     </question>
    " : integer[1]
```

The label will only be visible when NrOfPeople holds a value greater than zero.

5. Electronic Production Statistics

After prototyping with the Tax Disc, it was time to implement a Statistics Netherlands questionnaire. Five EPS questionnaires were chosen for to be implemented 'by hand'. After that, it should be possible to generate the remaining 188 questionnaires from a meta server and a datamodel server using the first five questionnaires as a template.

Because the design of the tool was such, that it could be reused for multiple questionnaires, the decision was made to ship the tool with Blaise. The name TEPS (Tool for Electronic Production Statistics) changed rapidly into BASIL (Blaise Alternative Self Interviewing Language).

The design of the prototype proved to be very solid. By inventing an XML-like language, it was possible to add new keywords and properties very easily. Implementing the first four questionnaires proved to be fairly easy.

After the completion of the first EPS questionnaire, it was evaluated by 10 respondents. This resulted in several additions and corrections to Basil:

- Improved navigation
- Facilitations to make an audit trail
- Index improvements
- Additions to Maniplus, which made it possible to access all meta data of a questionnaire

During installation of the five questionnaires, Basil was also used to display a username/password dialog.

In March 2006, all five questionnaires went in production. They were sent to (or rather, downloaded by) 7400 respondents. The results thus far are very satisfying. Approximately 95% of the respondents use the internet to send data to Statistics Netherlands. The rest uses e-mail.

6. Conclusions

It proved to be possible to meet a deadline of half a year. This was accomplished by an intensive cooperation between developers, questionnaire designers and layout specialists.

Most of the requirements as mentioned in Appendix A were met. Those that weren't met were deliberately removed from the feature list.

By designing the application as a generic program, it is possible to reuse the tool for different projects, and even ship it as a standard tool with Blaise.

Appendix A: Feature List

Download & installation

- Downloadable through a respondent portal
- Maximum program size: 2 MB
- No reboot obligation after installation
- Large font support
- Default screen resolution: 800 x 600

Login

- Authentication possibility (no obligation) at login using a password
- Automatically reload previously filled out data at startup

Design

- Different panes with different colors:
  - Pane for index
    - Tree view (index)
    - Expandable tree
    - Hyperlink possibility
    - "Teletext" principle (numbers)
    - Only one chapter expanded at any moment
    - Possibility to place information (company info) at any position on the index pane
  - Pane for content
    - Possibility to place question and answer at arbitrary positions within the pane
  - Pane for navigation
    - Possibility to place clickable navigation buttons at arbitrary positions
  - Pane for menu
    - Possibility to place clickable menu buttons at arbitrary positions

Navigation

- Clear navigation
- Overview of the questionnaire + clear indication of the position in the questionnaire
- Navigation with both mouse and keyboard

Data Entry

- Confirm possibility (confirm button) on a page
- Immediately save entered data (even before confirmation)
- Progress indicator that indicates what part has been filled out and what part still has to be filled out/corrected
- Remark field, available for every answer
- Calculator field functionality: available for every answer (when applicable)
- Roll-up menu (lookup file), table from which selections can be made

Explanation

- Available with a single mouse click or keystroke
- Possibility to add a clickable or non-clickable button/image to every question/answer
- Explanation must be able to appear at any position
- Explanation is removable with a single mouse click or keystroke
- Windows Screen must be available
- A pop-up must be able to appear at any moment
- "Google-like" search function at any place in the questionnaire

Auxiliary Functions

- Windows Calculator (in menu area), callable with a mouse click or keystroke

- Printing of the questionnaire at different levels, for instance:
  - Empty form
  - Filled out form
  - Chapter
  - Clarification

Transmission

- Transmission:
  - option through internet (default)
  - option through email

Appendix B: Layout wishes

Figure 1: Layout structure

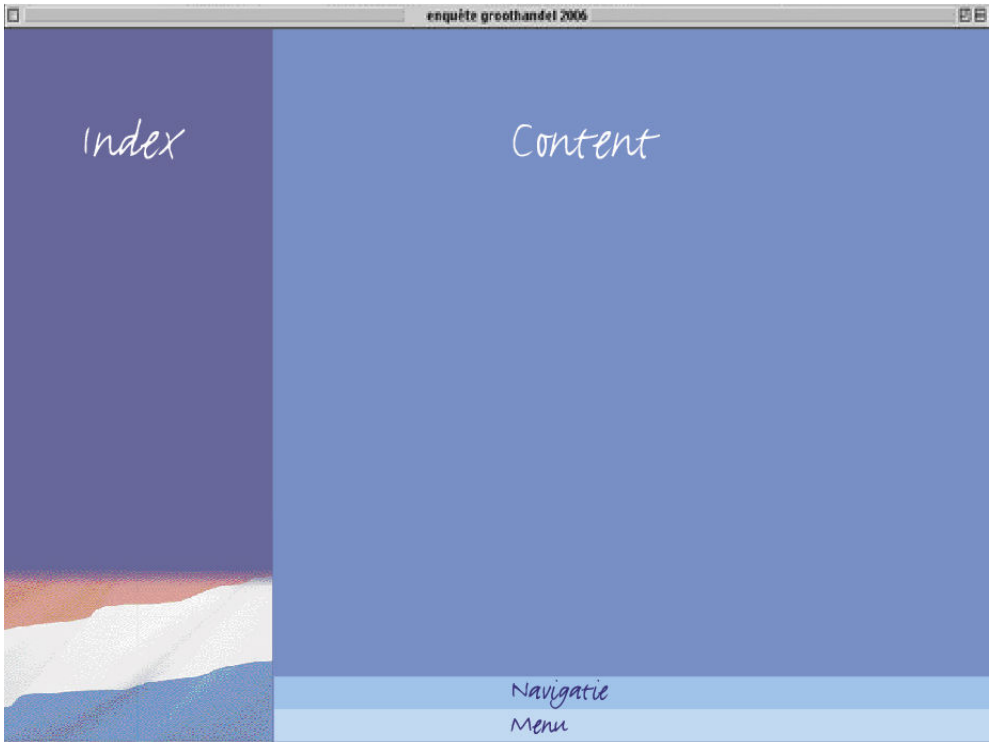


Figure 2: Sample page

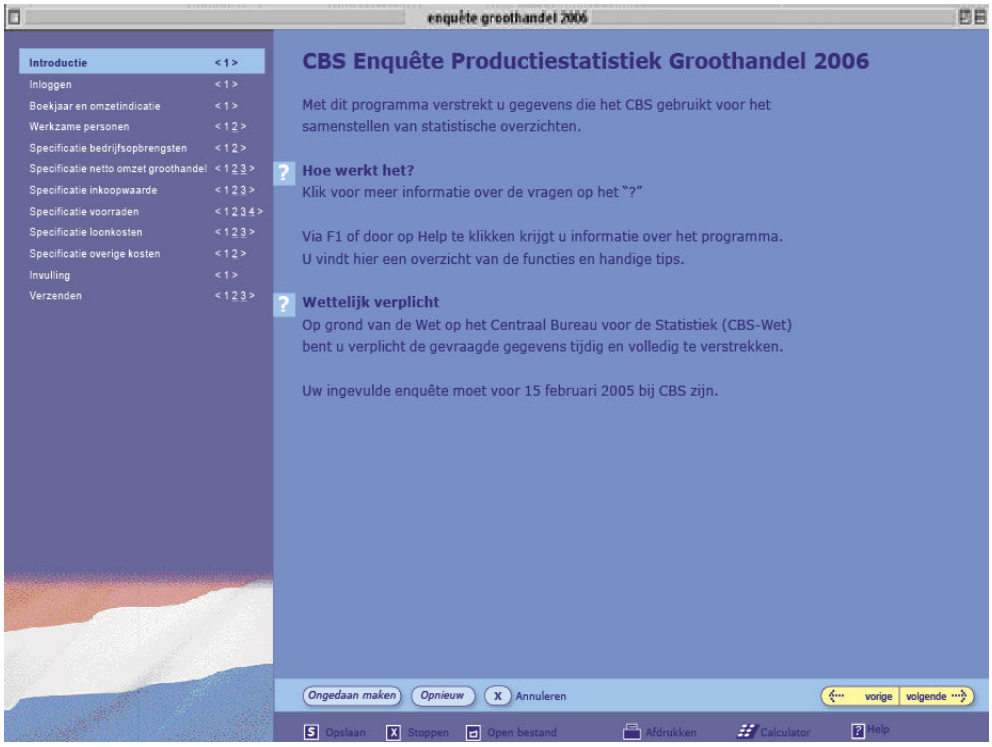


Figure 3: Context Sensitive Help

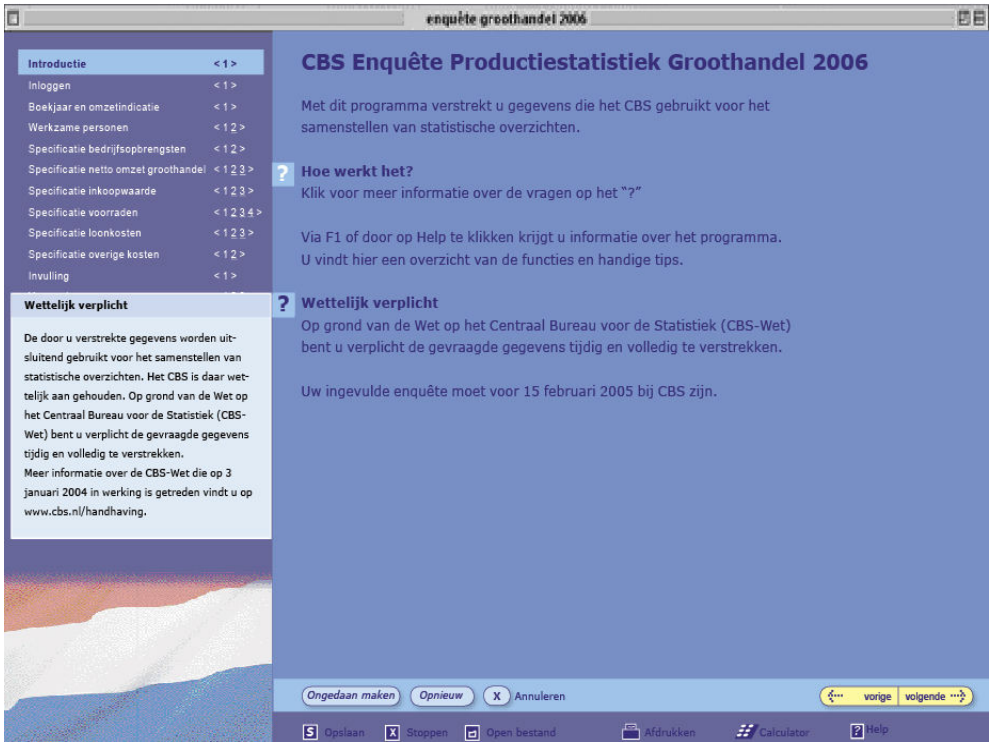


Figure 4: Index & Teletext principle

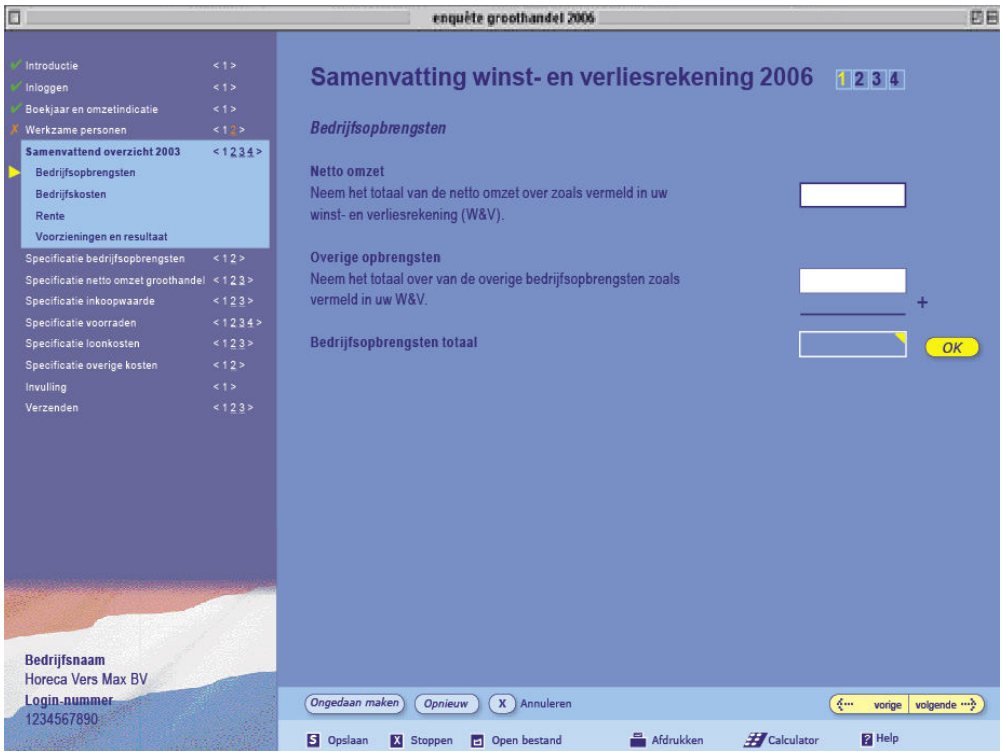


Figure 5: Calculator Fields

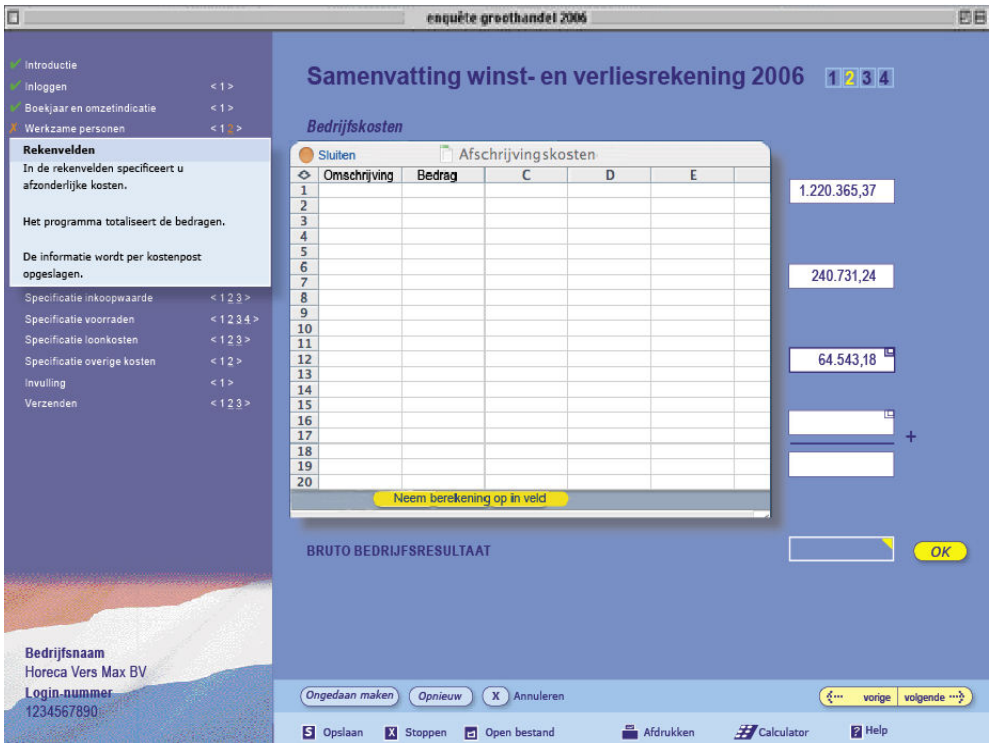


Figure 6: Error handling

Figure 7: Printing

Figure 8: Transmission

Appendix C: Tax Disc Prototype

Figure 9: Tax Disc (Original) Introduction Screen

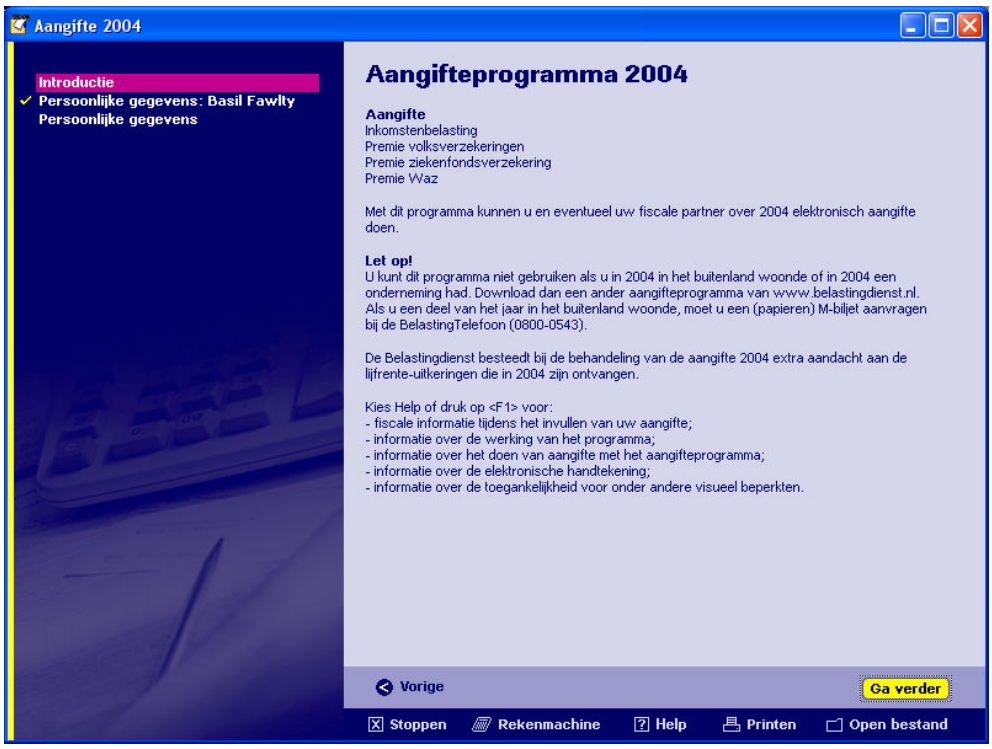


Figure 10: Tax Disc (Prototype) Introduction Screen

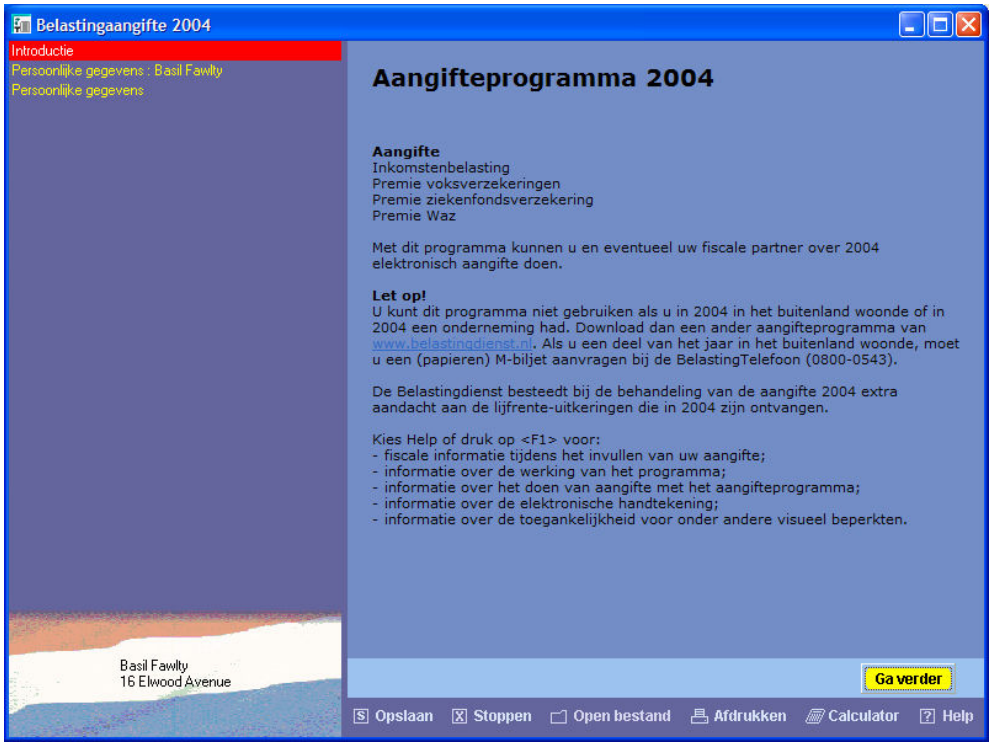


Figure 11: Tax Disc (Original) Sample Screen

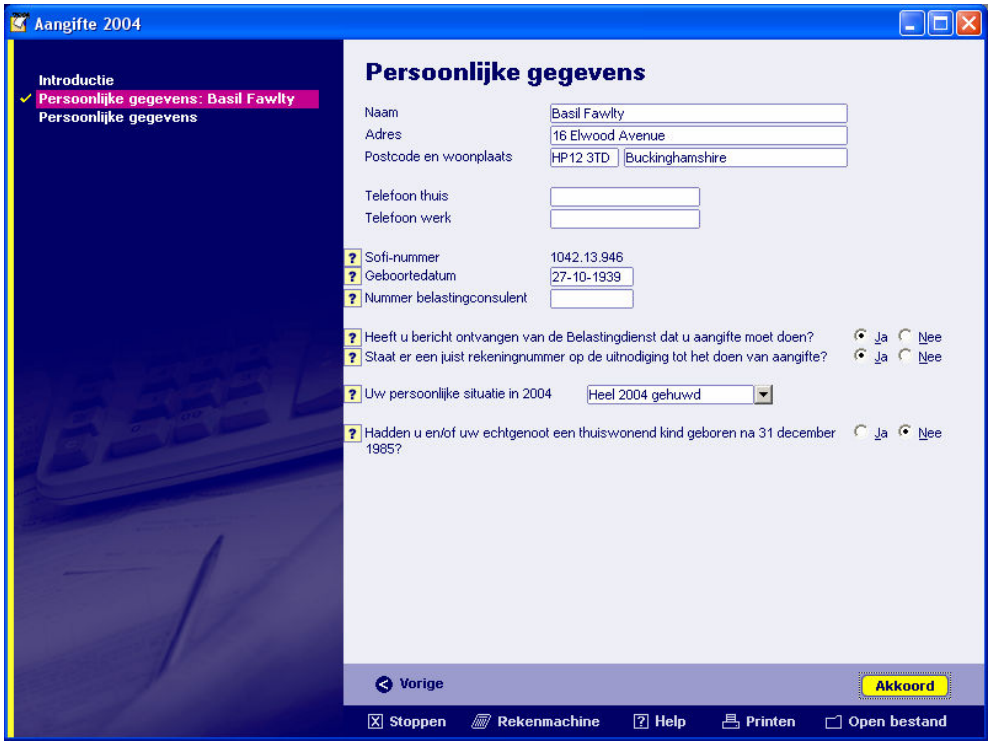
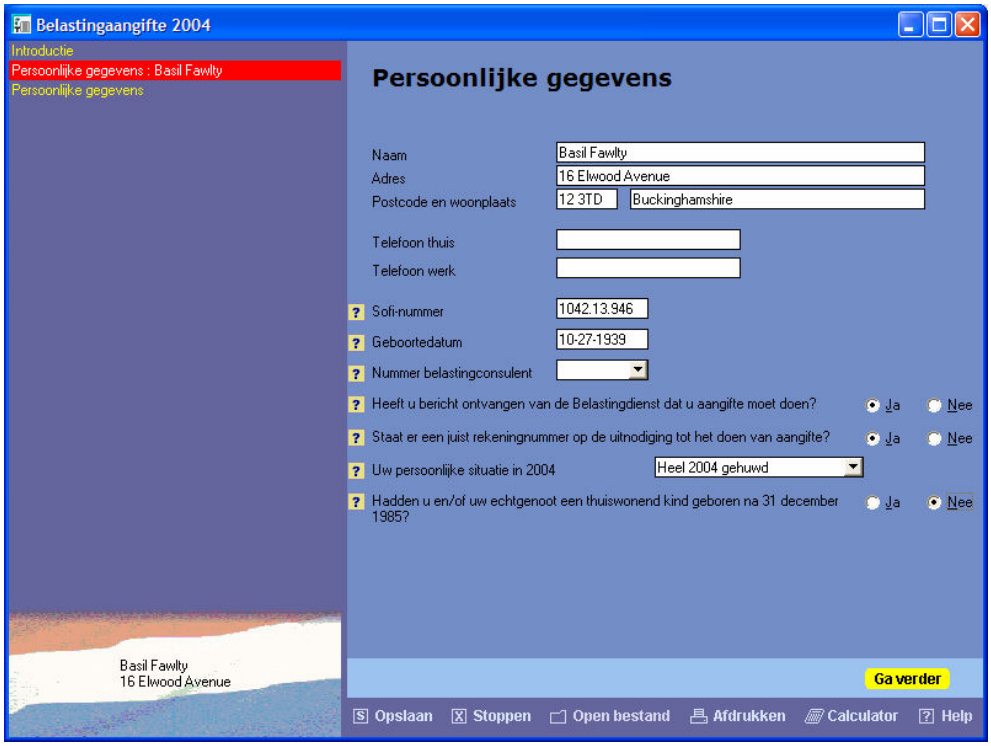


Figure 12: Tax Disc (Prototype) Sample Screen



Session 3.3: Transformations

Computer Assisted Coding by Interviewers
Wim Hacking, John Michiels & Saskia Janssen-Jansen (Statistics Netherlands)

MultiRelaX with BCP as a replacement for Cameleon
Marien Lina (Statistics Netherlands) & Richard Frey (Westat)

Computer Assisted Coding by Interviewers[1]

Wim Hacking, John Michiels & Saskia Janssen-Jansen (Statistics Netherlands)

1. Introduction

Coding activities play an important role in a statistical production process. These activities are usually associated with the assignment of responses to predefined codes in a classification, so that these responses become available for further data-editing operations. Coding can be done at various stages of statistical production: during data collection by respondents or interviewers or during the data-editing process by coding experts and/or automated systems. In most cases coding is a rather trivial exercise: when questions are clear and unambiguous and the number of answering categories is limited an answering category can easily be provided by respondent or interviewer. The situation becomes more difficult when the number of answering categories is large or several open text answers are required. Examples of these more difficult coding activities are the assignment of a respondent's educational background or occupation to a corresponding classification. In these examples usually more than one question is involved in gathering the information required for assigning valid codes and part of these questions are of the open text type.

In previous years the approach adopted at the *division of social statistics* for coding (open text) responses from a number of related questions was the following. First, the answers to these questions are collected using CAPI or CATI modes of data collection. In the case of long answers they are interpreted and modified by the interviewer to obtain a concise response. At the statistical office the collected and sometimes edited information is then fed to a 'batch' process for automated coding. The records that are not coded are classified interactively by coding experts in a second pass. Coding economic activities at the *division of business statistics* was partly done manually at Statistics Netherlands and partly externally, at the chambers of commerce.

In this paper we would like to describe alternative coding techniques that are more successful than the traditional approach. The technique chosen depends on a number of factors (and possibly more than given here):

- The desired level of detail for statistical output (publications, international obligations) and for intermediate processes (e.g. weighting).
- The desired quality of coding (error rates).
- The available budget and expertise for development, implementation, operation, and maintenance.

The first point is relevant because for obtaining low detail classifications in most cases less elaborate methods can be developed that produce good results. The desired quality puts similar restraints on the choice of coding method. As higher demands on detail and accuracy require more complex coding techniques, it will in general also mean higher development costs.

In this paper we will discuss an alternative coding strategy that has been developed and selected for implementation at Statistics Netherlands: Computer Assisted

[1] The views expressed here are those of the authors and do not necessarily reflect the position of Statistics Netherlands

Coding by Interviewers (CACI). The new strategy is more cost effective than traditional approaches with comparable level of detail and reliability. The available expertise for implementing this approach can usually be found at a statistical office; if not, the required effort to obtain this knowledge is certainly less than for some more advanced methods in machine learning or for expert systems. In the next section we will describe the methods that have been applied for semi-automatic coding. In section 3 we will discuss the interactive coding approach as applied in the interview process at Statistics Netherlands, based on the methods from section 2. Finally, section 4 shows results using these new techniques. Here the percentage of coded records and coding reliability for each technique will be considered, along with results on interview lengths for CACI (based on field work done by the division of social statistics). In addition, some early results for the coding of economic activity at the division of business statistics are given.

2. Coding Techniques Used

We will start by giving an overview of the coding techniques used. The merits and drawbacks of each technique will also be discussed in brief. Basically, three coding techniques have been chosen depending on the (coded) material at hand: in 2.1 we discuss the situation that (many) previously coded records are available in electronic form. In 2.2 we start with a registry containing an extensive description for each code. In 2.3 no electronic material is available and a search file is constructed specific for the coding process. Depending on what initial material is at hand methods have selected for semi-automatic coding.

2.1. Using previously coded material

A number of techniques for automatically classifying text strings has been developed and implemented by the Institute for Knowledge and Agent Technology (IKAT) at Maastricht University: Nearest Neighbours (NN), Term Frequency Inverse Document Frequency (TFIDF), and Naïve Bayesian (NB) (Smirnov 2003, Kaptein 2005). All of these techniques involve learning algorithms that need a training data set with combinations of text strings and corresponding codes for ‘optimization’ or ‘training’. The nature of this optimization depends on the technique being considered. Training a learning algorithm produces a text-classifier with an approximate mapping of text strings to codes. The text-classifier assigns a weight or probability to each code in the classification and the code with the largest weight or probability is then selected.

The problem is to determine the reliability of this type of classification. There are ways to do this using another kind of learning-machine called meta-classifiers. These classifiers use ‘meta-information’ from text-classifiers: in case the code assigned by the text-classifier equals the true code in the training data set the combination of text string and code vector is labelled as ‘good’, otherwise it is labelled as ‘bad’. The meta-classifier uses this kind of information in order to optimize a set of rules for correctly assessing text-classifier results: with these rules the meta-classifier decides whether the codes assigned by the text classifier for new text strings are to be considered as ‘good’ or as ‘bad’. There are different ways in which these rules can be constructed and hence there are different meta-classifiers. Examples of meta-classifiers are described in the thesis of Kaptein 2005 and the main results are mentioned in section 4.2.

Instead of coding the material afterwards based on classifier techniques we can also apply them during the interview in the field. We will an example to clarify the approach chosen. Suppose that we are interested in coding the open text ‘carpenter’ (as a possible answer to the question: what is your current occupation?). This answer has been given in previous surveys many times before and at the statistical

office it has been assigned a number of different occupation codes (using additional information on job activities). It is now possible to calculate conditional probabilities $P(Code_i|$ *'carpenter')* representing the frequency with which each code has been assigned to 'carpenter'. Therefore to each word in the open text answer a vector of conditional probabilities can be determined. In case the open text contains more than one word the vectors are added to produce a combined vector of weights (not probabilities). For example, in the open text answer 'carpenter at shipyard' there are probability vectors for 'carpenter', 'at', and 'shipyard'. These vectors are added to produce a vector of weights:

Carpenter: $P(code_1|$ *'carpenter')=0,60 ;* $P(code_2|$ *'carpenter')=0,20 ; .....*
At: $P(code_{14}|$ *'at')=0,02 ;* $P(code_2|$ *'at')=0,01 ;* $P(code_{11}|$ *'at')=0,01 ; .....*
Shipyard: $P(code_2|$ *'shipyard')=0,50 ;* $P(code_4|$ *'shipyard')=0,35 ; ..... +*

Carpenter + at + Shipyard : Weight($code_2$)=0,71 ; Weight($code_1$)=0,60 ; .....

Or in a formula:

$$Weight(Code_i) = \sum_j P(Code_i \mid Word_j)$$

The probability vectors for the individual words are stored in index files containing the conditional probabilities $P(Code_i|Word_j)$. After calculating the weights for each code based on the search string, the code descriptions of codes with the largest weights are then presented to the respondent if the combined weight of the first (say) 6 codes is above a certain threshold. The respondent then selects a particular code description and a code is uniquely established. In case the combined weight is too small there are too many codes possible for the given answer and these can not all be presented; in that case it is better to ask additional information in order to limit the number of possible code descriptions.

The main advantage of using text- and meta-classifiers is that they represent a cost-effective technique as far as operational and maintenance costs are concerned. Another advantage is that the use of these classifiers does not require a deep understanding of the classification problem at hand. Accurate training data are not necessary although "noisier" data degrades the classifier performance. However, there are some drawbacks: the cost of developing text- and meta-classifiers can be high. And often the expertise needed for development and implementation is not available at a statistical office. Moreover, a lot of data are needed to train text-classifiers and the technique does not increase our knowledge about the coding process itself.

2.2. Using registries

The most cost effective approach to coding is probably by using results that have already been collected elsewhere. For example, if one is interested in coding the economic activity of one-man businesses in the Netherlands one can link the complete files of the population and business administrations using the social security number as the connecting key. Although in principle registries could work well for coding purposes one usually faces the following problems: the registry does not exist (yet), the registry does exist but is not available to the statistical office, the quality of the linking field(s) or the target variables is insufficient, or the information is outdated. In the latter case there is still need of a questionnaire and the respondent should be asked whether the (outdated) information is still valid. Formulating appropriate questions is not always easy. Also one has to link sampling frame and registry in order to identify sampling elements for which the information in the registry is applicable.
On can also use the registry as a search file which contains information about the item to be coded. For example, in the case of businesses, we use a search file that consists of records containing information about business names (legal and trade

names) and the corresponding codes for economic activity. The search strategy is then to compare open text answers with text strings in the search file and select records that 'match'. Records in the search file 'match' with open text answers if the words in the record description are very similar or equivalent to the words in the open text answer. For example, a respondent may have indicated that he or she is an employee at 'Philips lighting'. This answer is very similar to the record description 'Philips lighting bv' in the search file for business names. One would also like to consider other records with similar descriptions like 'Philips lights nv' or 'Philips displays'. The similarity between descriptions can be measured using a kind of metric (i.e. a kind of distance formula). We applied the 'Levenstein metric' where the difference between descriptions is measured by counting the minimum number of character inserts+deletes+changes that are required to go from one string to the other. Records with descriptions that are similar to the open text answer are presented to the respondent for further selection.

This kind of approach is only applicable if we are looking for terms that the respondent knows relatively accurately (in this case business name) and the existence of a registry that maps those terms to the desired code (SBI in this case). In addition, the quality of the codes assigned depends on the quality of the codes present in the registry.

2.3. Using handmade search files

If no material is available to that links open text to the codes, a search file may be constructed. For each code a definition must be made and stored in conjunction with the code. Using this naïve approach will result in very little hits, since the descriptions given by the respondents may differ greatly. In order to alleviate this problem a synonym file was added that translates "field terms" (as used by the respondents) to "standard terms" (as defined in the search file).

Coding education at the division of Social and Spatial Statistics

This approach was chosen for the coding of education, where a search file is used containing a list of the most frequent regular educations along with their defining characteristics; a tiny extract is shown below to illustrate the concept:

Table 1: For each code a number of characteristics have been determined and these are stored in separate column in a search file. Each code corresponds with one record. With each column a specific question is associated.

Level	***Field***	***Teacher***	***University Degree***	***Teacher Type***
Medium level	English	No		
Higher level	English	No		
University	English literature	No	Master	
Higher level	Interpreter English	No		
Higher level	Translator English	No		
University	English	Yes	Master	First degree
Higher level	English	Yes		Second degree

To code education, the first question asked concerns the name of the education. When the respondent's answer has been entered (e.g. "higher level English") the search engine tries to locate matching records. This results in a preliminary selection of educations. This selection is usually too large to present to the respondent and therefore further questions about the education's defining characteristics are asked in order to limit their number. The possible answers for subsequent questions consist of the distinct items from column X corresponding with the question. If the set of selected records has finally become sufficiently small (less than 7 educations) then either one of two situations can occur: 1) only

one education is left; this education is presented to the respondent for confirmation, or 2) 2-6 educations are left; the interviewer presents these options to the respondent for final selection. Occasionally, more than one record left, but all questions have been asked. In this case, the code must be derived manually at Statistics Netherlands based on the answers and selections form the respondent.

Coding economic activity at the division of Business Statistics

However, for the coding of economic activity at the division of business statistics this approach was not sufficient. Apart from a much broader range of field terms, which can be solved by extending the synonym file, another problem needed to be addressed. In order to explain this problem we must first describe the search file in more detail:

Table 2: an excerpt of the search used to code economic activity.

(SBI) Code	***Description of the code***	***Process***	***Good***
X	Cultivation of agricultural product	Cultivation	agricultural product
…	…	Cultivation	…
X+N	Cultivation of flowers	Cultivation	flowers
Y	Processing of potatoes	Processing	potatoes
Z	seedpotatoa wholesale	wholesale	Seedpotatoa
Z+1	Potatoes for consumption wholesale	Wholesale	Potatoes for consumption
…	…	Wholesale	…
Z+M	Wholesale of computers	Wholesale	Computers

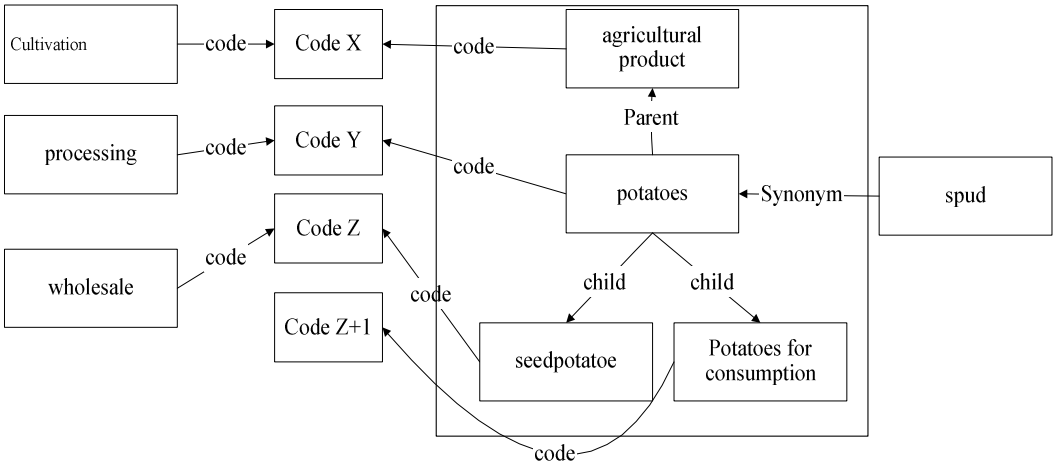
Based on the excerpt of the search file above we will describe the search result for 3 search strings as shown in table 3 in the first column. The second search works fine. The first search string specifies the good too narrow; as a result one only gets hits for the word *cultivation* (i.e. codes X….X+N). The same holds for search string 3.

Table 3: Three prototypical search strings to illustrate the necessity to introduce classifications in the search method.

No	***Search term***	***Comment***	***Hits without classification***	***Hits with classification***
1	Cultivation of potatoes	Just hits through *Cultivation*	N	1
2	Processing of potatoes	Both words match	1	1
3	Potatoes wholesale	Just hits through *wholesale*	M	2

In order to know the level of detail when specifying the good, one needs to know the code, but this is not known in advance, by definition. To solve this problem a classification has been added for goods and code descriptions. A very small extract of the classification relevant for the current problem is shown below inside the box as part of the semantic network.

Figure 1: This network is based on the three file types (search file, synonym file and classification file) which have been merged into one. The specific roles of the files have been translated into edge-types (code, parent, child and synomym).



Search in this network proceeds as follows:

1. For each word of the search string assign a score 1 to the corresponding node
2. Pass the activity on to the next nodes (following the arcs in the direction) with one restriction in the classification part: once a child edge has been passed, parent edges can no longer be traversed and vice versa.
3. Perform step 2 until "code nodes" are encountered.

This kind of search is called spreading activation (see Crestani 1997, Berger 2004). The spreading of activation in our implementation is governed by the formula:

$A_x = \sum A_i * EdgeStrength_{x,i}$

A_x : the activity of node ***x***

A_i : the activity of the nodes ***i*** *that have node* ***x*** *as input*

$EdgeStrength_{x,i}$: the "strength" of the edge from node ***x*** *to node* ***i***

In the current implementation $EdgeStrength_{x,i} = 1$.

All three search strings in table 3 contain "potatoes". Due to the classification this word is expanded to more general terms ("agricultural product") and to more narrow terms ("seedpatatoa" and "Potatoes for consumption'). As a result we already have an activity 1 for the codes X, Y Z and Z+1. If we search with the other term from the search string the relevant code(s) will receive an activity or score 2 and the others will remain at a score 1.

This approach has been used successfully for the initial search after an open text has been given by the respondent. If too many records remain for direct selection an approach similar to that for coding education is used, i.e. additional questions are asked to reduce the number of records.

The disadvantage of this method, compared with the previous two ones, is the considerable amount of work needed to construct the search file, the synonym file and the classification files. However, it does explicit our knowledge and understanding of the classification itself.

3. Computer assisted Coding by Interviewers (CACI)

3.1. Concept of computer assisted coding

At Statistics Netherlands we developed and implemented an interactive technique for the classification of economic activities, occupations, and educations, where the coding activities are mainly performed by interviewers during CAPI-interviews. In this process the interviewers collect open text answers from respondents and enter these text strings on their laptop computers. The laptop software then evaluates the entered text strings and generates a small list of appropriate codes. Next, these codes are presented to the respondent in the form of a short list of code descriptions from which the respondent can select a unique code description. Having done this a code has been established. The role of interviewer and laptop software is to guide the respondent to a correct answer. Open text answers for which no code can be found are referred to coding experts at the statistical office.
The CACI methods described in section 2 have been implemented as laptop software with search engines and search files. The implementation of these methods in questionnaires will be discussed in the next paragraph, where we will consider the mechanism of the search strategies in detail.

3.2. Coding economic activity for the labour survey

For the coding of economic activities of businesses two search files are used, based on methods from sections 2.1 and 2.2. One of these files is a registry: the General Registry of Businesses. It contains statistical information for most active businesses located in the Netherlands including their legal names, trade names and a code number representing the main economic activity. Because of the size of this registry only a small number of businesses are selected in the search file. The selection is based on business size (businesses with 100 or more employees are selected). The reason for this selection is that a large part of the employees work in a relatively small number of large businesses; businesses with 100 or more employees represent approximately only 1% of the population, however approximately 57% of all jobs are found within these businesses (Statistisch Jaarboek 2004, Statistics Netherlands). The second search file is based on results from previous surveys. It contains coded descriptions of economic activities as has been discussed in 2.1.

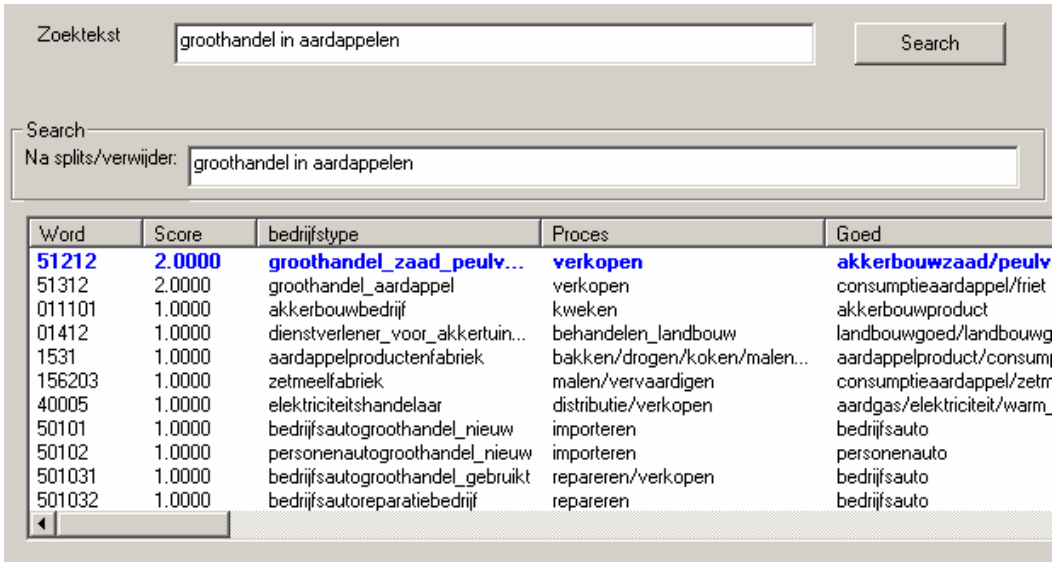
The first question in the Blaise questionnaire (concerning business variables) is about business size. The answer to this question determines which search file is going to be used: large businesses can probably easily be located in the registry data. Smaller businesses are not present in this file and in this case the search engine should use the search file with historical data. If a business has 100 or more employees the question ‘What is the name of the entire business?’ is presented to the respondent. The answer is entered and the search engine then tries to locate appropriate business names in the search file, using the Levenstein metric for calculating the ‘closeness’ of words. If more than one hit and less than 7 hits are found the selection is presented to the respondent in order to identify a single business name and establish a unique code. In case the search engine cannot find similar business names or if more than 7 business names are found the respondent is asked to describe the business activities. This is also the first question for businesses having fewer than 100 employees. When an open text answer has been entered the search engine tries to locate records with similar descriptions of business activities in the search file containing historical data (based on the method from 2.1). If there are fewer than 7 hits or if there are 6 hits which have sufficient weight the selection is presented to the respondent in order to select an appropriate economic activity. In case no hits or too many hits are found (or if no option is applicable) the search engine interface is switched off and the routing of the questionnaire returns to the Blaise-environment. Here a hierarchical type of

question is asked to determine economic activity with less detail (1 or 2 digits instead of 3 digits).

3.3. Coding economic activity at the division of economical statistics

The prototype for coding economic activity at the division of business statistics is currently implemented as a standalone desktop application. Once started, the application asks the user to describe the main economic activities of their business. If there is a single code that has the largest score, the description is considered to be coded.

Figure 2: Initial screen of the prototype to code economic activity. The search string is "groothandel in aardappelen" (= "Potatoes wholesale"). As described in section 2.3 this leads to 2 best candidates.

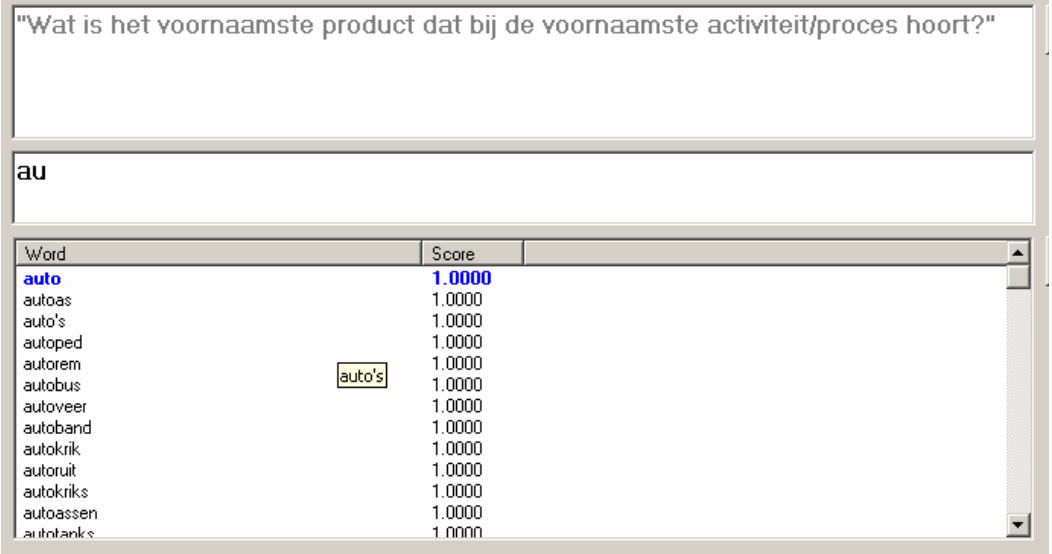


Word	Score	bedrijfstype	Proces	Goed
51212	**2.0000**	**groothandel_zaad_peulv...**	**verkopen**	**akkerbouwzaad/peulv**
51312	2.0000	groothandel_aardappel	verkopen	consumptieaardappel/friet
011101	1.0000	akkerbouwbedrijf	kweken	akkerbouwproduct
01412	1.0000	dienstverlener_voor_akkertuin...	behandelen_landbouw	landbouwgoed/landbouwg
1531	1.0000	aardappelproductenfabriek	bakken/drogen/koken/malen...	aardappelproduct/consum
156203	1.0000	zetmeelfabriek	malen/vervaardigen	consumptieaardappel/zetm
40005	1.0000	elektriciteitshandelaar	distributie/verkopen	aardgas/elektriciteit/warm_
50101	1.0000	bedrijfsautogroothandel_nieuw	importeren	bedrijfsauto
50102	1.0000	personenautogroothandel_nieuw	importeren	personenauto
501031	1.0000	bedrijfsautogroothandel_gebruikt	repareren/verkopen	bedrijfsauto
501032	1.0000	bedrijfsautoreparatiebedrijf	repareren	bedrijfsauto

If not, more questions will be asked by the application to try and reduce the set of possible codes as shown in the screenshot in figure 3.

If, after asking some questions, the set of possible codes is less than 7, the respondents asked to choose from a fixed list of descriptions.

Figure 3: Screenshot from the prototype. At the top the question ("what is the main good used for the main activity ?"). Below that the answer can be typed; while entering letters, the most fitting distinct items from the column goed (=good) will be displayed below.



3.4. Technical implementation

The coding system as implemented at the Division of Social and Spatial Statistics consists of the following components:

- The main search routines have been implemented in C++ because of speed
- Some additional logic and the graphical user-interface have been implemented in VB
- Finally, the above two components are embedded in a Blaise questionnaire, so that interviewers can start the search engine at specific questions using a key stroke.

For the Division of Business Statistics a prototype has been implemented based on the same C++ code in combination with some new VB code to render a desktop application.

4. Results

An important part of the development of automated coding and the implementation of computer assisted coding is the evaluation of their performance. There are two issues to address here. First, there is the question of effectiveness and reliability, i.e. is the percentage of cases that are coded sufficiently high and is the reliability and level of detail according to specifications? The second issue is about the requirements for the new technique to work well in a production environment. This covers matters like: can interviewers handle the new type of questionnaire, what additional training is required, what other software applications are needed for maintenance and monitoring, etc. In this paper we will only address the first issue, although the second one is very important once one decides to move beyond the development phase.

Definitions

Evaluation of the performance of CACI and text-classifiers will be based on a number of parameters: coverage and error rate. The definition of these parameters for CACI is as follows:

Coverage = [# cases that are coded / total # cases] x 100%.

Error rate = [# cases that are not coded correctly / # coded cases] x 100%.

where a 'case' is one occupation, one business, or one education for which a code is to be established (# means: 'number of').

The qualification of a case as coded correctly or not in CACI is made by comparing results with the codes from coding experts. A code is classified as correct if the CACI code is equivalent to the result established by the coding expert or if this latter code was present among the selection made by the search engine. In other cases the CACI code is classified as false. For text-classifiers the error rate is calculated differently. It is determined from the accuracy of the meta-classifier:

Accuracy = [# cases correctly labelled 'good' / total # cases labelled 'good'] x 100%

The error rate of the meta-classifier is then simply: 100% - *Accuracy*.

Performance of text-classifiers

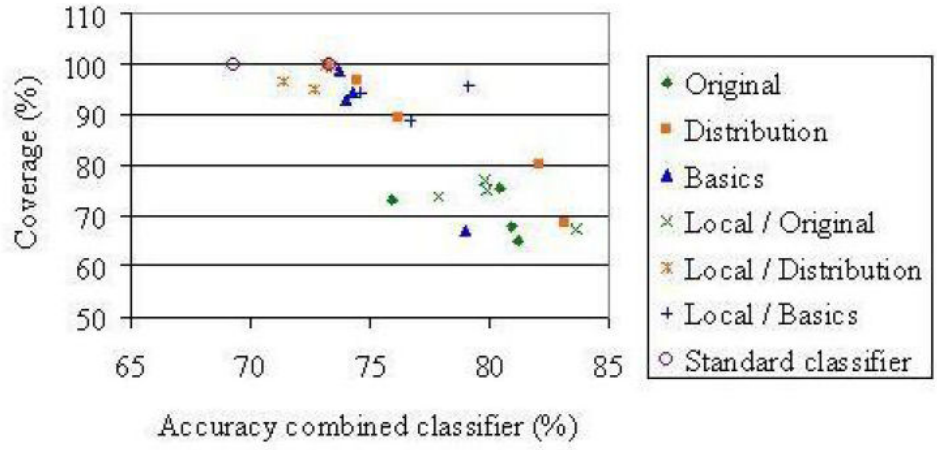
The meta-classifier approach from section 2.1 was applied two datasets from Statistics Netherlands, occupation (6 digits) and the level of education (1 digit). The training and test data were obtained from several Labour Force Survey files, containing 182038 cases. Optimal settings for different parameters have been determined, e.g. text representation and classification algorithms. The best results for the two datasets are given in table 1.

Table 4: Accuracy and Coverage of the meta-classifier approach when coding occupation (SBC codes) or education level. (NB: Naïve Bayesian; NN: Nearest neighbours)

Dataset	Base classifier	Meta classifier	Metadata	Class level	Accuracy	Coverage
Profession	NN	NB	Distribution	global	67.7%	16.5%
	NB	NB	Original	global	62.1%	44.6%
	NB	NN	Original	global	63.0%	39.9%
Education (First digit)	NN	NB	Original	local	83.6%	67.4%
	NB	NN	Distribution	global	83.1%	68.6%

As apparent from figure 4 a choice must be made between coverage and accuracy as they are inversely related:

Figure 4: Accuracy and Coverage of the meta-classifier approach when coding education level. The descriptions of the methods mentioned can be found in Kaptein 2005.



One the conclusions from her thesis confirmed the starting point of the work described in this paper, namely that the largest problem encountered with automatic coding is the (lack of) quality of the input data. The input contained much "noise", e.g. misspellings and descriptions that were either too vague or ambiguous.

4.3. Performance of CACI

In this paragraph and in paragraph 4.4 we present results on coverage and error rates for the coding techniques discussed previously in this paper. The performance of the CACI coding technique has been measured in two phases. The first phase consisted of a field experiment (in September 2003) with approximately 700 respondents using a Labour Force Survey questionnaire with CACI modules for coding economic activity, occupations and education, the second phase included an analysis of the production version of the Labour Force Survey in January 2004 with the same CACI-modules as in the first phase test. In this case a regular survey sample was used with 11607 respondents. The first phase was intended to establish preliminary results on coverage and reliability of the CACI coding technique before release of the production version in 2004. In the second phase the coverage was measured anew in regular production. Also, an additional check of coding reliability has been realized for a limited number of respondents. At the time of writing this article results on error rate were available for the coding of economic activity (based on a sub sample of 500 respondents). In table 5 an overview of these results is given.

Table 5: results from the coding in the field for 2 periods.$Code_{field}$ coded correctly if same as code $Code_{manual}$ from coding experts, or $Code_{manual}$ was present among the selection made by the search engine, but not selected by the respondent.

Attribute	First phase: September 2003		Second phase: January 2004	
		%		%
Number of respondents	699	100	11501	100
respondents age: 14 years or older	564	81	9128	79
respondents with a job	364	52	5726	50
Number of businesses to be coded	383	100	5978	100
coded with search engine	315	82	4653	78
coded with hierarchical question	49	13	947	16
not coded	19	5	378	6
Number of occupations to be coded	364	100	5726	100
coded with search engine	289	79	4299	75
not coded	75	21	1427	25
Number of (current) educations to be coded	107	100	1650	100
coded with search engine	88	82	1235	75
not coded	19	18	415	25
Number of completed educations to be coded	1199	100	19127	100
coded with search engine	989	83	15569	81
not coded	210	17	3558	19
Error rate economic activity of businesses		<12		7
Error rate occupations		<10		-
Error rate educations		<13		-

4.4. Performance of the semantic network

Although not used in the field yet, there are some early results for the coding of economic activity at the division of business statistics (based on section 2.3):

Number of codes found:

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	0	26	1,9	1,9	1,9
	1	1174	85,1	85,1	87,0
	>1	180	13,0	13,0	100,0
	Total	1380	100,0	100,0	

The results can be divided into 3 categories:

0 hits: the word combination in the search string is not found in any of the descriptions of economic activity in the search file.

1 hit: either exactly one hit of there is just one hit with the highest score

>1 hits: there is more than one record with the highest score

The results for the complete coding process (including the additional questions) are not available yet at the time of writing.

4.5. Length of interview

In addition to coverage and accuracy, interview lengths have also been measured during the pilot at the division of social and spatial statistics. Interview lengths are an important factor to consider when one is interested in making a more cost-effective coding process using computer assisted coding in the field. If interview lengths increase then more interviewers are needed to carry out the same number of interviews. The gain in cost-efficiency due to a reduced number of manual coding activities may then be partly or completely cancelled due to increases in interview lengths. Therefore we also present results on interview lengths for the Labour Force survey in 2003 (with the old questionnaire) and in 2004 (with the new CACI questionnaire). When compared to the old situation the interview lengths of CACI questionnaires on economic activity of businesses, occupations and education change considerably, but not all in the same direction. The average interview length for businesses decreases whereas the average interview lengths for occupations and educations increase. This is illustrated in table 6 below.

Table 6: since the primary drive of the project was a reduction of fte's, too large an increase of interview time is unwanted. The average time per person for each of the 3 subjects education, occupation and business. After an initial increase, the total time for the 3 subjects returns the value for the old situation.

Questionnaire	**Interview length (old)** Labour force survey 2003	**Interview length CACI** Labour force survey January-March 2004		
		January	February	March
Business	72 sec	53 sec	48 sec	49 sec
Occupation	36 sec	49 sec	46 sec	47 sec
Education	177 sec	209 sec	196 sec	190 sec
Total	285 sec	311 sec	290 sec	286 sec

From the table it becomes clear that the interview lengths of the new CACI questionnaire decrease over time. In January 2004 the interview length for businesses, occupations, and educations combined was on average +26 seconds larger per interview when compared to the situation in 2003. In February and March of 2004 the combined interview length was only slightly larger than in 2003: on average respectively +5 seconds and +1 second per interview. This may indicate that interviewers get accustomed to the new questionnaire.

The change in interview lengths has a plausible explanation. In the old questionnaire information on business names, location and type of business activities were collected in order to establish the code for economic activity. A large part of this information, however, was redundant. In the CACI questionnaire this redundancy is removed and often a single question (concerning name or type of business) is sufficient for a successful classification. The situation is different in the case of occupations and educations: here the number of questions required to establish a code is not reduced and interview lengths actually increase. The reason for this to occur is that each question in the new questionnaire demands extra tasks to be performed. Apart from posing a question and entering an open text answer a code has to be selected from a listing produced by the search engine. For this selection process to work the interviewer must present these codes to the respondent and the respondent has to identify an appropriate option. The number of questions that include this search-and-select strategy is small in the case of occupations (approximately 1-2) and large for educations (on average >3 as all educations of the respondent are recorded). Therefore the increase in interview length is largest with educations.

5. Conclusions

Interactive coding during interviews has an important advantage over traditional approaches. If the information contained in the open text answers is not sufficient to select a unique code the interviewer can ask the respondent to limit the selection. In case the information is sufficient no extra questions need to be asked. This possibility of getting extra feedback from respondents where required is an improvement on techniques that do not use an interactive scheme.

Although a large part of the coding effort is transferred to the field interviewers, measurement of the timing of the field interviews shows that, after an initial rise in interview time, there appears to be no extra time required; in addition, interviewers appreciate the feedback given by the search engine.

The development of CACI requires programming expertise and knowledge of statistical processes. This experience is usually available at a statistical office. Development and implementation include the construction of a questionnaire, software tools, test procedures, and the writing of documentation. Also maintenance of software and search files for production is important. Examples are the updating of search files due to changes in classifications, the correction of errors in search engines, and transmitting new versions of search engines and search files to interviewers.

Based on our results we conclude that CACI is a promising technology for the classification of survey data. Most of the occupations, educations, and economic activities can be coded successfully giving results with sufficient reliability. In addition, it is cost efficient in operation and search files can be constructed and updated automatically using new data from coding experts. This makes it also a technology with low maintenance costs.

6. References

Berger H., Dittenbach M., Merkl D., *An Adaptive Information Retrieval System based on Associative Networks*, APCCM04

Conrad, F. (1997), "Using Expert Systems to Model and Improve Survey Classification Processes", *Survey Measurement and Process Quality*, John Wiley & sons, inc.

Crestani, F. (1997), *Application of spreading activation techniques in information retrieval,* Artificial Intelligence Review, 11(6):453–582

Kaptein R. (2005), *Meta-Classifier Approaches to Reliable Text Classification* , master's thesis, IKAT, Maastricht University.

Mitchell, T. (1997), *Machine Learning*, McGraw-Hill.

Smirnov, E.N., Sprinkhuizen-Kuyper, I.G., Wiesman, F.J., Donkers, J., Postma, E.O., and van den Herik, H.J. (2003), *Typing professions adequately*, IKAT, Maastricht University.

Statistics Netherlands (2004), *Statistisch Jaarboek 2004*, published by Statistics Netherlands.

MultiRelaX with BCP as a replacement for Cameleon

Marien Lina (Statistics Netherlands) & Richard Frey (Westat, USA)

1. Introduction

This paper examines whether the Blaise Component Pack (BCP) can be used as a replacement for Cameleon when generating Datamodel descriptions. Cameleon is available in the Blaise system to retrieve meta data from the Blaise meta file and translate them into meta-descriptions and formats for ASCII and ASCII-relational output. Cameleon allows you to control your output when writing set-up files for SAS, SPSS, oracle, paradox, and for many other software applications. Blaise Data can be exported according to these formats with Manipula.

The recently developed Blaise Component Pack provides an opportunity to address data and meta information together in one program. The BCP enables you to create applications that prepare meta descriptions for software packages and at the same time write data in the related format. This paper discusses the outcome of an experiment in which a BCP application developed in VB writes, like Cameleon, related meta information. The meta descriptions for this application were designed to generate a language neutral output. The next step of this experiment is to write data, like Manipula, in the specified format of the exported Datamodel descriptions. The application constructed in this experiment has been named: MULTI RELational eXport tool, in short: MultiRelaX.

2. Exporting Datamodel Descriptions with Multi RelaX

The application is designed to read the Blaise meta information and the Blaise data base to create ASCII Relational output files. The output files are based on the block structure of the Datamodel where the information and data for each block are written to separate output files. Any fields in an embedded block are treated as being part of the parent block, and embedded parent blocks are treated as being part of their parent block.

The application was developed in Visual Basic, using the Blaise Component Pack to read meta information from the Blaise Meta file, generate language neutral descriptions and export Blaise data based on the description.

Exporting data in other formats requires more flexible options for writing meta information suitable to read the data in other packages. Obviously there are many ways to export data, asking for different meta descriptions.

The first step in the process is the creation of a general ***Meta Table.*** Three table types are part of this meta table: the ***Field Definitions Table***, the ***Long String Table*** and the ***Type Definition Table***. These tables identify the elements of the descriptions depending on the selected options.

The second step in the process is the selection of the layout for the exported meta data. Default options of the application can be changed using: ***File Options***, ***Field Options***, ***Long String Options***, and ***Type Options***. These options are partially related to the treatment of data, such as data selection or the truncated values of certain string fields. It is also related to layout options for the external meta (such as used keywords for types). The selection of options for the meta fields affects the export format of the data. The general advice is to use the same options when exporting meta and later, when exporting data. The final step, ***Data Export,*** is still

to be developed. For the moment the application deals with creating the meta files. The idea is to add facilities to define more tables and to control the writing of the data in specific formats.

3. The Meta Table

Producing the meta table is the first step to facilitate the organization of the essential parts of meta information into a suitable format. This meta table can be used as a working table and contains parts of the BMI file that are required to produce meta descriptions for other packages. Using the Meta Table as a work table it is possible to change the layout without having to read the Blaise meta files over and over again.

3.1. The Field Definitions Table:

The Basic Field Table keeps track of relevant field definitions within a block, including the order in which field values are to be exported. For each field in the block a selected part of the meta data is listed. For the export table, required data elements can be included or excluded by selecting options in a "meta table options" dialog. Following is a complete list of the data in the work table:

- The Datamodel name
- The block name

and for every field:

- The field name
- The field type
- The type name
- The Array or Set indicator of the type
- The field position
- The field length
- The question text
- The question description
- The defined export value for DONTKNOW scores
- The defined export value for REFUSAL scores

The ***field name*** is copied from the Blaise meta. For fields in non-embedded blocks, both the Datamodel name and the hierarchical block name can optionally be included in the field name. For fields in embedded blocks, the field name includes all information of the hierarchical position of the field within the embedded block structure. It includes all parent block names until the first encountered non-embedded parent block.

The ***field type*** is one of the following basic Blaise types: integer, real, enumerated, string, open, datetype, timetype, classification or external. The displayed export keyword for the types can be changed in the meta options. The field length can also be displayed in the field type.

The ***type name*** is an identifier for the field type. By default this is the user defined type name in the Blaise meta. If the field does not have a named type then the field name will be used as type name. The ***field type*** and ***type name*** are single types. Array and Set indicators are stored separately.

The ***Array or Set indicators*** indicate the field is an Array or Set, and keep track of the number of array cells. If a field is an array or a set, this will be available in the field definitions table of the work table. This information can eventually be used,

for example when selecting an export format in ARRAY notation, which has not yet been implemented.

The ***field position*** and the ***field length*** hold information about the precise position of the exported field.

The ***question text*** and the ***description text*** are field texts as defined in the Blaise meta.

The ***defined export value*** for **"*dontknow*"** or **"*refusal*"** values in Blaise are the scores that automatically would be written when exporting Blaise data to ASCII format with Manipula.

3.2. Long String Table

Apart from the field definitions table, there is a ***Long String Table.*** The reason for using this sub tables is related to the length of strings. Strings and memo fields may be large and have the potential to cause the export data file to exceed the size limits of some database systems. To avoid this problem the user has been given the option to exclude memo fields and strings or specify a maximum field length if the field is to be included in the export file. The Long String Table lists all fields that are excluded from the export data caused by oversized string length. The Data Output table for these fields will include records with the field name, the form key and the string value.

3.3. Type Definition Table

The ***Type Definition Table*** holds the Type meta descriptions of all found types in the Datamodel. This table is a flat list of all separately defined field types in the entire data model. At this moment there is only one type definition table.

If a type is repeated for an array or a set, the type definitions only include the first element of the type. Including the others is not necessary as the other elements are of the same type. Nevertheless, the type definition table can include similar types, if they are defined more than once in the Blaise Datamodel.

By default the type meta descriptions take the name of the type defined in a TYPE section. The TYPE section can be on the main level in the Datamodel hierarchy, but can also be defined inside a block. If the type is defined in a block, then the type name of the parent block (a block type) is included in the type name. These type names are copied from the Blaise meta descriptions.

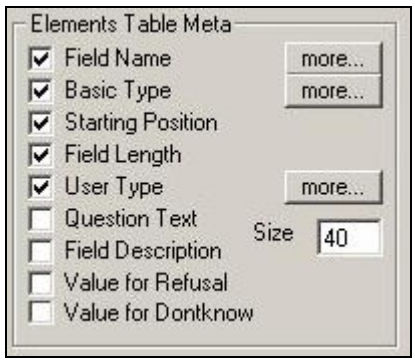
In Blaise you can define types directly after the field name and not in a TYPE, BLOCK or TABLE section. These types are not named in Blaise meta. For export meta it may be convenient, in some cases, to give these types a name. Therefore, the decision was to use the field name as the type identifier when there is no user defined type name. However, in Blaise, fields in different blocks may have the same name but a different type definition. Therefore, the preceding parent block name is included in the field name. This ensures that all types in the *Type Definitions Table* have a unique name.

4. Options

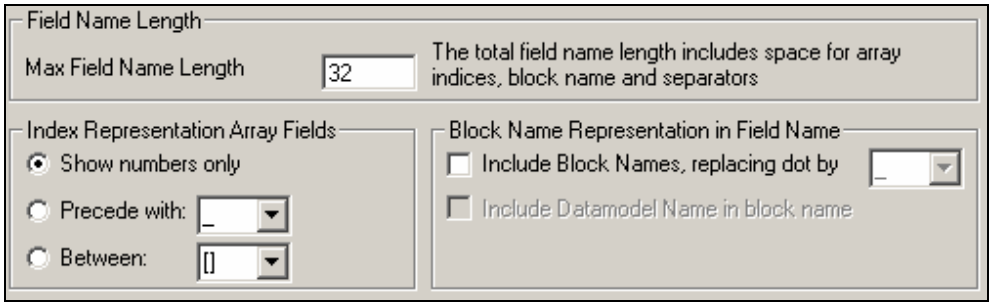
The application offers additional options for the formatting of the Meta description output and the substitution of some data attributes. Note that all the displayed dialogs presented in this paper are subject to change.

4.1. Field Options

The field options are the elements to be included in the field definition table. One can select whether to include the described options of the meta table in the field definitions table, for example, including the field name, basic type and the question text. The ”Size” box enables you to truncate the question texts and descriptions in your field definition table, in this case to 40 positions. All options can be included or excluded individually. In the current prototype this can be done in the following dialog:

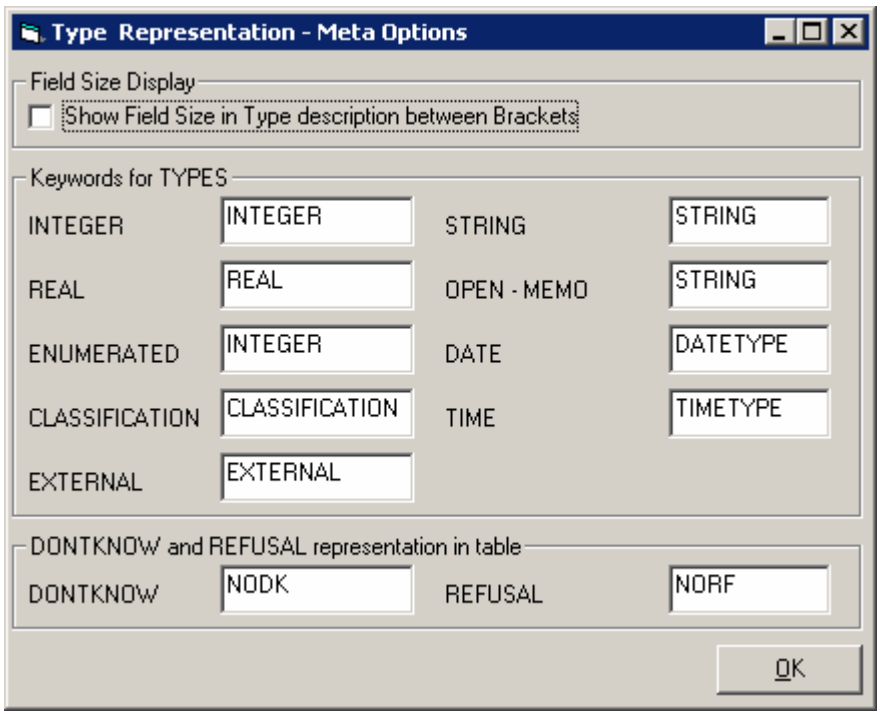


For some elements there are “more…” buttons. If you click the “more…” button next to the Field Name then the following items can be specified:



Here you can specify additional characteristics for displaying the field name, whether a block name should be included in the field name, the maximum field name length and how to array numbers should be treated and displayed.

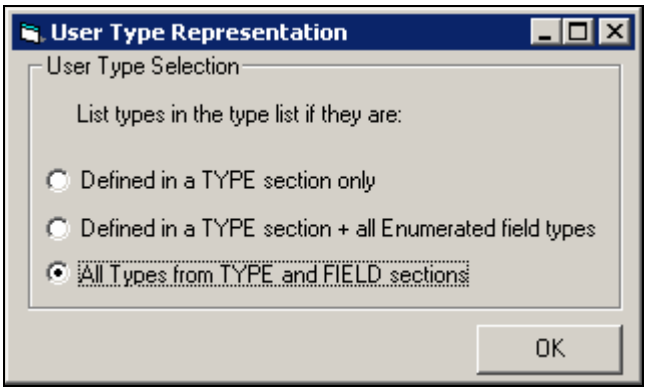
For the basic type there is also a “more…” options button. After selecting this button a dialog appears in which you can define keywords for the field definition table. The words will also be used for the type definition table. This is to meet requirements for data descriptions in different languages, for example, you may want to have a data description in which the Blaise type INTEGER is represented as LONG, and STRING as TEXT.



In the above dialoge you can specify the keywords for type specifications to be used in the field definition table. The default type names are displayed in the dialog and for each type you can change the type name you want to use in your tables. The dialog enables you to define other keywords for a type. For example, you could display the word FLOAT in stead of REAL in the definition table. These keywords will also be used for the type table.

Another option of this dialog is the specification to display the size of the field in the type. For example, if you have a string variable of 20 characters, you can select whether you want to see STRING (the default) or STRING[20] in the table.

The third “more…” options dialog is related to the user types.



By default a type list is made for all types that are defined in the data model. For a large data model one may include the types declared in a type section, striking out all types declared in the field section. In that case you will miss value labels of enumerated types declared in a field section. Therefore it has been made possible to select the user defined types and add enumerated types.

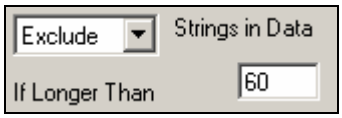
4.2. Additional options for the field definition table.

In the field descriptions section of the ***file options*** dialog you can select a separator and delimiter for the field definitions table. You can also indicate whether to print table headers and whether to truncate question texts and descriptions.

A header can be included for every block in the field definition table. The following dialog gives you also the possiility to change the default separators and string delimitors in the meta table.

4.3. Long String Options.

The main choice here is whether a string that exceeds a certain length should be excluded from the export data. The user can specify a maximum string length for strings to be excluded from the export data. The string will be written to a separate output table.



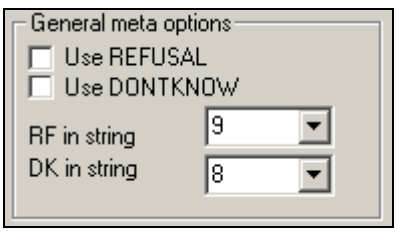
The meta definitions for the long strings excluded from the field definitions for the export tables is just a list of fields that are to be excluded. The options for the elements displayed for these fields is not very relevant as the required meta information can be included in the long string data.

4.4. Type Options.

As part of the field definition, this options also affect the representation of types in the type definition tables. This includes the choice of keywords for the basic types. It also includes the type selection of user types. In addition, it is possible to change the used separator and delimiter in the type table. This is done in the illustrated entry fields:

4.5. *Dontknow* and *Refusal* recognition in the export.

Other options in the main dialog enable the user to ignore dontknow and refusal scores, or to include them in the export file. This can be done in this dialog.



If the values are exported, these standard values may be taken. For numerical variables the value of a dontknow and refusal is numerical. For strings you may wish to specify specific characters as a dontknow value or refusal value in exported data, For example, one could change “9” into “#”, and that character could be exported if a string field has a Refusal score.

4.6. File Options.

One part of the *file options* enables the user to identify the locations of the input files (BMI and BDB) and the output folders for the export meta and data files.

After pressing on of the "…" button a dialog appears for specifying input files and output folders

5. Export.

5.1. Field definition output.

Once the options are selected and a BMI file has been selected, you can preview the generated meta tables. All options can be changed and previewed before exporting. A part of such a field definitions preview could look like this:

```
HaveNanny;INTEGER; 512;1;typYesNo;
HaveJob;INTEGER; 573;1;typYesNo;
PreviousJob;INTEGER; 574;1;typYesNo;
ThankYou;STRING; 575;1;ThankYou;
Nanny_NanName;STRING; 576;15;blkNanny_NanName;
Nanny_NanCountry;STRING; 591;15;blkNanny_NanCountry;
Paparazzi;INTEGER; 2;1;blkJob.zp;
CoName;STRING; 3;20;blkJob_CoName;
StartJob;DATETYPE; 23;8;blkJob_StartJob;
Region;INTEGER; 31;1;typRegion;
Paparazzi;INTEGER; 2;1;blkJob.zp;
CoName;STRING; 3;20;blkJob_CoName;
StartJob;DATETYPE; 23;8;blkJob_StartJob;
Region;INTEGER; 31;1;typRegion;
```

In this case the table displays a field name, field type, start position, the field size and a type name. After changing some options, a preview might be changed completely. One could instruct to make a variable lists (including name, type and size indicators) first. Then one could define another table: a list of variable names and labels (questionnaire text). For example, the following table, using the same fields as the list above, contains a very short field definition table with only field name and field type including field length:

```
HaveNanny;INT[1];
HaveJob;INT[1];
PreviousJob;INT[1];
ThankYou;TEXT[1];
Nanny_NanName;TEXT[15];
Nanny_NanCountry;TEXT[15];
Paparazzi;INT[1];
CoName;TEXT[20];
StartJob;DATETYPE[8];
Region;INT[1];
Paparazzi;INT[1];
CoName;TEXT[20];
StartJob;DATETYPE[8];
Region;INT[1];
```

If the result is what you want you can save the table in a file. After that you can create a second table with field name and field text only. By changing some options you would have the following table:

```
HaveNanny;"Do you have a nanny?";
HaveJob;"Do you have a job?";
PreviousJob;"Did you have any previous jobs?";
ThankYou;"Thank you for paticipating";
Nanny_NanName;"What is your nanny's name?";
Nanny_NanCountry;"What country is your nanny from?";
Paparazzi;"";
CoName;"What is the company name?";
StartJob;"When did you start working at ^CoName?";
Region;"Which region is ^CoName located?";
Paparazzi;"";
CoName;"What is the company name?";
StartJob;"When did you start working at ^CoName?";
Region;"Which region is ^CoName located?";
```

The separator ";" in the table can be changed into a space or any other character or string. These previews can be saved one by one.

5.2. Meta information for excluded long string fields

The long string data can be stored in a separate export file. For every excluded field the file will contain the record key, the block name, the field name and the string value. This is a point of concern when writing the data and it does not require a separate meta information table. In fact this meta would just be a list of the fields with long strings that are to be excluded form the export meta. This output can eventually be skipped as the required meta information will be included in the export data.

5.3. Type Definition output

Here is an example of a type definition table, the layout is still being developed. Each type is displayed over several lines. The included block name *blkNanny* is the name of the block in which the type *NanCountry* has been declared.

```
blkNanny_NanCountry
STRING[15];

blkJob.zp
 1;Cocina;"";
 2;Mamino;"";
 3;Paponi;"";

blkJob_CoName
STRING[20];

blkJob_StartJob
DATETYPE;
 99999998;refusal
 99999999;dontknow
```

For the type definition table, different layouts will be available, for example, using different separators, and possibly options to include or exclude value labels and descriptions. Obviously, useful information can be read from the Blaise Meta, and it would be a small step to adapt the layout to produce value labels and type declarations for SAS and SPSS.

Also here the idea is to preview the tables until the desired format is created. Then the table can be exported and the status of the selected options can be saved in a table model. At the moment this paper has been written, the prototype is still being developed.

5.4. Exporting Data with Multi RelaX

The current prototype does not support data export and only produces meta definitions. This is the next step furthering the development of the current prototype.

6. Conclusion

The experiment showed that Meta information can be retrieved from Blaise Meta information files using the Blaise Component Pack, and it can be organized like Cameleon to produce meta information. Similarly BCP can be used to read Blaise data and export it in the same format as analyzed from the BMI files. The application shows that it is technically possible to make this a Cameleon replacement application using Blaise Component Pack.

Session 4.1: Connecting

Experiences with Dynamic Link Libraries
Youhong Liu & Gina-Qian Cheung
(University of Michigan, USA)

Methods of Integrating External Software into Blaise Surveys
Lilia Filippenko, Joseph Nofziger, Mai Nguyen & Roger Osborn
(RTI International, USA)

Using the Blaise Component Pack from Within the .NET Framework to Develop Data Management Tools
Leonard Hart, Robert Thompson & John Mamer
(Mathematica Policy Research, USA)

Using Blaise to apply edits to data held in an Input Data Warehouse
Fred Wensing
(Australian Bureau of Statistics)

Experiences with Dynamic Link Libraries

Youhong Liu & Gina-Qian Cheung (University of Michigan, USA)

1. Introduction

Blaise supports the use of Dynamic Link Libraries (DLLs) to perform a process or action that is not currently available in Blaise itself. An example is a DLL used as an external subroutine that can be called by the Blaise Data Entry Program (DEP) or Manipula. The Blaise instrument passes information to the DLL and the DLL acts on the information and passes the modified or new information back to the instrument. Then the Blaise instrument stores or acts on the modified information. There are two types of alien or external references. Those that perform calculations are called alien procedures and those that ask questions are called alien routers.

Previously, DLLs could only be developed using Borland Delphi. The Delphi DLL itself would then communicate with DLLs developed using a different development environment, such as C++ or VB. From Blaise version 4.6 forward, alien procedures and alien routers in data models and Manipula setups also support the use of ActiveX. The external subroutines now may be implemented by using a COM object method.

This paper describes how the University of Michigan's Survey Research Center (SRC) used an alien router technique to meet a study requirement that could not be met directly with Blaise. After describing the problem addressed and the solution implemented, we will discuss the advantages and disadvantages of the approach taken.

2. Background

In the fall of 2005, the decision was made to move the instrument for a project from paper and pencil to Blaise, in order to ensure better data collection quality. The instrument is very straight forward and does not contain very complex logic. But a significant portion of the survey asks respondents about events that happened during their lifetime. There was a need to have a calendar with grids for interviewers to fill out for each event. This required development of special grids to simulate what is called an Event History Calendar (EHC). Blaise does not provide an easy way to implement the interface and functionality of an EHC. Thus, SRC decided to create the EHCs with the alien router technique. The Visual Basic (VB) ActiveX Com option was selected for creating this DLL. VB was chosen since it is a language very familiar to programmers and we already had some experience working with VB and the Blaise Component Package (BCP).

3. How It Works

The questionnaire is designed to ask filter questions that pass control to the external subroutine if the response is affirmative. For example, if the respondent answers "Yes" to the question "Did you ever live on a property where pets frequently went in and out of your home?" (Figure 1), then the alien router EHC window would open (Figure 2). If the response were "No," Blaise would continue to the next field.

Figure 1: Filter Question C1 Used to Trigger Alien Router Call to EHC

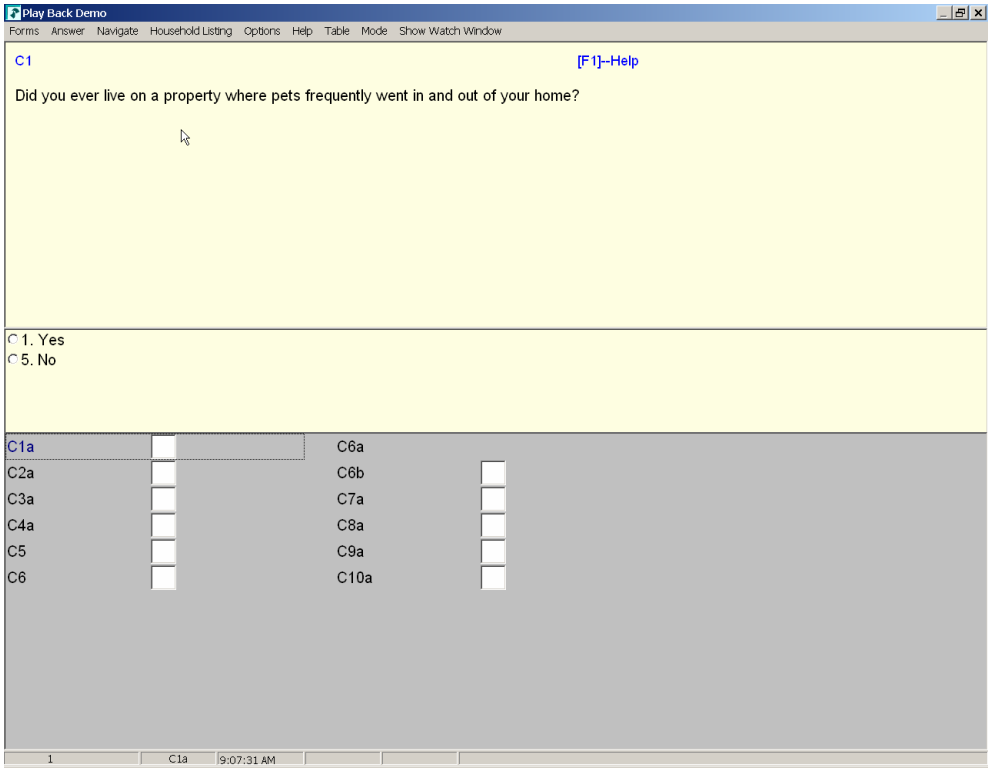
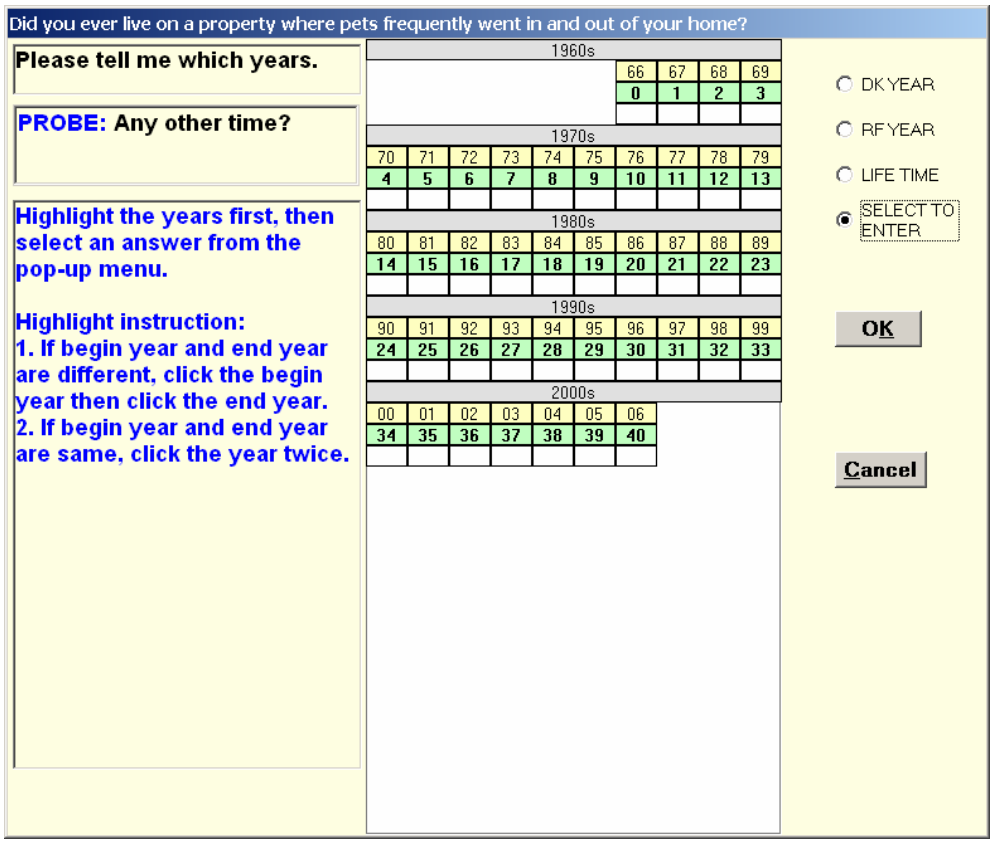


Figure 2: EHC Displayed by Alien Router Call after Filter Question C1

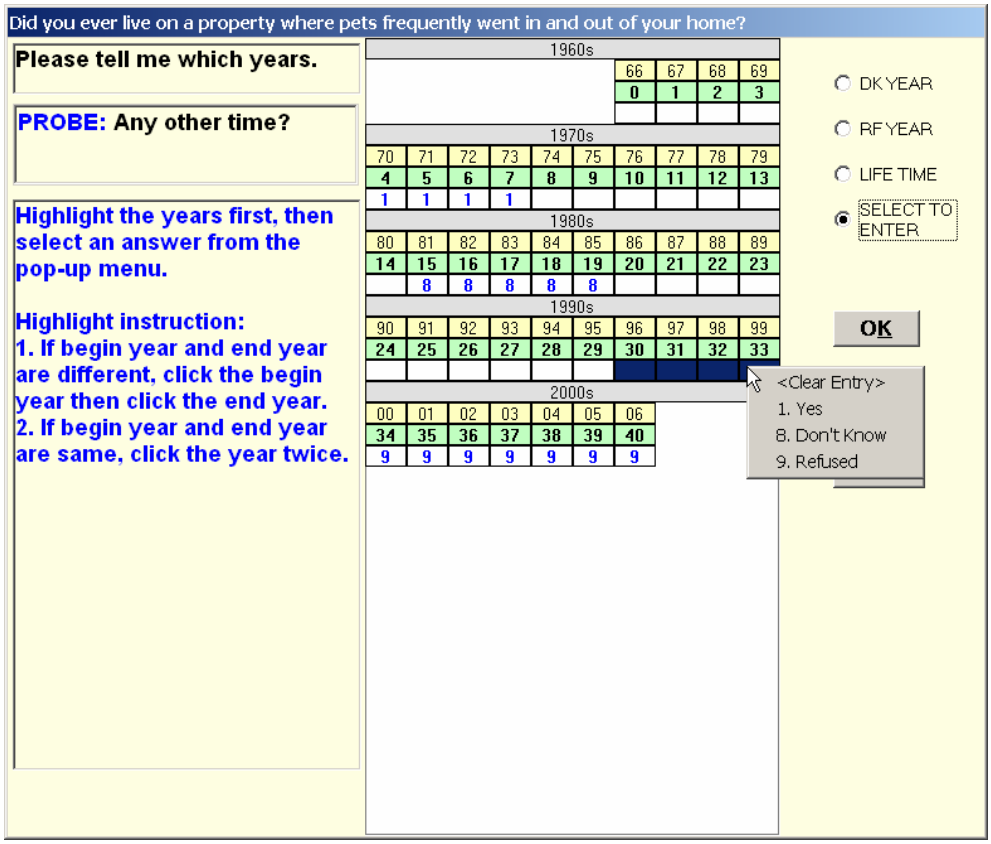


This is a Visual Basic form implemented in the DLL called by Blaise Alien router. The calendar's grids are displayed based on the respondent's birth year. The yellow grid indicates the year, the green grid represents the respondent's age, and the gray bar identifies the decade. All are for display only, so the Interviewer can navigate easily. The white grid is the only one used for data entry. The color grids

are VB label controls. All white grids are cleared when the calendar is accessed the first time. In the background, the DLL will read all information passed from the Blaise database, and save it for later use.

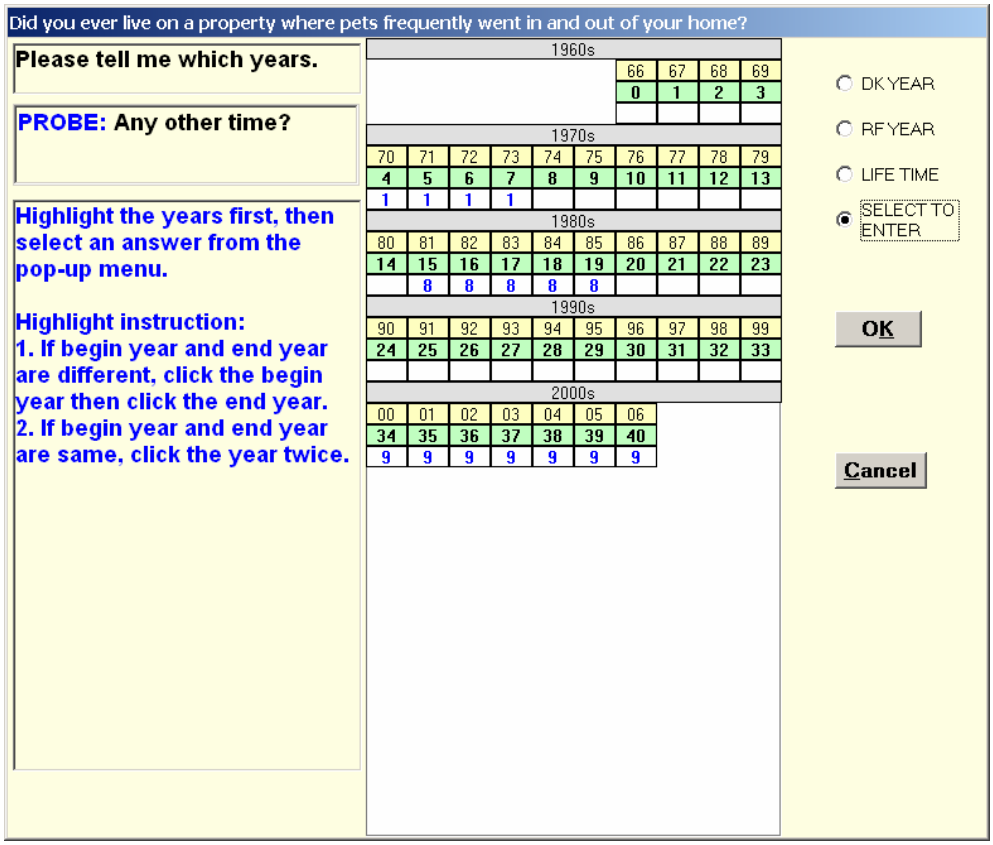
Normally, an Interviewer will highlight a few input fields and choose an option from the pop-up menu (Figure 3). If an Interviewer selects "DK Year", "RF Year," or "LIFE TIME", the response grids will be filled as 8, 9, or 1 respectively. After selections are completed, and "OK" is clicked, the DLL will send data to the Blaise database. Then the calendar is closed and the control is passed back to Blaise.

Figure 3: EHC Highlighted Period for Which Interviewer Can Record That Event Occurred



If the same calendar is accessed later, the DLL will read the Blaise database and fill the appropriate response grids in the form load event (Figure 4). The Interviewer can start to modify the data on the calendar and save it back to the Blaise database.

Figure 4: Prior Responses to Question C1 EHC Displayed Upon Reentry



4. Blaise Alien Router Block

Following is the source code for the Blaise alien router block that creates an EHC calendar grid, in this case for Question B1a, about ever living in households where members usually removed their shoes.

```
BLOCK RouterB1
      AUXFIELDS
            B1
            "BSec_B.B1.B1b~BSec_B.B1.B1_DK_Year~BSec_B.B1.B1_
            LifeTime~BSec_B.B1.B1_RF_Year" /
            {This is the field text of B1 which is used to pass field
            name list to the DLL, so DLL knows which field values need to
            be updated. The reason we want to pass the field list to the
            DLL is that we can use one calendar for multiple events.}
            "Did you ever live on a property where most
            household members usually removed their shoes
            before coming into the house?: STRING
            {This is the field description. It is used to be populated as
            EHC's title.}
      FIELDS
            B1a (B1a) "Did you ever live on a property where
            most household members usually removed their
            shoes before coming into the house?" : TYesNo

            ROUTER Calendar ALIEN('Dioxin.clsDioxin',
            'Calendar')
            {Calling the DLL from here. The DLL name is Dioxin, the class
            name is clsDioxin, and the sub name is Calendar. B1a is the
            field to access the alien router, and the EHC will be opened
            up if "Yes is answered.}
            B1b (B1b) "Please tell me which years?" :
            ARRAY[1..111] OF 0..1
```

```
            {First parameter in B1, corresponding to the white grids in
            the EHC, e.g. B1b[1]=>1900, B1b[2]=>1901… B1b[111]=>2010}
            B1_DK_Year (B1_DK_Year) : (Yes, No)
            {Second parameter in B1, DK option on the EHC}
            B1_LifeTime (B1_LifeTime ) : (Yes, No)
            {Third parameter in B1, LifeTime Option on the EHC}
            B1_RF_Year (B1_RF_Year ) : (Yes, No)
            {Fourth parameter in B1, RF option on the EHC}
      LOCALS
            I : INTEGER
      RULES
            B1a
            IF B1a <> Yes THEN
            {Empty all values; we need to do this here to clean up if the
            root question is not "Yes"}
                  FOR I := 1 TO 111 DO
                        B1b [i]:= EMPTY
                  ENDDO
                  B1_DK_Year := EMPTY
                  B1_LifeTime := EMPTY
                  B1_RF_Year := EMPTY
            ELSE
            {Values updated by the DLL that need explicit keep.}
                  FOR I := 1 TO 111 DO
                        B1b[i].KEEP
                  ENDDO
                  B1_DK_Year.KEEP
                  B1_RF_Year.KEEP
                  B1_LifeTime.KEEP
            ENDIF
ENDBLOCK
```

5. VB ActiveX DLL

There are many functions in the VB ActiveX DLL, and we show two of them below. The first is the function in the clsDioxin class. It is the connection point between Blaise and the DLL. The database or DB parameter gives access to the data and metadata that are currently present in the Data Entry Program (DEP). db.Field.([FieldName]) can be used to access any field in the database. DS is the DepState object that gives access to the state of the Data Entry Program. For example, one can use the DepState object to determine the name of the currently active Field in the Data Entry Program.

```
Public Sub Calendar(db As BlAPI4A2.Database, ds As
BlAPI4A2.DepState)
Set dbp = db
Set dsp = ds
Set Layout =
dbp.Screens.LayoutSetCollection(dsp.LayoutSetIndex)
Set Parallel = Layout.ParallelCollection(dsp.ParallelIndex)
Set Page = Parallel.StoredPageCollection(dsp.StoredPageIndex)
Set Quest = Page.QuestionCollection(dsp.QuestionIndex)
Dim PassedNamestr() As String
PassedNamestr =
Split(Quest.Field.Parent.Fields(blfkAuxField).Item(1).Questio
nText(1, False), "~")
      {B1's question text contains the list of fields that will be modified
      by the DLL}
DKYearName = ""
AllLifeName = ""
RFYearName = ""
If UBound(PassedNamestr) >= 0 Then
  RootName = PassedNamestr(0)
```

```
End If
If UBound(PassedNamestr) >= 1 Then
        DKYearName = PassedNamestr(1)
End If
If UBound(PassedNamestr) >= 2 Then
        AllLifeName = PassedNamestr(2)
End If
If UBound(PassedNamestr) >= 3 Then
     RFYearName = PassedNamestr(3)
End If
formCaption                                                    =
Quest.Field.Parent.Fields(blfkAuxField).Item(1).DescriptionTe
xt(1, False)
{B1's description contains the EHC's title text}
YearBorn = dbp.Field("YearBorn").Value
{"YearBorn" is in the datamodel; the EHC grids are displayed according to
it's value}
If ds.AlienRouterStatus = blrsPostEdit Then
      If Quest.Field.Value = 1 Then
            Form1.Show vbModal
            {Form1 is the EHC form that only opens the EHC, when question
            value is "Yes"}
      End If
End If
If ds.AlienRouterStatus = blrsEdit Or _
      ds.AlienRouterStatus = blrsPreEdit Then
      ds.AlienRouterAction = blraEditQuestion
      {This is necessary here, so the alien router questions can take
      input}
End If
End Sub
```

The next subroutine is behind the "OK" button on the EHC form. It updates the data in the database according to the entries on the form.

```
Private Sub cmdOK_Click()
Dim i  As Integer
Dim FieldName As String
Dim answer As Integer
Dim msgstr As String
If Entry_check(msgstr) = False Then
      answer = MsgBox(msgstr & "Do you like to proceed?",
      vbYesNo + vbExclamation)
      {Make sure at least one grid has value}
End If
If answer = vbNo Then
     Exit Sub
End If
setValueEmpty
{Initialize}
If opt1(0).Value = True Then
      dbp.Field(DKYearName).Text = "1"
      {Update DKYear field in the BDB}
ElseIf opt1(3).Value = True Then
      dbp.Field(RFYearName).Text = "1"
      {Update RFYear field in the BDB}
ElseIf opt1(1).Value = True Then
       dbp.Field(AllLifeName).Text = "1"
      {Update AllLife field in the BDB}
      {OK if AllLife EMPTY then will not come to this path}
End If
For i = YearBorn - 1900 + 1 To Year(Now()) - 1900 + 1
      {Update event array in the BDB according to the caption on the
      labels}
      DoEvents
      FieldName = RootName + "[" & i & "]"
```

```
      If lblResult(i).Caption <> "" Then
            If lblResult(i).Caption = "8" Then
                  dbp.Field(FieldName).Status = blfsDontKnow
            ElseIf lblResult(i).Caption = "9" Then
                  dbp.Field(FieldName).Status = blfsRefusal
                     Else
                  dbp.Field(FieldName).Text                  =
                  CStr(lblResult(i).Caption)
            End If
      End If
Next i
Unload Me
End Sub
```

6. Multiple Instances of the Same Calendar

In the study, there are over 30 EHCs. If the developer programs one subroutine for each EHC, it would require lots of programming time and testing, and the code would be difficult to maintain. The technique here is to use one routine for all instances of the EHC, and to use an auxfield to pass in the field names to the DLL. This way the DLL knows which fields to update.

The DLL can be programmed to be quite flexible. For example the DLL subroutine described here allows passing a special character string from Blaise to the EHS, so the subroutine knows how to process information differently based on filter question or location in instrument. For example, the lifetime options are not applicable to the events in Section D of the Dioxin study. As a result, an "Empty" was passed into the lifetime column. When the subroutine sees this, it makes the "LIFE TIME" option ratio button invisible and when it is time to update the database fields, it skips updating the LifeTime field.

7. Conclusion and Summary

DLLs are external routines that can accomplish many complex tasks in a flexible fashion. At the University of Michigan, there are several studies using this technique. Our experiences with the alien router calls to an external EHC have been very positive and interviewers have found the interface user-friendly. With careful programming between Blaise and the DLL, the quality of data collection has improved significantly over the prior paper-and-pencil EHCs.

However, the DLL could create problems in the application without careful implementation and testing. For example, if we use a DLL in our Blaise program without the DLL physically existing in user's machine or if it is not properly registered, the Data Entry Program will stop since it cannot find the DLL. Thorough testing of the whole system before production is essential.

Another disadvantage of using the DLL is that the data entered in the DLL program is not recorded in the Blaise Audit trail system. This makes it difficult for later data recovery or examination. We have thought that for future implementations the data should be written to an Access database or a text file, so the data are backed up in case the Blaise database becomes corrupt.

8. Acknowledgement

We would like to acknowledge following SRC programmers for their help on the VB DLL programming and input on this paper: Sujatha Venkatapuram, Peter Sparks and Jim Hagerman

9. References

Blaise OnLine Assistant (release 4.7.1000). (2005). Retrieved February 24, 2006 from World Wide Web:http://www.blaise.com/onlinehelp/

Methods of Integrating External Software into Blaise Surveys

Lilia Filippenko, Joseph Nofziger, Mai Nguyen & Roger Osborn (RTI International, USA)

1. Introduction

For surveys that require calls to external applications, Blaise provides support for various implementations. At RTI International we have studies that take the following approaches:

- Use Manipula setups to manage switching between a Blaise instrument and calls to external applications
- Use "Action" from a Blaise instrument and a BOI file to pass data between the Blaise database and MS Access databases
- Use an alien router from a Blaise instrument to call external applications and to collect data from them

Each method of integration has advantages and disadvantages. Although the alien routers may provide the most flexible solution, their development and testing could take more time than the first two approaches. In this paper, we will discuss the pros and cons of each implementation. We will also present our experience with several alien router implementations written in Visual Basic.Net and C# to meet the following aspects of integration:

- Seamless User Interface
- Ease of Programming (Blaise and Windows)

Finally, we present a case study of using an alien router to implement complex randomization of questions.

2. Invocation from Manipula setup

2.1. Case Management System Overview

At RTI International a Visual Basic Case Management System (CMS) program is used on laptops for many CAPI interviews. Along with providing user functions, the CMS manages invocation of the CAPI instrument. Various software packages or languages are used to develop instruments, including Blaise, ASP, CASES, and others. The type of software package is stored in configuration files and is used by the CMS to create and invoke an appropriate driver object to manage work with individual package.

Methods implemented by the driver and applied for every case on the laptop include:

- Import (load a case into the laptop database)
- Update events and status codes to the instrument database
- Conduct interview
- Export (transmit a case from the laptop)

For Blaise instruments, the driver invokes a single main Manipula setup, passing parameters to execute the requested method. The name of the data model and database is passed as a parameter. Our Manipula-driven architecture is illustrated in

Figure 1. The case ID and a study identifier are passed as parameters. The "Edit" function is used to run the Data Entry Program (DEP) for a case. This Manipula setup was developed in general way so that it can be prepared independently of the Blaise instrument with which it is used, and without changes to the CMS. This approach gives us the flexibility to make adjustments for different studies as needed. An example of this is the ability to call an external application while conducting the interview without returning to the CMS.

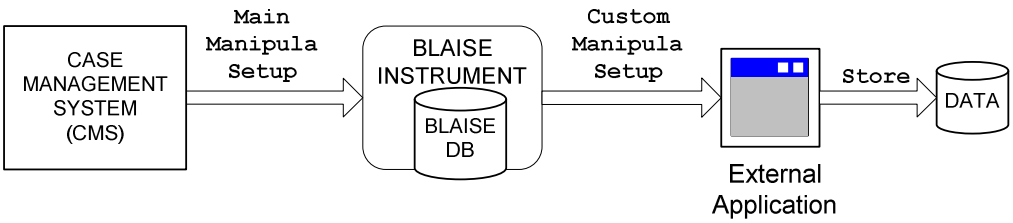


Figure 1 Manipula Setup Architecture

2.2. Example of Invoking External Applications from the Manipula Setup

RTI International conducted a study that required calls to three assessments in random order from the middle of the Blaise interview. Our approach was to associate a unique code ("sum_evt" in the snippets) with the completion of each assessment. The code can be used to decide which assessment to execute next. We added program code to the main Manipula setup to call a custom Manipula setup which randomizes the order of external applications, invokes them, and updates the code in the Blaise database.

First we define parameters to start DEP:

```
   'INVOKE':  { PARAMETER(1) - Case ID }
   strWorkF := PARAMETER(2)
   strInstName := PARAMETER(3)
   strProjectID := PARAMETER(5)
   strCaseId := ' /K' + PARAMETER(4)
```

Then we create a command string and start DEP for the Blaise instrument:

```
   strRun := strWorkF + '\' + strInstName + ' /G /X /C' +
     strInstName + '.diw @' + strInstName + '.bcf ' +
strCaseId
   Reslt := EDIT(strRun)
```

Next, we call the custom Manipula setup after the first part of the interview is completed:

```
   strTemp := GETVALUE(strInstName, strInstName,
      PARAMETER(4), 'main_case.sum_evt')

   IF (strTemp > '301') AND (strTemp < '306') THEN
      strRunTemp:='BlzUpdateINHL /W' + strWorkF + ' /P' +
         PARAMETER(4)+ ';Assessmt;' + strTemp + ' /Q'
      Reslt:= CALL(strRunTemp)
           {……}
   ENDIF
```

Finally, we check the code again in the Blaise database to determine if all external assessments were finished successfully, and start DEP again:

```
REPEAT
   strTemp := GETVALUE(strInstName, strInstName,
PARAMETER(4),
      'main_case.sum_evt')
   IF (strTemp >= '306') THEN
     strRun := strWorkF + '\' + strInstName + ' /G /X /C' +
     strInstName + '.diw @' + strInstName + '.bcf ' +
strCaseId
     Reslt := EDIT(strRun)
   ELSE
     { Assessments not finished }
     { Don't need to restart the instrument }
     EXITREPEAT
   ENDIF
UNTIL 1=1
```

3. Use of Action and BOI Files

The above method of invocation works well when Blaise and assessment data are stored separately. This is acceptable when data are to be analyzed separately. However, analysis can be facilitated by integrating the assessment data with the Blaise data at an earlier stage. Furthermore, data management is simpler when data are integrated into a single entity at the time of collection. With recent versions of Blaise including tools allowing us to pass data between Blaise databases and relational databases like MS Access and SQL Server, we have begun to make frequent use of BOI files to store all data in one place - the Blaise database - at the time of collection.

We chose to control invocation using an action because of its simplicity. Creating an action for a user defined type to start an executable application is quicker and less error prone than programming an alien router. This architecture is shown in Figure 2.

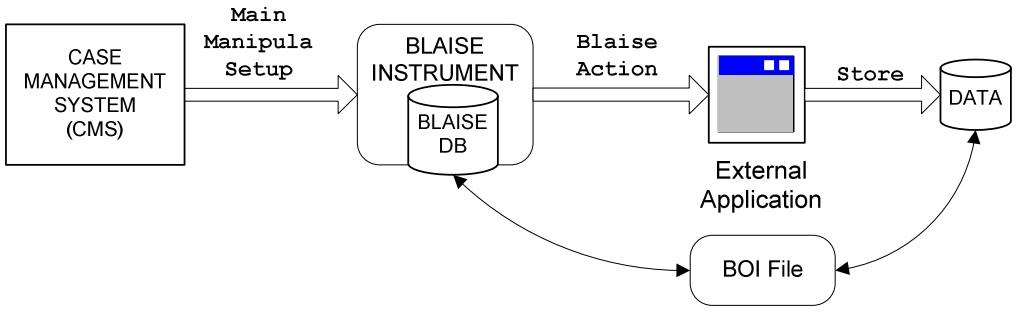


Figure 2 Action/BOI Architecture

3.1. Invoking the Assessment Using an Action

First we create a user defined type with the action “Start Executable”. We can then use this type to declare a field that is used as the entry point to invoke the assessment..

In case of a breakoff and subsequent re-entry into the instrument, we take steps to ensure that the incomplete assessments are invoked, and that already completed assessments are not improperly redone. A completion indicator field is used as a gate to determine whether to enter the assessment or to keep the existing data.

3.2. Data Transfer from MS Access to Blaise Database During Interview

Ideally, upon completion of an assessment, control passes back to Blaise. At that time the external BOI file is used to pull data from the assessment's Access database back to the Blaise database. This is accomplished by using the SEARCH method, with the case ID as a parameter, to locate the record in the MS Access database that is used by the external assessment to store the data. If the SEARCH is successful, then the READ method is used to assign values from the MS Access database to Blaise fields. The completion indicator field is set to value that will later be used to determine the state of the assessment.

3.3. Data Transfer Before Exporting Case from Laptop

A problem arises if the interviewer skips the question where the assessment would ordinarily be launched. We program a SIGNAL to remind the interviewer to conduct the assessment, but the SIGNAL may be suppressed. The interviewer may later realize the mistake and return to the appropriate question to launch the assessment. However, after proceeding past the action question the first time, Blaise would have tried to import the data using the .BOI file, even though the Access database would not contain any data for the case. The problem lies in the fact that the BOI file data import only occurs the first time. Despite the interviewer having now corrected the mistake and completed the assessment, the SEARCH method would not find data for the specified case.

A second problem situation occurs if the interviewer completes the assessment, but then, whether as a result of a mistake or system error, fails to return as planned to the Blaise instrument. If the system was shut down before Blaise regained control then there would be no opportunity to read data through the .BOI file. If the case happens to be exported at this point, for example due to transfer to another laptop, then without backup measures the assessment data would remain on the laptop where it was captured.

To be certain that data collected in the assessment will be in the Blaise database whether or not control is properly returned to Blaise, we created an external program that reads the assessment data and pushes it into the Blaise database immediately before transmitting the data back to our servers. A call to this program was added to the main Manipula setup under the "Export" option. It filters for only those cases for we expect to have data from the assessment in the Blaise database.

```
IF (Uppercase(strInstName)='CHILD') AND
   (strProjectID='9231')) THEN
   strTemp := GETVALUE(strInstName, strInstName,
      PARAMETER(4), 'WJComplete')
   IF strTemp <> '1' THEN
      strRun := strWorkF  + '\WJUpdateCase.exe' + ' ' +
         PARAMETER(4)
      Reslt := run(strRun, WAIT, HIDE)
   ENDIF
ENDIF
```

4. Alien Routers

Blaise alien routers enable external applications to be called from Blaise. Although Blaise expects an alien router to be a COM component, alien routers can be created in .NET utilizing COM interoperability. Using the Blaise Component Pack (BCP) alien routers can read and write variables in a Blaise database. This approach allows us to maintain all collected data in one place and therefore to simplify data management and later analysis.

One of our recent Blaise studies included seven alien routers and associated external applications. We used external applications for address verification, address mapping, random number generation, Woodcock-Johnson cognitive assessments, Event History Calendar (EHC) from the University of Michigan, and other purposes.

The primary goal was to provide a richer user data collection experience that minimized disruptions to the interview flow. Another goal was to simplify how the alien routers were used in Blaise code. In addition, we wanted to introduce some code reusability and common designs within the routers and applications. These goals were accomplished by incorporating the following:

- Set external application window to "Always on Top"
- Consistent look-and-feel amongst Blaise UI and external applications
- Use of complex data types in Blaise code
- Common architecture and code in routers

4.1. General Overview of Alien Router Architecture

Before examining the specifics of our implementation, a general overview of the alien router architecture is in order. The basic alien router architecture is shown in Figure 3. A Blaise program instantiates an alien router variable and calls it using the Blaise Router method. This loads a COM-based alien Router DLL and runs the method specified in the alien Router call. The router can use the Blaise Component Pack (BCP) API to read or write variables in the Blaise database. The Alien Router can also launch an external application using any of a number of mechanisms. One mechanism is to use the Start method of the .NET System.Diagnostics.Process class.

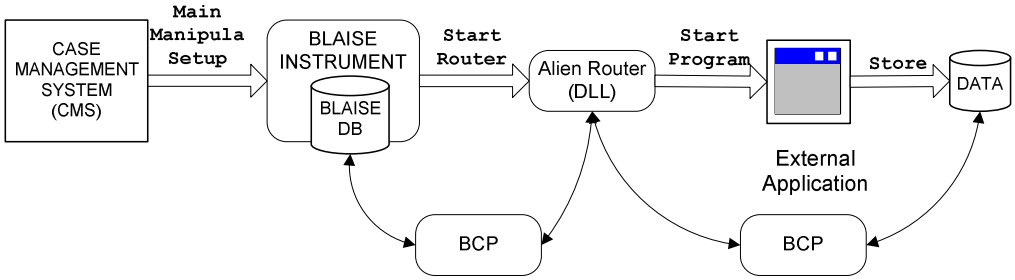


Figure 3 Basic Alien Router Architecture

4.2. External Application Window as Always on Top

The call from Blaise to the alien router and the call from the alien router to the external application are blocking, meaning the router does not continue to run until after the external application exits, and the Blaise instrument does not continue to run until after the alien router exits. While blocked, the Blaise instrument will not

respond to user input. Thus if the external application window is somehow moved behind the Blaise window, the Blaise instrument will appear to ”hang” or ”freeze”.

An easy way to prevent this is to set the external application windows as ”Always on Top”, by setting the TopMost property of the external application window to True. When the TopMost property is set to True, the window will always be in front of other windows. If the external application is not a .NET application, other methods can be used to make the window top-most.

4.3. Consistent Look-and-Feel

When an external application is invoked from a Blaise instrument via an alien router, it runs in its own window and has its own look-and-feel (visual representation) which may be very different from the Blaise instrument. This could create confusion for inexperienced field interviewers and respondents. To reduce confusion and improve the user experience, the visual properties of external applications can be set so that they appear to be an integrated part of the instrument. From the user’s perspective, the external application should appear as a Blaise pop-up dialog, not a separate application.

An example of a visually consistent external application is shown in Figure 4. In this case, the external application called from an alien router shows 14 houses and allows the respondent to select a race for each house in the neighborhood. The ”x” marks the resondent’s house. Also shown in Figure 4, the instructions to navigate the external application is clearly provided in the Blaise instrument question text. The user can move from one house to another using the standard ”TAB” key and label each house with different letter keys.

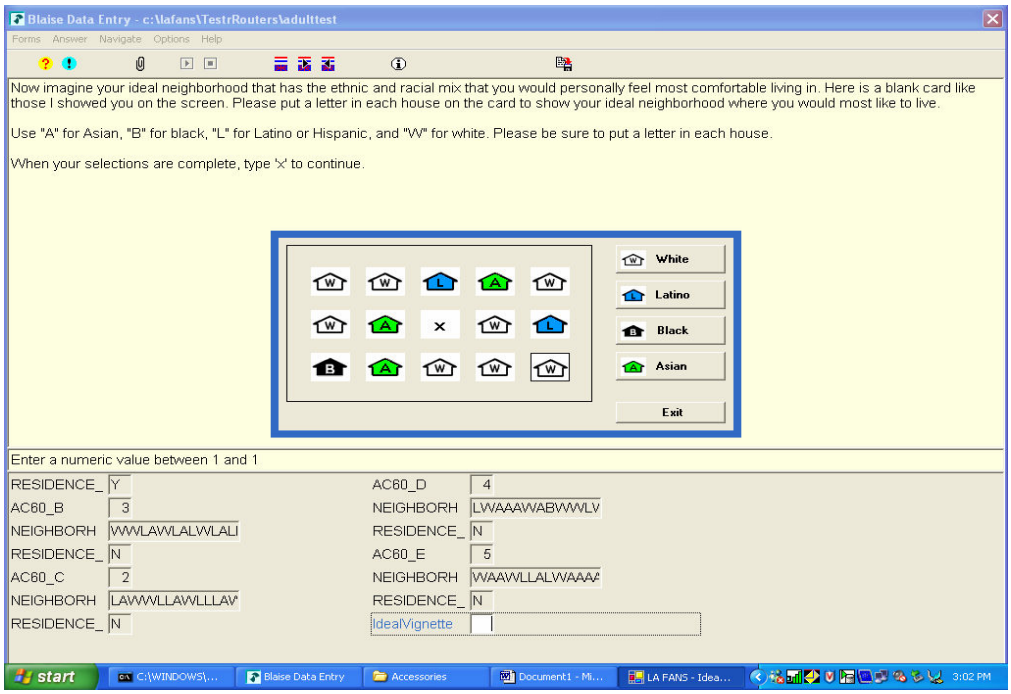


Figure 4 Visually-Consistent External Application

The external application’s window size and position can be specifically set using:

```
this.StartPosition = FormStartPosition.CenterScreen;
this.ClientSize = new System.Drawing.Size(480, 232);
```

These properties can also be set in the visual designer.

To prevent the user from prematurely closing the external application, the window's border can be set using:

```
this.FormBorderStyle = FormBorderStyle.None; //C#.NET
```

or

```
Me.FormBorderStyle = FormBorderStyle.None   'VB.NET
```

This property can also be set in the visual designer as shown in Figure 5.

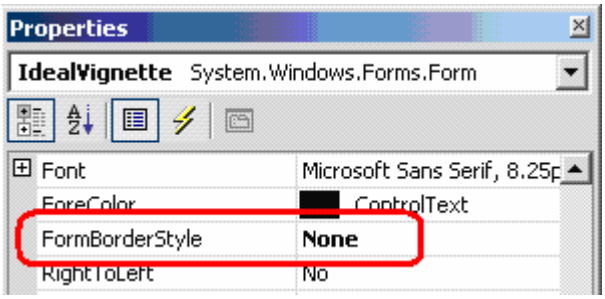


Figure 5 Setting FormBorderStyle Property

This level of seamless graphical presentation is easily achievable with external applications developed in-house. For third-party applications, for example the Event History Calendar (EHC), this level of integration is usually unachieveable or would take significantly more effort.

4.4. Complex Data Types in Blaise Code

In order to simplify and componentize the Blaise programming effort, we elected to use complex data types for router-calling variables in Blaise. This is best shown by example. The Blaise code is shown in Listing 1.

One of our external applications performed address verification. A user-supplied address is collected in Blaise and passed to an external application that uses a third party address verification API to ensure the supplied address is valid. In the Blaise instrument, we created a block called BAddressVerification. The BAddressVerification comprises other fields defined as blocks: AddressSearchElements, AddressOutputElements, and VerifyAddress.

AddressSearchElements contains the input address, which is collected by the Blaise instrument. VerifyAddress is essentially a wrapper variable for the call to the alien router. The results of the address verification by the external application are written to the sub-fields of AddressOutputElements.

When address verification is required in an instrument, the Blaise programmer simply creates a variable of type BAddressVerification.

Since the fields of the data type are well-defined, and the call to the alien router is within this block, the alien router has a relative reference to all the names of the sub-fields within the block and can easily address them.

Other benefits and implementation details of using such block-within-block complex data types are described in the next section.

4.5. Common Router Implementation

In order to increase code reuse and maintainability, we used a common implementation amongst our alien routers. While we did not go so far as to create a base class for our alien router implementation, such a class could have easily been developed.

Source code for a typical alien router is shown in Listing 2. This router is written in C#.NET. In order to use the BCP API, the Blaise BCP DLL is referenced and included:

```
using BlAPI4A2;
```

The entry point for all of our alien routers is the Run method, which we'll describe in detail.

The first thing the Run method does is store a class-level reference to the database and DEP state variables that were passed in. This allows the router to use these variables in all of its methods.

```
this._data = data;          // BlAPI4A2.Database
this._state = state;        // BlAPI4A2.DepState
```

Then the Run method looks at the AlienRouterStatus property of the state variable to determine what the reason is for this call to the Run method. The router will ignore all AlienRouterStatus types except BlAlienRouterStatus.blrsPreEdit.

```
if (this._state.AlienRouterStatus !=
   BlAlienRouterStatus.blrsPreEdit) return;
```

Next, the router gets the path of the current Blaise database from the DataFileName property of the _data variable and stores it in a variable called BlaiseDBPath. This path is used to locate the log file in the same directory as the Blaise database.

```
this.BlaiseDBPath = this._data.DataFileName.Substring(0,
    this._data.DataFileName.LastIndexOf(@"\"));
```

For debugging purposes, the router sends logging information to a log file. The CreateLogFilePath function sets the LogFilePath variable to the full path of the log file for this instance of the router. The name of the log file is generated using the name of the router class. Once the LogFilePath variable is set, the Log method can be used to send strings to the log file.

The variable called data that is passed in to the Run method is of type BlAPI4A2.Database. This object contains, via a number of index indirections, a reference to the current field in the Blaise instrument. In the case of our AddressVerification router, this will always be a reference to the AddressStart field of a VerifyAddress subblock of an instance of an AddressVerification block.

Since we know that the AddressStart field is two levels down from the top of the AddressVerification block, we can use the Parent property twice to find the root of this instance of AddressVerification. Having this, we can refer to any element within the AddressVerification block by name relative to this location.

The first Blaise field that the Run method looks at is the VerifyAddress.AddressStart field.

```
if (this._data.get_Field(this.activeField.Parent.Parent.Name +
```

```
          ".VerifyAddress.AddressStart").Text != "1")
```

This field is used to indicate whether or not the external application has already been run for this particular instance or not. After the external application runs, the router sets this field to “1”. On subsequent calls to the router, the router looks at this field and if it is not “1” then the router will run the external application. If this field is already set to “1” that means the external application already ran and the router will simply return without running the external application.

The router uses XML files to communicate the input parameters to the application, and to retrieve the output results from the application. Before launching the external application, the router updates the input XML file. The specifics of the XML file are beyond the scope of this paper.

The external application is launched using the System.Diagnostics.Process class.

```
    System.Diagnostics.Process program = new
       System.Diagnostics.Process();
    program.StartInfo.FileName =
       @"Rti.Data.AddressVerification.exe";
    program.Start();
    program.WaitForExit();
```

The StartInfo.Filename property is set to the name of the exectuable to run and the Start method is called. The router then calls WaitForExit, which causes the router to block on the return of the external application. Other application spawning mechanisms could have been used. We chose this method primarily for simplicity and because we did not want the router or Blaise to receive focus until the external application has exited.

After the external application exits, the router reads the output XML file and loads the returned values into Blaise variables.

```
this._data.get_Field(this._activeField.Parent.Parent.Name +
   ".OutputAddress." + addressElement).Text =
   outputNode.InnerText;
```

The router sets the current field to ”1” to indicate the external application has been run once.

```
this._data.get_Field(this.activeField.Parent.Parent.Name +
   ".VerifyAddress.AddressStart").Text = "1";
```

Finally, the router moves the Blaise instrument to the next question.

```
          this._state.AlienRouterAction =
             BlAPI4A2.BlAlienRouterAction.blraNextQuestion;
```

When code execution exits the Run method, the router is closed down and control returns to Blaise.

5. Case Study: Alien Router for Randomization of Questions

Many Blaise instruments require randomization of questions in various ways. In this section, we present one example we implemented by using an alien router.

The requirements were to randomize the order of 6 groups of 3 questions each and also randomly populate fills for hypothetical people used in the questions. More specifically, we were given the following tasks:

- Randomize the order in which the 6 question groups appear
- Randomize the order of the 3 questions within each group
- Randomize the names used for the people in the questions (based on the language of the interview)
- Construct the text of the randomized questions depending on the gender accociated with the randomly selected name

Figure 6 illustrates the randomization of questions and question groups.

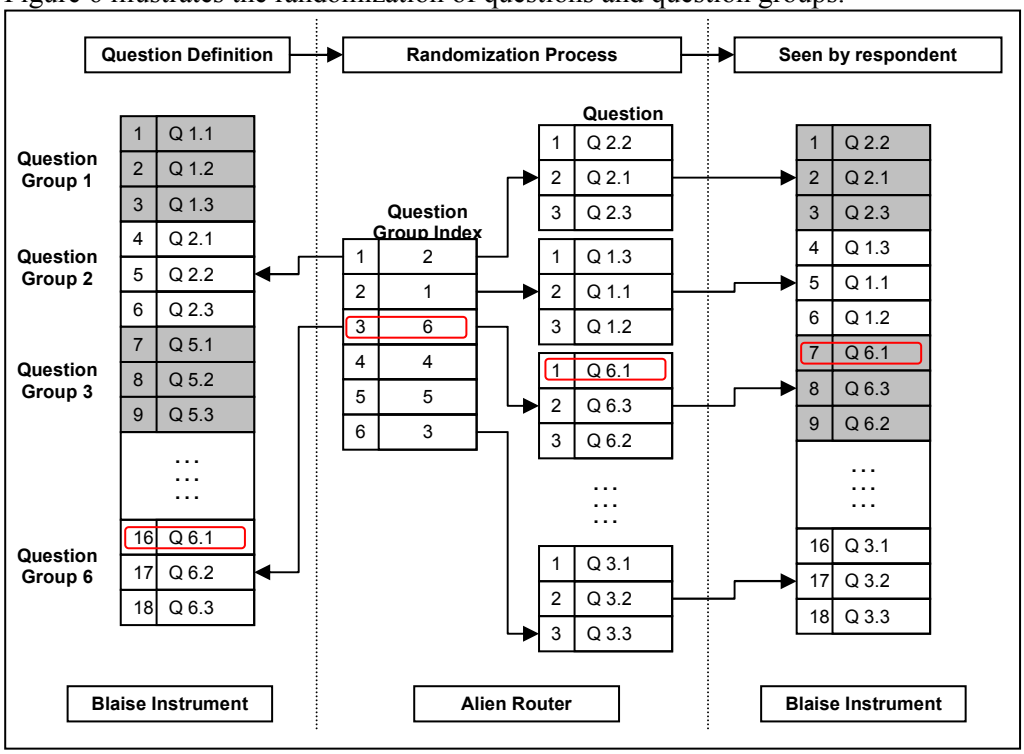


Figure 6 Question Randomization

Each question group is of a different Blaise type, but the number of response options is the same for all 18 questions. To construct the question text, we use a procedure in a Blaise instrument, while a VB.Net alien router calculates randomization numbers. The numbers are stored in the Blaise database where the Blaise instrument uses them to administer the questions in the correct order.

Several variables were added to the Blaise instrument for this purpose.

- Array RandomNumber[1..18] comtains an index into the question definition array. The index of RandomNumber itself is the order in which questions are asked. For example, if RandomNumber[7] contains the randomly selected value 16, then the seventh question asked of the respondent will be the 16th question defined, or question 1 from group 6 (labeled Q 6.1 in Figure 6).
- Array RandomName[1..18] contains the name that was used for the corresponding question. Continuing the example above, if RandomName[7] is "Daniel", then the name "Daniel" will be used in group 6, question 1.
- Array RandomGender[1..18] contains the gender for the corresponding name. In our example, RandomGender[7] is male.
- Array RandomField[1..18] actually asks the questions. Fields from this array are added to the route in the Blaise instrument and will be asked during an

interview. Text for the question and response options will be created "on the fly". The value entered in RandomField[7] in our example will be saved in a variable accociated with the first question 1 from group 6.

- Field RndSelection indicates the status of the call to the alien router. If all arrays defined above were filled with values, the router sets the value of the RndSelection field to "1" and will not be called again to calculate random numbers.

In the Blaise datamodel we add a type with fills for the response options. We create a procedure GetQuestAndType to create the response options ”on the fly”. Input parameters to the procedure are the question number, randomly selected name, and gender of selected name. Return values are the constructed question text and the type. In our example for the seventh question asked, the text for group 6, question 1 was created to use the name "Daniel". The resulting question asked of the respondent is shown in Figure 7.

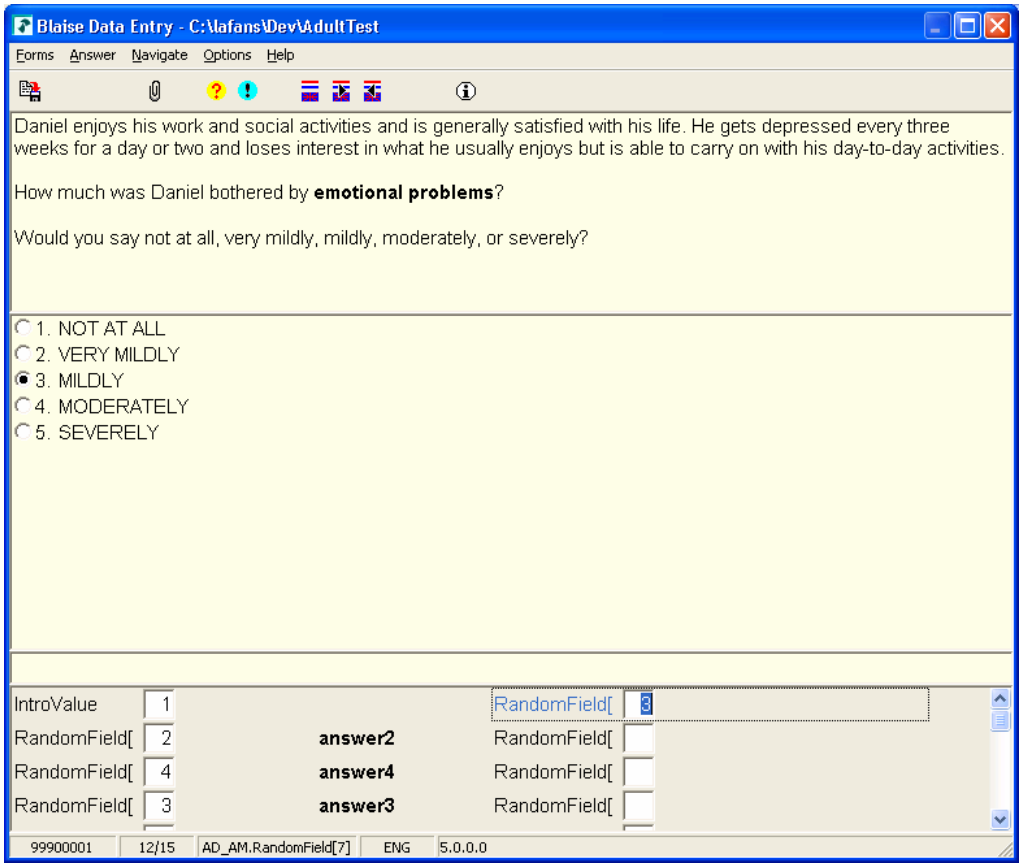


Figure 7 Randomization of questions

The values for the fields used in our example are shown in Figure 8. Analysts can later use these variables to reconstruct the order of questions and what exactly was asked.

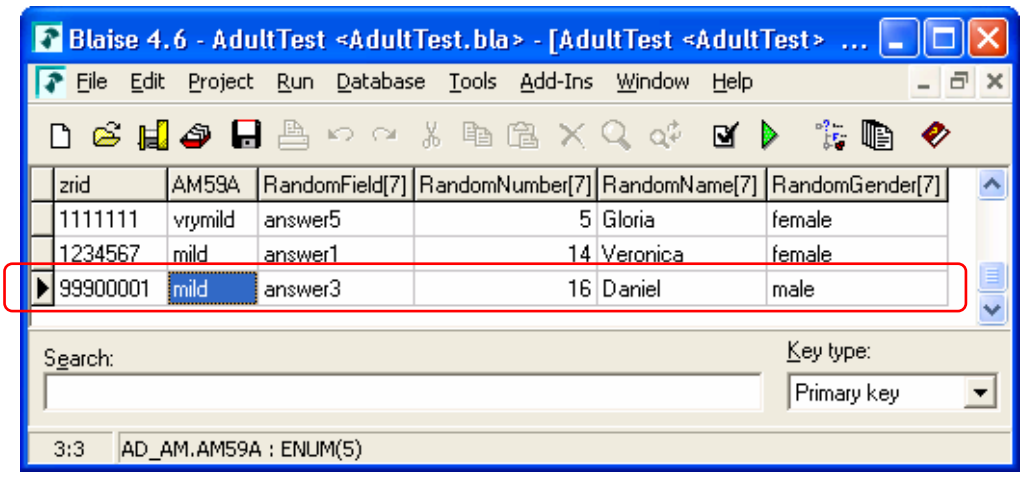


Figure 8 Randomization Data in Database browser

The Blaise source code is shown in Listing 3. See Listing 4 for a key excerpt from the VB.Net alien router.

6. Conclusion

If a study requires invocation of an external application during the Blaise interview, the following factors should be weighed:

- Does the application need to return data to the Blaise interview for future processing, and/or should it be stored in the Blaise database along with the interview data?
- Is complicated data processing needed?
- Does an external application need to be controlled?
- Is access to relational databases needed only for reading? For writing?
- Does collected data need to be in a format that is ready for analysis?
- How much time is available for development?

Depending on the requirements of a particular study, one of the discussed methods may be used. Our experience shows that many types of external applications can be integrated into Blaise surveys using these three methods.

Listing 1 Blaise Code for Address Verification Alien Router

```
BLOCK BAddressSearchElements
  FIELDS
    Street "What is the street address?": STRING[100], EMPTY
    City "What is the name of the city?": STRING[50], EMPTY
    State "What is the name of the state?": STRING[2], EMPTY
  RULES
    Street
    CITY
    { Call procedure to get a state from look-up table }
    GetState(State)
ENDBLOCK

BLOCK BAddressOutputElements
  FIELDS
    Street "QAS verified street address": STRING[100], EMPTY
    City "QAS verified city": STRING[50], EMPTY
```

```
    State "QAS verified state": STRING[2], EMPTY
    Zip "QAS verified zip": STRING[5], EMPTY

  RULES
    Street.SHOW
    City.SHOW
    State.SHOW
    Zip.SHOW
ENDBLOCK

BLOCK BVerifyAddress
  FIELDS
    AddressStart: 0..1, EMPTY

  ROUTER StartRouter
ALIEN('AddressVerificationRouter.AddressVerificationRouter','Run')

ENDBLOCK

BLOCK BAddressVerification
  FIELDS
    SearchAddress: BAddressSearchElements
    OutputAddress: BAddressOutputElements
    VerifyAddress: BVerifyAddress

  RULES
    SearchAddress.Keep
    IF SearchAddress.Street <> '' THEN
      VerifyAddress.AddressStart := EMPTY
      OutputAddress.Street := EMPTY
      OutputAddress.City := EMPTY
      OutputAddress.State := EMPTY
      OutputAddress.Zip := EMPTY
    ENDIF

    SearchAddress
    VerifyAddress.StartRouter
    OutputAddress

ENDBLOCK
```

Listing 2 C#.NET Code for Address Verification Alien Router

```
using BlAPI4A2;
namespace AddressVerificationRouter
{
  public class AddressVerificationRouter
  {

    private BlAPI4A2.Database _data;
    private BlAPI4A2.DepState _state;
    private BlAPI4A2.Field _activeField;
    private string BlaiseDBPath;
    private string LogFilePath;

    public void Run(BlAPI4A2.Database data, BlAPI4A2.DepState state)
    {
      try
      {
        // save the main Blaise objects as instance fields,
        //so they can be referenced in other methods
        this._data = data;
        this._state = state;

        //only handle BlAlienRouterStatus. blrsPreEdit
        if (this._state.AlienRouterStatus !=
```

```
            BlAlienRouterStatus.blrsPreEdit) return;

        // get the current directory of the Blaise database
        this.BlaiseDBPath = this._data.DataFileName.Substring(0,
              this._data.DataFileName.LastIndexOf(@"\"));
      CreateLogFilePath();

      // retreive the current field
      this.activeField = this._data.Screens.LayoutSetCollection
          [this._state.LayoutSetIndex].ParallelCollection
          [this._state.ParallelIndex].StoredPageCollection
          [this._state.StoredPageIndex].QuestionCollection
          [this._state.QuestionIndex].Field;


      // detect if the user is trying to move forward
      // or backward through the survey and call the
      // MoveForward or MoveBack routine acordingly.
      // The local blaise field AddressStart is either
      // blank or zero when moving forward, and is a 1
      // when moving back.
      if (this._data.get_Field(this._activeField.Parent.Parent.Name
+
          ".VerifyAddress.AddressStart").Text != "1")
      {
          // create the input file
          Log("Before BuildInputXml");
          this.BuildInputXml();

          // start the address verification program
          Log("Before StartVerificationProgram");
          this.StartVerificationProgram();
          Log("After StartVerificationProgram");

          // read the output and write to Blaise fields
this._data.get_Field(this.activeField.Parent.Parent.Name +
   ".OutputAddress." + addressElement).Text =
   outputNode.InnerText;

this._data.get_Field(this.activeField.Parent.Parent.Name +
   ".VerifyAddress.AddressStart").Text = "1";

          // proceed to the next question
          this._state.AlienRouterAction =
            BlAPI4A2.BlAlienRouterAction.blraNextQuestion;

      }
      }
      catch (Exception Ex)
      {
        this.Log(Ex.Message);
      }

    }


    private void CreateLogFilePath()
    {
      LogFilePath = BlaiseDBPath + @"\Log";
      DirectoryInfo di = new DirectoryInfo(LogFilePath);
      if(!di.Exists) di.Create();
      LogFilePath += @"\";
      //name the log file according to the class name
      //Pull out the class name
      LogFilePath +=

this.ToString().Substring(this.ToString().LastIndexOf(@".")
          + 1);
      LogFilePath += @"_Log.txt";
```

```
    }

    private void Log(string message)
    {
      StreamWriter writer = new StreamWriter(LogFilePath,
         true);
      writer.WriteLine(DateTime.Now.ToString() + ": " +
         message);
      writer.Flush();
      writer.Close();
    }

    }
}


private void StartVerificationProgram()
{
  try
  {
    System.Diagnostics.Process program = new
         System.Diagnostics.Process();
    Log("Changing directory to " + @"c:\test");
    Directory.SetCurrentDirectory(@"c:\test");
    program.StartInfo.FileName =
         @"Rti.Data.AddressVerification.exe";
    program.Start();
    program.WaitForExit();
    Log("Changing directory to " + BlaiseDBPath);
    Directory.SetCurrentDirectory(this.BlaiseDBPath);
  }
  catch (Exception Ex)
  {
    this.Log(Ex.Message);
  }

}
```

Listing 3 Blaise Code for Randomization of Questions

```
BLOCK BCallRouter
   FIELDS
     IntroValue: 1..1 {STRING[1], empty}
     ROUTER CallRandom ALIEN('Random.RandomRun','Run')
ENDBLOCK
TYPE
   TRandomField= (answer1 (1) "^Ttext[1]"
                , answer2 (2) "^Ttext[2]"
                , answer3 (3) "^Ttext[3]"
                , answer4 (4) "^Ttext[4]"
                , answer5 (5) "^Ttext[5]")
   INCLUDE "GetQuestAndType.prc"
   INCLUDE "AD_AMFields.inc"
   FIELDS
        RndSelection "Random selection made": 1..1
        RandomField "^Qtext[i]": array[1..18] of TRandomField
        RandomNumber : array[1..18] of integer
        RandomName : array[1..18] of string
        RandomGender : array[1..18] of Tgender
  LOCALS
    I, J, K, L : INTEGER
  AUXFIELDS
```

```
    Qtext: array[1..18] of TstrLong
    Ttext: array[1..5] of string
RULES
  NEWPAGE
  { Call the alien router to randomize order of AM54a-AM59c }
  AM54A_INTRO.CallRandom
  FOR i:= 1 TO 18 DO
    RandomNumber[i].keep
    RandomName[i].keep
    RandomGender[i].keep
    GetQuestAndType(RandomNumber[i],RandomName[i],
           RandomGender[i],Qtext[i])
    RandomField[i].ASK
  ENDDO
 { Initialize values in the corresponding fields }
 FOR j:=1 TO 18 DO
   IF  RandomNumber[j]=1 THEN
     IF RandomField[j] = RESPONSE THEN
        AM54A := ord(RandomField[j])
     ELSEIF RandomField[j] = DK THEN
        AM54A := DK
     ELSEIF RandomField[j] = RF THEN
        AM54A := RF
     ENDIF
   { ... }
   ELSEIF RandomNumber[j]=18 THEN
     IF RandomField[j] = RESPONSE THEN
        AM59c := ord(RandomField[j])
     ELSEIF RandomField[j] = DK THEN
        AM59c := DK
     ELSEIF RandomField[j] = RF THEN
        AM59c := RF
     ENDIF
   ENDIF
 ENDDO
```

Listing 4 VB.Net for Randomization of Questions in Alien Router

```
  'Read names from external file
  LoadNameAndGender(intTotalNames, Names, BlaiseDBPath)
  'Initialize random-number generator
  Randomize()
  'Populate array with randomly selected numbers for names
  PopulateArrays(rndName, usedName, 17, 18)
  'Populate array with randomly selected groups
  PopulateArrays(rndGroup, usedGroup, 5, 6)
  'Populate array with randomly selected questions in group
  For i = 0 To 5
    PopulateArrays(rndQuestion, usedQuestion, 2, 3)
    For k = 0 To 2
      rndField(i * 3 + k) = (rndGroup(i) - 1) * 3 +
                            rndQuestion(k)
      rndQuestion(k) = 0
      usedQuestion(k) = False
    Next
  Next
  'Write data into Blaise database (array: order of
questions)
  For i = 1 To 18
    sField = BlkName & ".RandomNumber[" & i.ToString & "]"
    db.Field(sField).Text = rndField(i - 1).ToString
  Next
```

```
'Get Active language variable
If sDep.LanguageIndex = 1 Then  'English
   Lang = 1
ElseIf sDep.LanguageIndex = 2 Then  'Spanish
   Lang = 2
End If
'Write data into Blaise database (arrays: names and gender)
For i = 1 To 18
  GetNameAndGender(Lang, Names, rndName(i - 1) - 1,
                   strName,intGender)
  sField = BlkName & ".RandomName[" & i.ToString & "]"
  db.Field(sField).Text = strName
  sField = BlkName & ".RandomGender[" & i.ToString & "]"
  db.Field(sField).Text = intGender.ToString
Next
```

Using the Blaise Component Pack from Within the .NET Framework to Develop Data Management Tools

Leonard Hart, Robert Thompson & John Mamer (Mathematica Policy Research, USA)

1. Introduction

Mathematica Policy Research, Inc. (MPR), has been conducting complex surveys for the United States federal government and U.S.-based foundations for over 35 years. Early surveys were conducted either in person or by telephone and generally were captured as hard-copy documents for data entry. Complex routing, rostering, and grid-based data collection were often a part of these surveys, thus the automated checking and control that became available with the advent of computer-assisted interviewing (CAI) was essential to increasing the quality of the data collected. MPR adopted computer-assisted telephone interviewing (CATI) tools in the early 1980s and computer-assisted personal interviewing (CAPI) tools later that decade. More recently, Web surveys have become an important aspect of our data collection activities. MPR currently uses the Blaise platform for most of its CATI, CAPI, and Web surveys.

To add to the difficulty of fielding these often long and complex surveys, many of MPR's surveys are longitudinal in nature and involve mobile, hard-to-locate populations. Automated support for sample management activities is essential in this environment. External to Blaise, MPR's Sample Management System (SMS) provides facilities for locating and tracking. The SMS, developed in Microsoft's .NET platform, usually operates simultaneously with Blaise in a production environment, and data are exchanged between the two systems in an overnight process.

Finally, data and sample management within the Blaise platform itself—beyond what is provided inherently in the Blaise call scheduler—is essential when dealing with our target populations so we may achieve the high response rates required in nearly all of our studies. Among the first tools we developed for our CATI operation was a Supervisory Review utility that allowed supervisors to review and re-status survey instruments in a very granular fashion. The first sections of this paper describe the evolution of the Supervisory Review facility from its initial implementation in Maniplus to its current version, renamed as the MPR Blaise Explorer and implemented in the .NET environment using the Blaise Component Pack (BCP). The next sections discuss the technical and performance issues we encountered in this effort. We end with conclusions based on this experience.

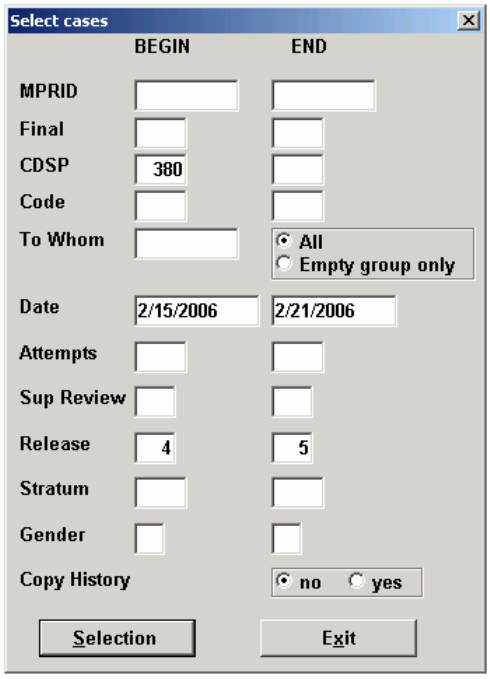
2. Supervisory Review Utility

MPR has called the tool built to handle the sample management tasks within the Blaise environment the "Supervisory Review" utility. In this section we provide an overview of the utility and discuss its implementation in Maniplus.

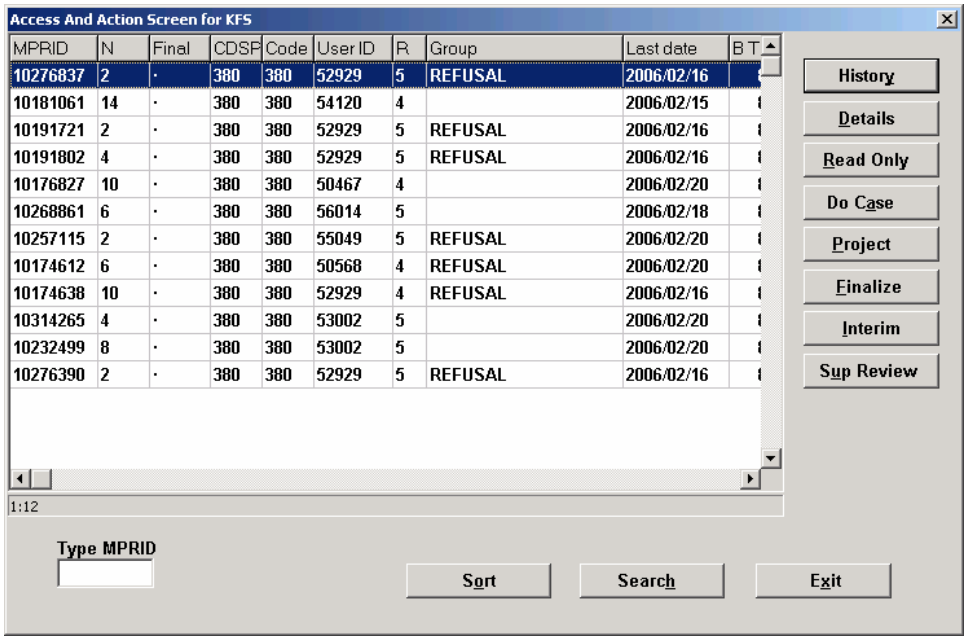
2.1. Overview of Supervisory Review Facility

The following paragraphs describe three of the primary functions provided in the Supervisory Review utility. While more functionality is implemented in the utility, this description provides a general overview of its basic purpose and functioning.

Reviewing Classes of Cases. One of the primary functions of the Supervisory Review utility is to provide the supervisors with the ability to display classes of cases for review. The menu below illustrates the ways in which supervisors may subset cases. They may extract cases by individual or ranges of primary keys (MPRID), final and current disposition, dates of interviews, and more. For example, the selection below will choose all cases with status code 380 administered between February 15th and 21st that are in release 4 and 5.



Cases are then displayed with a variety of options for looking at the case in more detail. The screen below lists 12 cases that fit the criteria.



Once the cases are displayed in this fashion, the supervisor may look at individual cases in a variety of ways, as indicated by the buttons on the right hand side of the screen. For instance, supervisors may look at the history of the case, may follow the logic of the interview in read-only mode, may load the case as a live interview with "Do Case," or may re-status a case as Final or Interim status. While this capability is quite powerful, the choice of fields in the *Select cases* dialog is relatively static and not under the control of the operational personnel.

Communicating with Interviewers on Specific Cases. Interviewers can communicate special situations that they have encountered on a case by adding notes to the case while administering the survey and putting the case in Supervisory Review status. Using the facility we described in the previous section, supervisors can pull up all cases that have a Supervisory Review status, read the notes, and then handle the case as appropriate, given the information they see in the notes and other aspects of the case history. In some instances this may involve changing a case's status, as described in the next section. In other instances this may involve writing notes back to the interviewer and placing the case back in the interviewing pool.

Changing Case Statuses. Within the Supervisory Review utility, supervisors may change the status of a case depending on the circumstances presented in the case history and the notes. Often this involves final statusing a case, but a number of other options are available. For example, on many surveys we have a survey instrument set up so that a case that has been refused twice will be automatically changed to Supervisory Review status. Within the Supervisory Review utility a supervisor may decide to send the case back to the interview pool so that it is handled by a refusal conversion interviewer. Alternatively, the supervisor may look at interviewer notes and see that the respondent has threatened a lawsuit if he or she is called again—a situation that has occurred more than once—and in this instance the case would be final statused as a Refusal and not called again.

2.2. Implementation of Supervisory Review in Maniplus

Early in this decade, MPR was in the process of changing its survey platform from CASES, a product of the University of California at Berkeley, to Blaise. Having used CASES for almost twenty years, MPR had developed a wide variety of tools to support CASES surveys. After several early Blaise efforts with relatively straightforward surveys, in 2003 we embarked on a multi-mode, worldwide survey of college graduates for the National Science Foundation. Some sort of Supervisory Review utility was essential for this complex multi-mode survey and the lead time was short. Given the tight integration of Maniplus with the Blaise data model and all other aspects of the Blaise platform, we deemed implementation of the Supervisory Review utility in Maniplus as the only feasible way of having the utility available in the required time frame.

A closely related reason for developing the utility in Maniplus was the availability of staff resources with the appropriate skills. In 2003, MPR was also beginning a transition to the .NET environment and the SQL Server database platform for all its major applications. While we had a number of proficient .NET and SQL Server programmers on staff, none of them had experience in the operational requirements of fielding a complex survey. This would have presented a steep learning curve had we embarked immediately on the task of implementing a Supervisory Review utility in .NET. At the same time we had a number of staff who understood surveys well and either knew Maniplus or could learn Maniplus with a relatively short learning curve. Both these reasons combined to make Maniplus the logical choice for getting the features we needed available in the requisite time frame.

2.3. Reasons for Change

As a consequence of the decision-making process outlined in the previous section, we believed from the beginning that a Supervisory Review utility implemented in Maniplus could be viewed as a prototype—albeit a very useful one—for the functionality that eventuality would be incorporated in a .NET application. As expected when Supervisory Review was first written in Maniplus, we have added more .NET trained staff that over time have became more familiar with survey requirements. The MPR Sample Management System (SMS), mentioned in the Introduction, became fully developed over the last several years using the .NET platform and a SQL Server backend. Hence, a Supervisory Review utility implemented in .NET that communicated to the Blaise database via the Application Programmer Interface (API) using the Blaise Component Pack could also be tightly integrated with the SMS.

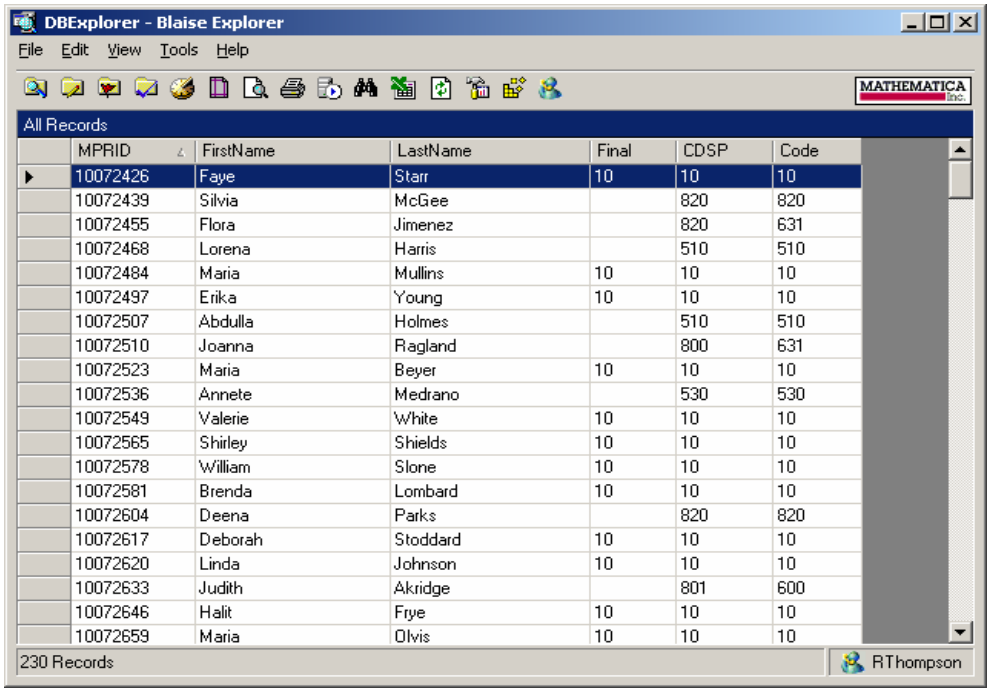
Another goal in the transition to a new system was to provide user authentication for a wide range of facilities. In order to grant access to the facilities deemed appropriate for a given staff member in a given position, we wanted to provide login authentication mechanism that would allow a user to access just those facilities.

Finally, in terms of user interfaces, for all our applications at MPR, we have tried to maintain the look and feel of Microsoft applications to ease user startup and training. Moving from Maniplus to .NET as an implementation platform allowed us to achieve this goal for the Supervisory Review facility.

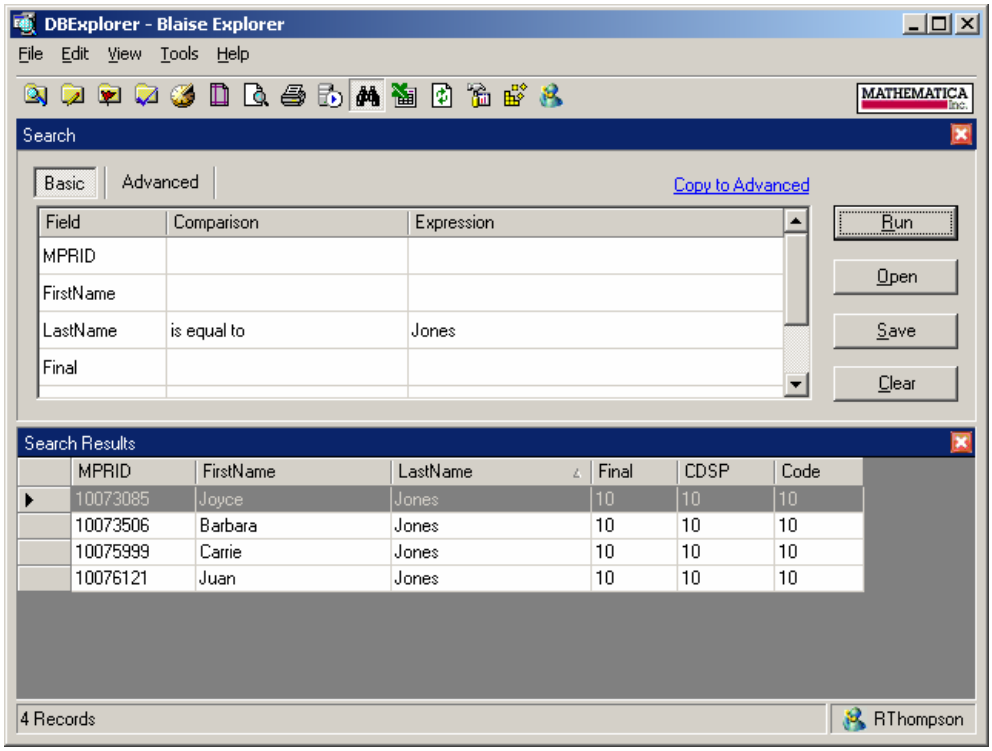
3. Blaise Explorer

"Supervisory Review" was a limiting name for the facilities we hoped to provide in the new version of this utility. As we moved to the .NET platform we planned, and still plan, to extend the functions embedded in the application beyond those that might be deemed supervisory in nature. As noted, the software would provide user authentication that allows authorized users access to the facilities they need. Because of the new functionality envisioned for the software we renamed the application the MPR Blaise Explorer.

On startup of MPR Blaise Explorer, the user is presented with the screen below, which lists all the cases. The new tool has the look and feel of a Microsoft application. In addition, supervisors at the call center can display useful data without getting a programmer involved to change the screen. Under the Supervisor Utility written in Maniplus, if a supervisor wanted to display different data on its main screen, a programmer needed to make the changes.

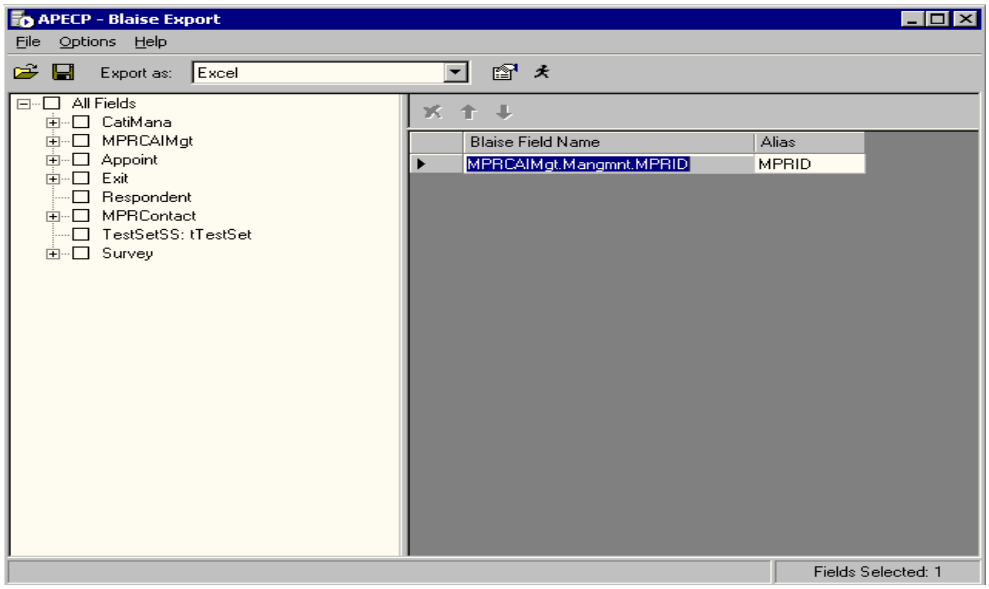


The search facility, which allows the user to subset the cases as desired, is denoted as in most Microsoft applications with the "binoculars" icon on the toolbar, or the user may use the menu bar. Invoking the search facility displays the following screen:

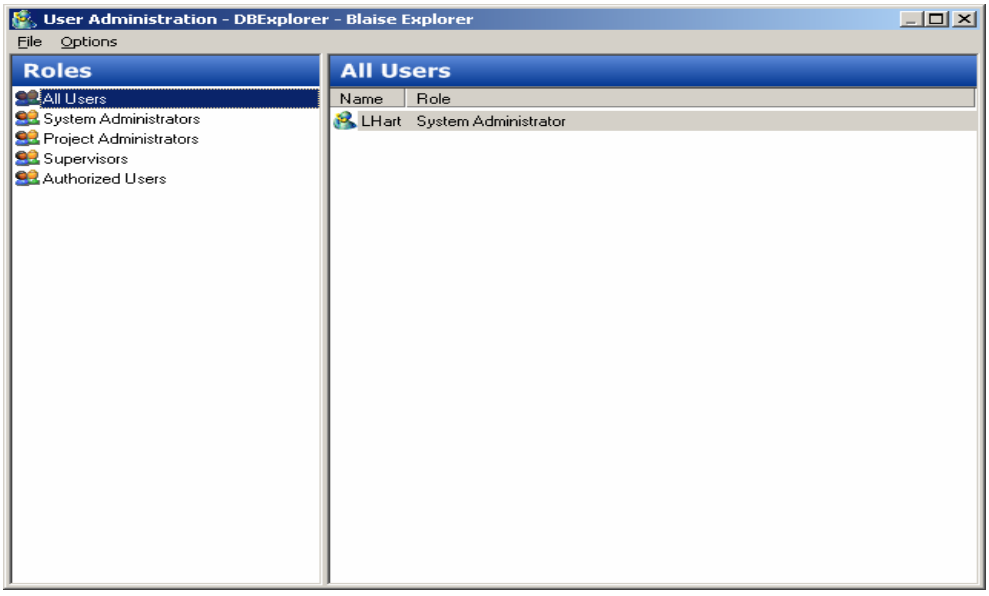


This screen allows the user to do many of the "case subsetting" functions of the Supervisory Review utility implemented in Maniplus but expands on those capabilities greatly. In the toolbar at the top of the screen are icons that allow the user to perform some of the activities on the individual cases such as display case history. As in most standard Windows applications, these activities are also made available though the main menu as well as right-click, context-sensitive popup menus. Reproducing Windows functions in this way allows users to work with the application in the manner that they are accustomed to.

One of the functions embedded in Blaise Explorer, which expands on and distinguishes Blaise Explorer from its Supervisory Review predecessor, is Blaise Export. We should note that due to the application's underlying architecture, Blaise Export can also be invoked as a standalone application in its own right and can be used by those not needing or requiring the full capabilities of Blaise Explorer. This facility displays the familiar Blaise data model "tree" and allows the user to select any parts of this tree to export to another data set for analysis. Thus at any time during a survey, supervisors or other project personnel may export Blaise data in a variety of formats (Excel, SAS XML, tab delimited, or XML) which they may then analyze using the application of their choice. Here is the Blaise Export screen:



Finally, a system administrator is given the facility to administer the system, granting and withholding rights for specific users to gain access to certain aspects of the system through the user administration screen below. This allows us great flexibility in granting specific rights to specific users.



4. Implementation in .NET

As previously noted, MPR adopted the use of the Microsoft .NET Framework as well as the Visual Studio .NET development environment shortly after their initial releases in February 2002. To date and for the most part, that experience has been a positive one. Although the .NET and Visual Studio .NET learning curves are steep, once programmers master them significant productivity gains can be realized.

The current implementation leverages a number of legacy and new technologies that include the Blaise 4.7 API Components, Microsoft Visual Basic 6.0, Microsoft SQL Server 2000, Microsoft .NET 1.1, Microsoft Visual Studio .NET 2003, and Microsoft Internet Information Server. While a solution relying solely on newer technologies would be preferable, the simple reality is that most enterprises cannot abandon their existing investment in legacy technologies, and the .NET Framework provides facilities for leveraging legacy applications.

4.1. Building Reusable .NET Components

While the discussion so far has provided a logical view of the Blaise Explorer and Blaise Export implementations, the physical implementations are markedly different in order to achieve the primary goal of developing a reusable set of .NET components that could be used to meet current and future MPR survey requirements. At the same time, these components were developed with extensibility in mind, to ensure that future requirements could be met with a minimal amount of code rewrite through a plug-in architecture. The core set of components reside in two .NET assemblies: MPR.Blaise.Shared and MPR.Shared. For those unfamiliar with .NET terminology, assemblies are simply the equivalent of executables or dynamic link libraries, and the .NET Framework makes little distinction between the two. MPR.Blaise.Shared is comprised of classes that are specific to Blaise and uses the Blaise 4.7 API. It contains all the Explorer, Export, and Administrator functionality outlined above. MPR.Shared is comprised of generic classes that contain no specific Blaise dependencies and are reusable in a wider variety of MPR applications.

Two additional assemblies, MPR.Blaise.Explorer and MPR.Blaise.Export, serve as standalone .NET applications that use the same MPR.Blaise.Shared assembly. Both executables have small footprints that optimize application start time, an important consideration with the deployment model that is discussed later. Excluding all of the boilerplate code automatically generated by Visual Studio .NET, both of these amount to a few lines of actual substantive code:

```
If MPR.Blaise.Shared.Authentication.Authenticate() Then
   Application.Run(New MPR.Blaise.Shared.Explorer)
End If
```

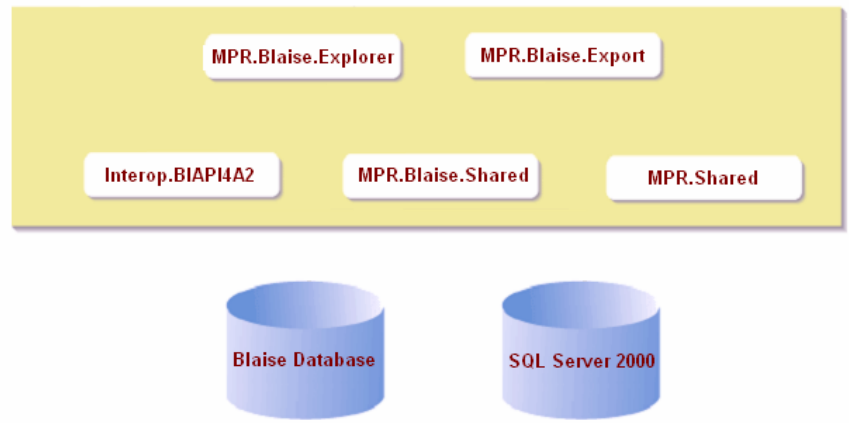
MPR.Blaise.Explorer

```
If MPR.Blaise.Shared.Authentication.Authenticate() Then
   Application.Run(New MPR.Blaise.Shared.Export)
End If
```

MPR.Blaise.Export

As shown here, both applications simply invoke MPR.Blaise.Shared where all of the Blaise specific functionality is located. The same approach can be used for exposing Administrator functionality as a separate application.

While a pure .NET implementation would have been preferable, the reality is that the Blaise 4.7 API is COM (Component Object Model) based. However, it can still be used within the .NET environment using a feature known as .NET COM interoperability. Any practical move toward next generation architectures using the .NET Framework undoubtedly needs to interoperate with existing COM applications, and .NET COM interoperability is the mechanism for leveraging an enterprise's existing COM applications. Using a simple command line utility available with the .NET Framework, a COM interoperability wrapper, Interop.BlAPI4A2, is automatically generated for the Blaise 4.7 API and provides MPR.Blaise.Shared with the ability to both access and update a Blaise dataset. Looking forward, this is an important first step towards eliminating disconnected, fragile overnight processes that are currently used to exchange data between Blaise and the MPR Sample Management System in favor of a more direct real-time data exchange. The initial development efforts have resulted in an architecture that can be extended in straightforward fashion. Currently that architecture can be visualized as shown below and effectively bridges the Blaise environment with the .NET environment:



4.2. Implementing Ad Hoc Blaise Search Capabilities

One of the more frequently recurring requests MPR users have is for specific data that are available within Blaise on an as needed basis. More often than not this requires substantial involvement of a Blaise programmer, whose time may or may not be available to immediately satisfy those requests; and, as we are all aware, users tend to want their data yesterday. One of the key components within the MPR tools that are being developed is the automatic maintenance of an index into a Blaise dataset. That index is maintained within a SQL Server database which provides users with the ability to search for specific Blaise records based on a preconfigured set of Blaise fields.

The figure below depicts a typical index that is maintained within SQL Server:

CaseIndex

Column Name	Data Type	Length	Allow Nulls
[MPRCAIMgt.Mangmnt.MPRID]	varchar	8	
[MPRContact.SampleInfo.FirstName]	varchar	20	✓
[MPRContact.SampleInfo.MiddleName]	varchar	1	✓
[MPRContact.SampleInfo.LastName]	varchar	20	✓
[MPRCAIMgt.Mangmnt.Final]	int	4	✓
[MPRCAIMgt.Mangmnt.CDSP]	int	4	✓
[MPRCAIMgt.Mangmnt.Code]	int	4	✓

Note that the SQL Server column names correspond directly to fully qualified Blaise field names. Although these fully qualified column names are unwieldy for typical users, more friendly aliases are displayed to end users throughout the user interface. Using the fully qualified Blaise field names as column names makes the index easier to maintain since there is a direct one-to-one mapping between Blaise and SQL Server. The initial challenge faced was how to maintain this SQL Server index without the need for a disconnected, manual process.

In older versions of Blaise there was a cumbersome process of executing external applications or DLLs through DEP options called alien routers or alien procedures. These were and still can be very useful options, but they have several drawbacks. With the introduction of Version 4.7 of Blaise a new feature called "actions" was introduced ("events" to .NET programmers) that added more flexibility and control to calls outside of the DEP. These actions have given us the control we needed to invoke DLLs at the appropriate time, such as updating a survey management database upon exiting a case in CATI.

MPRBLAPI4A4 is an ActiveX DLL written to the Blaise 4.7 API specification in Visual Basic 6.0. It contains two methods:

```
Update(Database As BlAPI4A2.Database, DepState As BlAPI4A2.DepState)

UpdateEx(ByVal FileName As BlAPI4A2.Field, ByVal Key As BlAPI4A2.Field)
```

Update is invoked by the Blaise Data Entry program in response to an action (such as exiting a case) while UpdateEx is invoked from Maniplus programs. Note that UpdateEx simply invokes Update by creating a BLAPI4A2.Database object and positioning the database to the record specified by the primary key parameter.

The Update method establishes an OLEDB connection to the SQL Server database and updates the index using the configured fields obtained from the Blaise Database that is positioned at the appropriate record. This keeps the SQL Server database in sync with the Blaise record at all times.

The MPR.Blaise.Shared .NET assembly includes classes that are used to configure the Blaise fields that are to be stored in the SQL Server index. Blaise Explorer uses these classes to expose the configuration user interface to authorized users, and the initial startup screen displayed by Blaise Explorer is nothing more than the contents of the SQL Server index table. The search facility allows users to locate records based on any of the configured fields and allows for complex binary searches. In addition, it provides an advanced text-based search that relies on SQL-like comparisons. Search specifications can be saved and later opened to accommodate frequently used searches, or programmers can create even more complex searches, save them and supply them to users. Since the Blaise primary key is part of the index, Blaise records can then be easily located, retrieved, or updated using the Blaise API. Reconfiguring the index to include additional fields

is a non-disruptive, dynamic process. Once reconfigured and the index is rebuilt, new fields become immediately available when the user refreshes the current display. Although the current implementation only provides for a single index per Blaise database, it could easily be extended to encompass multiple indexes per database, thereby providing users with different logical views of the database.

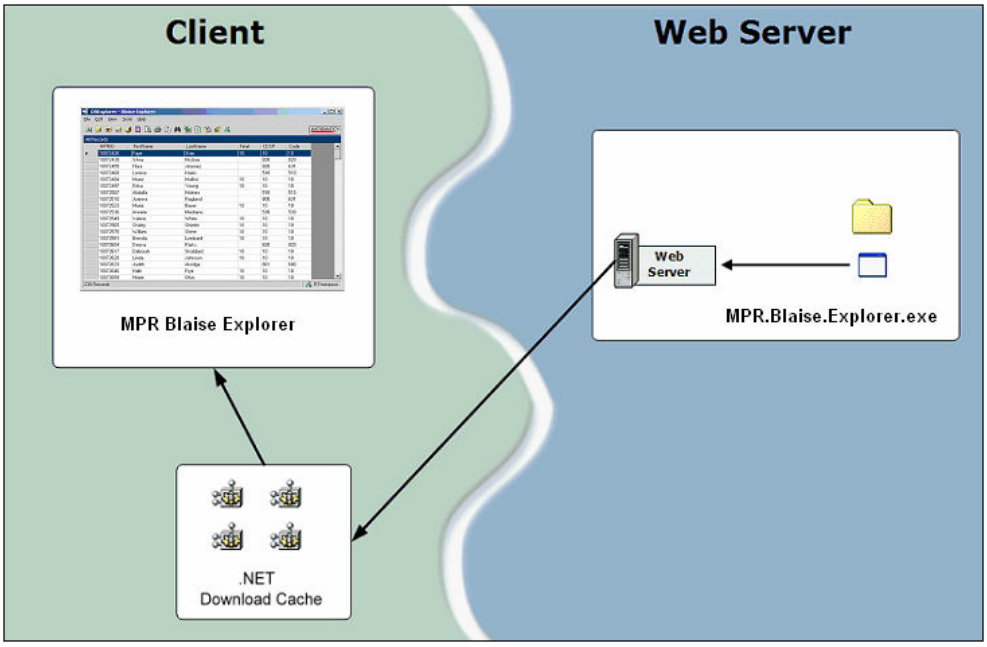
4.3. Deployment

Although not specific to the Blaise .NET integration effort, deploying and maintaining code on the Windows platform has never been an easy task largely due to logistical issues, and browser-based Web applications continue to control the mindshare. When IT managers see that simply updating a Web site automatically gives users the new version, it's hard to get them to go back to the painstaking process of making sure that all desktops in a corporation have been updated to the latest version of a traditional desktop application. However, one undeniable fact remains when it comes to browser-based Web applications: HTML is often inadequate for building fully-featured applications because its user interfaces are, by nature, primitive to use and difficult to implement and maintain because incompatibility between browser versions forces the use of lowest-common-denominator features and therefore the elimination of many user conveniences.

The .NET Framework's no-touch deployment model offers the best of both worlds, enabling richer desktop applications that leverage .NET Windows Forms to be as easily deployed as browser-based Web applications. Just like a Web application, a Windows Forms application can now be deployed on a Web server and launched by merely surfing to a URL (hyperlink) or simply providing the user with a desktop shortcut that points to the application on a Web server.

Deploying new versions of an application is as simple as copying new files to a virtual directory on a deployment server, and the new version is then immediately available to all users the next time they start the application. Deployment of new versions of an application is less disruptive to users, and fixes can be deployed in a timely fashion that generally requires no scheduled system down-time.

The figure below depicts No Touch Deployment. Note that while MPR uses Microsoft's Internet Information Server, applications deployed using the No Touch Deployment model can, in theory at least, reside on any vendor's Web server.



For logistical purposes, the MPR Blaise .NET application suite is deployed from a single Microsoft Internet Information Server that also houses the Microsoft SQL Server 2000 databases, but the applications will scale to different physical configurations as user demands grow. At the same time, this configuration has allowed some initial proof of concept development to proceed. Most recently, we were asked whether it was possible for the Blaise DEP to communicate directly with an XML Web Service. Since a usable XML Web Service was already being developed for another MPR project, within a matter of hours a sample Blaise Survey Instrument was implemented. We quickly showed that the Blaise 4.7 API could be used to invoke an ActiveX COM object that in turn used automatically generated .NET proxy classes to retrieve information from the Web service returning it to Blaise.

5. Performance Issues

While the transition towards Blaise and .NET integration hasn't always been a smooth one, it has been a promising one that has MPR rethinking its Blaise and MPR Sample Management System integration approach. Of those problems that we encountered, performance issues were most prevalent.

COM Interoperability Performance. Although .NET COM Interoperability allows organizations to leverage their existing COM applications while making the transition to the .NET Framework and Common Language Runtime, it shouldn't be considered a long term solution. The intent is to provide a migration path until those COM applications can be rewritten to the .NET specification. There is a significant amount of overhead involved in context switching between the unmanaged COM environment and the managed .NET environment. Within the MPR Blaise .NET application suite, this overhead is most noticeable when reading large numbers of Blaise records for export using MPR Blaise Export. While not so obvious when only reading a single record at a time, the cumulative overhead associated with reading thousands of records is very obvious to end users. For the most part they are willing to cope with this inefficiency when balanced against the fact they can now get the data they want when they want them rather than wait for programmer turnaround.

No Touch Deployment Performance. Using No Touch Deployment, applications are launched using Internet Explorer. Because there is no interaction between Internet Explorer and the .NET download cache, applications having large footprints can be slow to load because Internet Explorer doesn't know that a usable version of the application is available in the .NET download cache and will only use a recent version that is available in the Internet Explorer cache. In many cases, Internet Explorer unnecessarily downloads a new version of the application. MPR has minimized the effect of this problem by keeping the startup application executables small, as we previously described with MPR.Blaise.Explorer and MPR.Blaise.Export, and relegating the substantive code to referenced assemblies. The result is that Internet Explorer only downloads a small 20K or less executable that is passed off to the .NET Framework, and subsequent loading of the referenced assemblies is handled more efficiently by the .NET Framework, which can recognize their availability in the download cache.

System Requirements. Traditional standalone Windows application are resource intensive by their very nature. Adding the .NET Framework can make them even more resource intensive because of the need for just in time compilation and the more robust security mechanisms that .NET offers. In many cases the workstations that are being used are simply not up to the task. Although 6-year-old, 650 MHz machines with 128MB of memory were once state of the art, they quickly become

obsolete with rapidly changing technology. Nevertheless they can still be used with the .NET Framework. Memory upgrades do tend to yield better performance. Further extension of their useful lives could also be achieved by moving to SmartClient application architectures that shift resource utilization to a server that is more easily scaled to constantly increasing user demands.

Performance Summary. Despite these performance issues, there are a number of techniques that programmers can use to make the application feel more responsive. While users are willing to wait for a response from a lengthy operation, they quickly become concerned when no feedback is provided during that operation and start questioning whether they actually clicked a button or not. The single most important technique that programmers can employ to make the user experience more comfortable is to provide users with indication that the program is actually working on their request. Sometimes even a simple wait cursor suffices. With the .NET Framework this becomes an even easier task with built-in, easy to use support for worker threads, progress bars, and status bars that can be used to provide more feedback during lengthy operations. At the same time, by using the consistently responsive user interface, end users quickly learn that they can switch to another unrelated task or application and return later rather than waiting or watching for the completion of that lengthy operation.

6. Conclusions

MPR recently implemented its first survey using the new Blaise Explorer and Blaise Export. While Blaise Export is backward compatible with older versions of Blaise, the Blaise Explorer is using the latest features of Blaise 4.7.1.b1000. All desired features are not yet in place for Blaise Explorer but the current feedback from end users at our Survey Operations Center has been positive.

Up to this point two very important items have made the project successful: first, reaching out for end user input and, second, implementing the normal look and feel of a traditional Windows application. From the outset of this project we sought user input into the design. We had a series of meetings with the end users asking questions about what they liked about Supervisory Review and what could be improved. As we started to build the tools, we continually demonstrated their progress to end users and solicited user feedback. This iterative process of review and feedback has given the users more of a feeling of ownership.

The look and feel of a traditional Windows application is already paying off. As we were going through the development process, the end users were given several opportunities to beta-test the tools. With little training most users were able to adapt quickly and tried things that are common in Windows applications. An excellent example of this is the use of Windows-based data grids in Blaise Explorer. Experienced users of Windows applications knew that clicking on the title field would sort the data in that field.

As part of the development process, we have gained important experience with the Blaise Component Pack and the actions within the DEP available to invoke the interfaces we have programmed. The BCP has provided us with many of the features we have needed, but the performance issues described in the previous section presented significant challenges. The uses of the .NET platform for the front end introduced the COM interoperability performance issues noted above. In our initial tests, for instance, the “sequential read” of a Blaise dataset to pull cases based on our common selection criteria took over 30 minutes on a 12,000 record data set. Blaise indexing did not provide the facilities we needed, and an attempt to speed up the application using C#—instead of our standard implementation

language of VB .NET—did little to improve the speed. These measures did not provide the response times needed for production applications.

Our solution to the performance issues was to use the "on exit" action in the Blaise DEP and equivalent exits in several Manipula programs to maintain an indexed SQL server table of searchable data. Searches on this table for individual cases or classes of cases are very fast, and we can then access individual cases in the Blaise dataset via the primary key. Retrieval of single cases via the primary key provides near instantaneous response times. (Note: our use of the "on exit" action uncovered an issue with the Blaise implementation of this facility; in response, the Statistics Netherlands team obligingly provided us with an interim build.)

In conclusion, we believe we have found a technical approach and an approach to the user interface that has made Blaise Explorer, and the accompanying Blaise Export, a successful application at MPR. In the process we have gained important knowledge about the BCP and Blaise DEP actions that should be basis for the implementation of real-time interaction between Blaise and our other .NET applications in the future.

Using Blaise to apply edits to data held in an Input Data Warehouse

Fred Wensing (Australian Bureau of Statistics)

1. Introduction

Business survey and administrative data collected by the Australian Bureau of Statistics (ABS) are delivered to an Input Data Warehouse (IDW), which provides a single uniform database structure to support the various processes that need to be applied before final statistics are obtained. Through a system of status codes, the data in the IDW for all collections and all providers can be tracked from raw value through to final collected, imputed or estimated result.

An important process to be applied to all data is that of "editing" to detect inconsistencies and gaps in the collected data. Blaise has particular strengths in being able to check for anomalies and deliver the results of its checking to the operator. Blaise components can also be used to deliver the results of its checking to a system, thereby enabling edits to be effectively applied in a "batch" mode.

This paper describes the system that has been developed to apply edits to data held in an IDW to support both on-line and batch editing. The paper discusses the issues associated with transforming the data from the IDW into a file structure that can be acted on by Blaise and the way that edit results can be extracted and delivered to a management system for scrutiny and possible resolution.

2. Re-engineering of business statistics processes at ABS

The ABS has recently completed a three-year project, called the Business Statistics Innovation Program, which involved the re-engineering of operations, methodology and technology associated with business statistics. The main business drivers for this project were improved data quality, improvements in provider management, increased capacity, standardisation and operational efficiencies.

A significant aspect of the re-engineering was the adoption of technological and methodological initiatives to rationalise and reduce the disparate methods, processes and systems which were in current use.

All developments were defined and managed in the context of an End-to-End Business Framework which provided a high level view of the business requirements. A key component of the new business model is a central input data warehouse, which is described in more detail below. Another important feature is a shared provider environment to facilitate a total approach to the management of contact with providers.

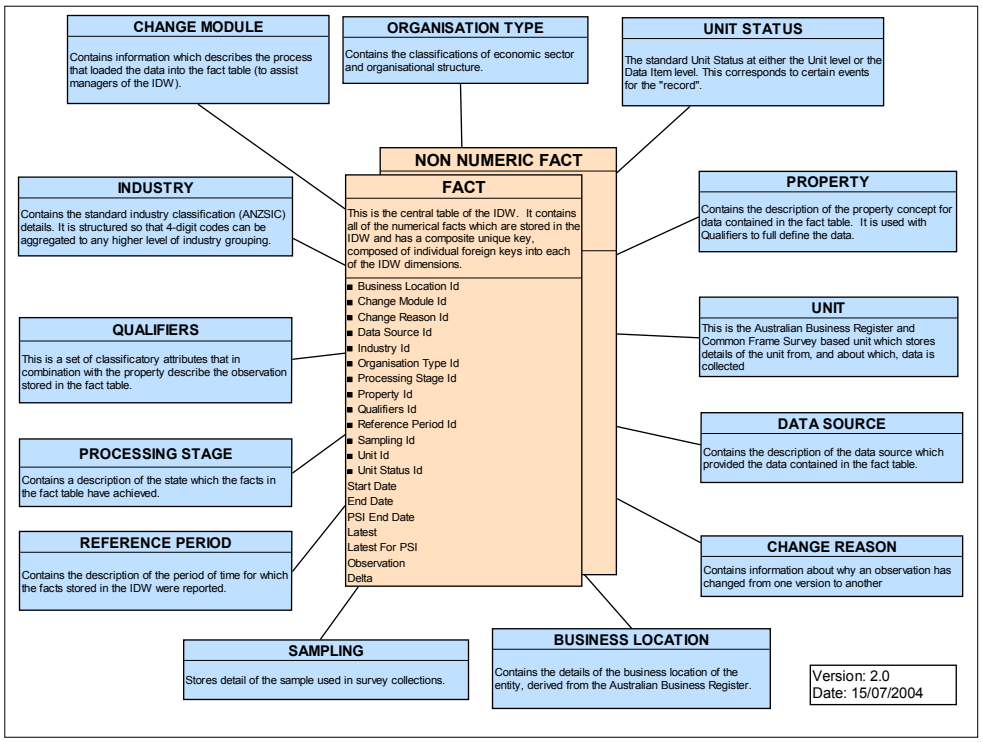
A key principle of the framework was the use of a modular approach to methodologies and systems, rather than a single all-inclusive solution. In this way processes are developed as needed, then updated and re-used where possible.

3. Input Data Warehouse (IDW)

The IDW is the central facility in the new business model and consists of a single managed unit record data store defined using a dimensional model (see Figure 1) and built in Oracle.

The centre of the dimensional model is a "fact" table which contains the numeric observations from the various data collections. Non-numeric observations are stored in a parallel "non numeric fact" table. The dimension tables define the various attributes of each fact, such as the data source, the business identity, industry etc. This structure, of a central fact table and a cluster of surrounding dimensional tables, is often called a "star schema".

Figure 1. An outline of the dimensional model used in the Input Data Warehouse at ABS



The fact table contains all the numerical observations in the IDW, stored as one fact per row, and has a composite unique key which is composed of the individual foreign keys into each of the IDW dimensions. Other attributes in the fact table identify the status of the observation and whether it is the latest for a particular key. The operating principle which applies whenever the fact table is updated, is to add rows rather than to overwrite them. In that way a full track-record is maintained of every observation through all its possible states (eg. captured, edited, imputed, estimated and released).

This dimensional model incorporates two tables (Property and Qualifier) which reflect the ABS's move to align with international standards on data definition and management, more particularly ISO 11179 (Specification and Standardisation of Data Elements), as described in *Oakley, Hamilton and McGrath (2004)*. These two tables together describe the data item definition.

The dimensional data model targets OLAP (Online Analytical Processing) and analysis activities. It produces a user-friendly model for constructing analyses and it gives good query performance. The model and associated infrastructure supports both "real time" transaction processing and post processing analysis.

To obtain the most benefit from the IDW, data should be loaded as soon as possible after it has been captured and before any corrections or other processes have been applied.

4. Editing requirements

One key business function that the systems are required to support is that of "editing", that is, the examination of data for the purposes of identification of anomalies and the possible correction of errors. It is for this function that Blaise software was identified as providing suitable functionality.

While the scope of editing is broad, the main interest in editing in the context of the IDW was related to input editing, which included the following requirements:

- to check the completeness of reported data (ie. identify gaps);
- to check the validity of reported data (ie. whether the answers are within acceptable range);
- to check the internal consistency of reported data (ie. cross check between items);
- to check reported data against historical data for the same unit;
- to distinguish between serious anomalies (usually fatal) and others (usually non-fatal);
- to apply edits in batch mode, as well as on-line;
- to report on all edit failures;
- to examine and resolve edit failures on-line (ie. make corrections).

It was expected that input editing would be carried out at the record level, within a particular collection, and to be associated with a particular collection form (or instrument).

For editing to take place, a record from a defined collection and using a defined form would be extracted from the IDW and examined in the editing tool (Blaise in this case). Edit failures would be identified and possibly corrected with any changes returned to the IDW, along with an updated status code for those facts. In a batch editing mode it was expected that the edit failure information will be passed to a reporting facility for scrutiny later.

5. The decision for a Blaise editing solution

Blaise was identified as the most appropriate software to provide the desired editing solution. The following factors were relevant in coming to this conclusion:

- Blaise has the functionality to apply the kind of edits required;
- Blaise provides a data editing interface which will support on-line examination of data records;
- Blaise OLE DB links and the API should enable it to be integrated with the IDW systems;
- Blaise was available "off-the-shelf" and would therefore be more cost effective than building our own solution;
- Blaise software was already in use elsewhere in the ABS and was therefore a lower cost solution.

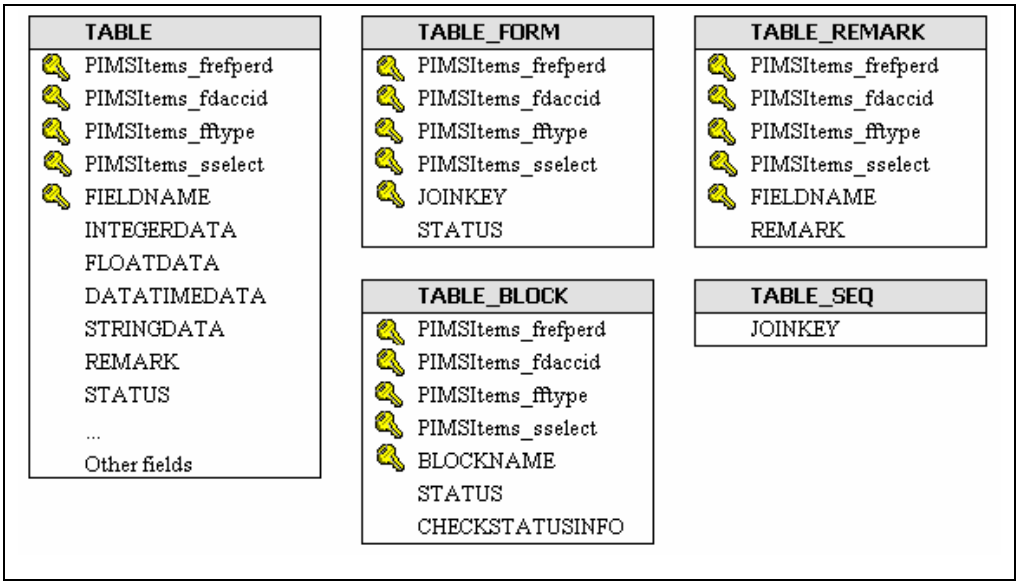
6. Using Blaise with data held in a dimensional model

While Blaise has the capability to access data in other data stores, through the Blaise datalink and OLE DB technology, none of the data partition types supported by that technology corresponds directly with the dimensional model described above.

Therefore, in order to operate on the IDW data using Blaise, it was necessary to develop processes which would extract the entries from the IDW (in Oracle), convert and load them to an interim store which was then readable by Blaise. Another process would also be required to transfer any changes made during editing back to the IDW.

Two data structures were considered suitable for the interim data store: a set of Oracle tables, formatted to suit Blaise; or a Blaise database. Oracle tables were preferable because the IDW was also held in Oracle, and the conversion and loading was considered to be simpler as a result. The OLE DB partition type found to be most suitable for the interim store was the "in-depth" partition type which has as its main table one which holds the data in a Field/Value format. The other tables for this partition type are used to store the Form information, Block information and Remarks (see Figure 2).

Figure 2. Tables used in the Blaise OLE DB in-depth partition



Before Blaise can use the interim store it would be necessary to transform the observations from the IDW into the set of tables needed for the in-depth partition type. The transformation which takes place involves:

- matching the observations with the business identity to create the primary keys for the in-depth tables;
- converting the IDW identities for the extracted observations to their corresponding full Blaise name;
- populating the main data table of the in-depth partition with the observations by inserting the full Blaise name and its correspnding value;
- adding an entry to the Form table for each record that is extracted;
- adding entries to the Block table for each block in the instrument (for each record).

Population of the Block table is important because it is used to track the status of the edits. One small oversight in the original implementation of this part of the transformation, however, was to forget to place an entry in the Block table for the datamodel itself. This omission led to a problem with some instruments because fields assigned in the top level of the instrument were not recognised as being on the path until the rules in the instrument were applied a second time. This was corrected by adding the internal datamodel name to the Block table (for each record).

Vital in the loading of the observations from the IDW, was the ability to map the identity (found in the Property dimension) of the facts extracted from the IDW to the field names in the Blaise instrument. This issue is discussed later.

Figure 3 shows what the main table and the block table look like after they have been loaded with one record. You will notice that the main data table has a field/value appearance. The Block table also contains a block called "main" which is the name used on the DATAMODEL statement in the instrument.

Figure 3. Sample of in-depth tables containing data for one record

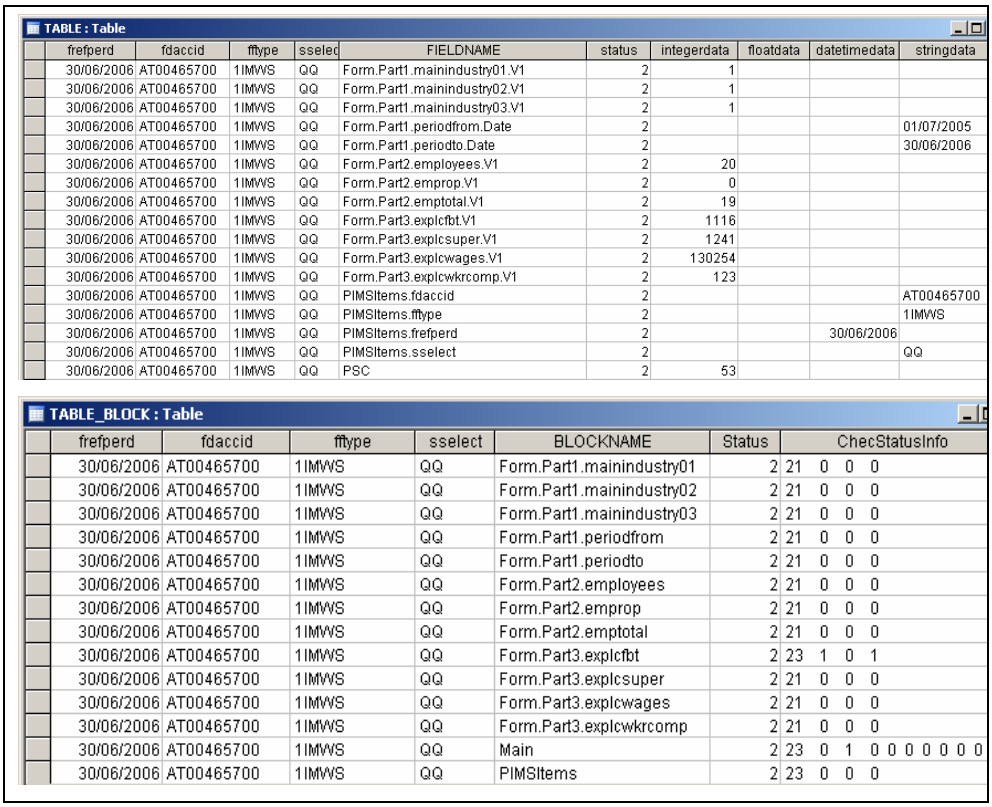


TABLE : Table

frefperd	fdaccid	fftype	sselec	FIELDNAME	status	integerdata	floatdata	datetimedata	stringdata
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry01.V1	2	1			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry02.V1	2	1			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry03.V1	2	1			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.periodfrom.Date	2				01/07/2005
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.periodto.Date	2				30/06/2006
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.employees.V1	2	20			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.emprop.V1	2	0			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.emptotal.V1	2	19			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcfbt.V1	2	1116			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcsuper.V1	2	1241			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcwages.V1	2	130254			
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcwkrcomp.V1	2	123			
30/06/2006	AT00465700	1IMWS	QQ	PIMSItems.fdaccid	2				AT00465700
30/06/2006	AT00465700	1IMWS	QQ	PIMSItems.fftype	2				1IMWS
30/06/2006	AT00465700	1IMWS	QQ	PIMSItems.frefperd	2			30/06/2006	
30/06/2006	AT00465700	1IMWS	QQ	PIMSItems.sselect	2				QQ
30/06/2006	AT00465700	1IMWS	QQ	PSC	2	53			

TABLE_BLOCK : Table

frefperd	fdaccid	fftype	sselect	BLOCKNAME	Status	ChecStatusInfo
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry01	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry02	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.mainindustry03	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.periodfrom	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part1.periodto	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.employees	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.emprop	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part2.emptotal	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcfbt	2	23 1 0 1
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcsuper	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcwages	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Form.Part3.explcwkrcomp	2	21 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	Main	2	23 0 1 0 0 0 0 0 0
30/06/2006	AT00465700	1IMWS	QQ	PIMSItems	2	23 0 0 0

7. Operating Blaise editing in a batch mode

There are two ways in which edits can be applied using Blaise. The edit conditions can be incorporated into a program which is run against the data (eg. in a Manipula program) or they can be incorporated into the instrument logic itself. While the former solution is one that fits a traditional view of editing, the latter is much better because the edits are activated whenever the instrument is opened, and re-activated whenever any of the data is changed.

With the edit rules incorporated into the instrument logic, batch editing amounts to simply opening each record, applying the rules, extracting details of the edit

failures, then closing the record and moving on to the next one. All these process steps can be carried out using any standard scripting language and the Blaise API.

Furthermore, the Blaise API provides access to the error objects held in the Blaise Object Model where it can extract the error messages, along with the fields and values involved. All that remains is for the error messages to be processed for reporting.

8. How to report the results of the application of edits

The error messages produced by the Blaise instrument can be conveniently transported to other systems by converting them to XML. In that format, messages concerning the editing outcome(s) can be readily sent to other systems for processing and display.

The processing script which extracted the editing results was enhanced to produce one XML report for each unit edited. Figure 4 shows typical XML entries for a unit in which some errors have been detected.

Figure 4. Extract of XML containing information about edit failures

```
- <pin:DetectedAnomalies xmlns:pin="urn:abs.gov.au/StatisticsProcessInputs">
  - <pin:UnitAttributes>
      <pin:UnitAttribute name="PIMSItems.FDACCID">AT00465700</pin:UnitAttribute>
      <pin:UnitAttribute name="PIMSItems.FREFPERD">30-JUN-2005</pin:UnitAttribute>
      <pin:UnitAttribute name="PIMSItems.FFTYPE">1IMWS</pin:UnitAttribute>
      <pin:UnitAttribute name="PIMSItems.SSELECT">QQ</pin:UnitAttribute>
      <pin:UnitAttribute name="PIMSItems.UANZSIC">XXXX</pin:UnitAttribute>
      <pin:ProcessingCode>53</pin:ProcessingCode>
      <pin:ProcessingCodeDescription>Fatal Anomalies</pin:ProcessingCodeDescription>
    </pin:UnitAttributes>
  - <pin:Anomalies>
    - <pin:Anomaly>
        <pin:Type>Soft</pin:Type>
      - <pin:AssociatedCells>
        - <pin:Cell>
            <pin:Name>Form.Part2.EMPTOTAL.V1</pin:Name>
            <pin:Value>19</pin:Value>
          </pin:Cell>
        </pin:AssociatedCells>
        <pin:Message>4.03 (Q) Total employment differs from BM by more than 20% , check that the
          reporting is correct.</pin:Message>
      </pin:Anomaly>
    - <pin:Anomaly>
        <pin:Type>Hard</pin:Type>
      - <pin:AssociatedCells>
        - <pin:Cell>
            <pin:Name>Form.Part2.EMPPROP.V1</pin:Name>
            <pin:Value>0</pin:Value>
          </pin:Cell>
        </pin:AssociatedCells>
        <pin:Message>4.05 (F) Working Proprietors reported and frame TOLO is not equal to 6 or 7 or
          8.</pin:Message>
      </pin:Anomaly>
    - <pin:Anomaly>
```

9. Managing the editing process - The Editor's Toolkit

To complete the editing facilities, it was necessary to develop a user interface that could be used to manage the editing process. This is particularly important given the number of collections which would eventually be using the editing systems.

The Editor's Toolkit (ETK) is a facility, built in Lotus Notes, which provides the required operational functionality to manage the editing process. Through this

facility it is possible for operators to see the status of records that have been edited. From within this facility they can launch Blaise editing in both on-line and batch modes. The manager can see summary information about the editing that has been done and can assign work to the operators.

The ETK consists of a database containing documents (or forms) for some or all units involved in a collection. For some surveys, only units which have failed at least one edit are recorded in the ETK. For other surveys, a document is created for every unit, even those which are "clean".

The ETK is mail-enabled and receives the XML messages that have been produced by the editing processes. These XML messages are then transformed into documents that end up containing the texts of the error messages along with the fieldname and value of the first involved field. The views in the database provide summary information about the edit failures and the status of records.

Figure 5 shows a typical view of records in the Editors Toolkit showing form type (code number) and error message with counts against each. The navigator window shows links to other views and actions. Figure 6 shows a sample document from the ETK with edit failures listed.

To provide a better understanding of the editing processes and how Blaise, the ETK and the IDW interact, a process diagram has been included at Attachment A.

Figure 5. Typical view of documents in the Editors Toolkit showing them grouped by error messages

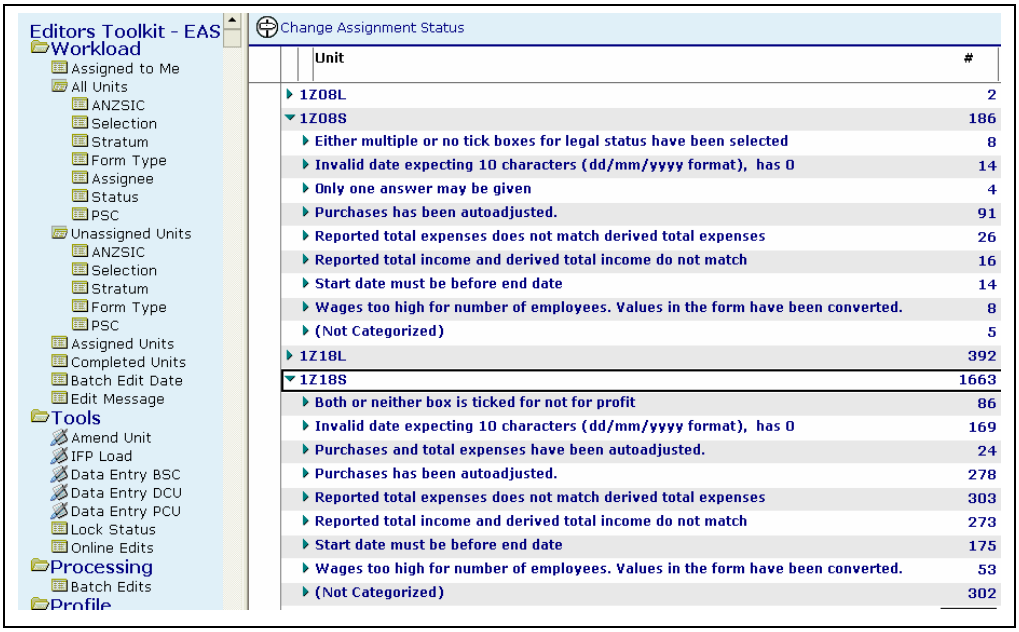
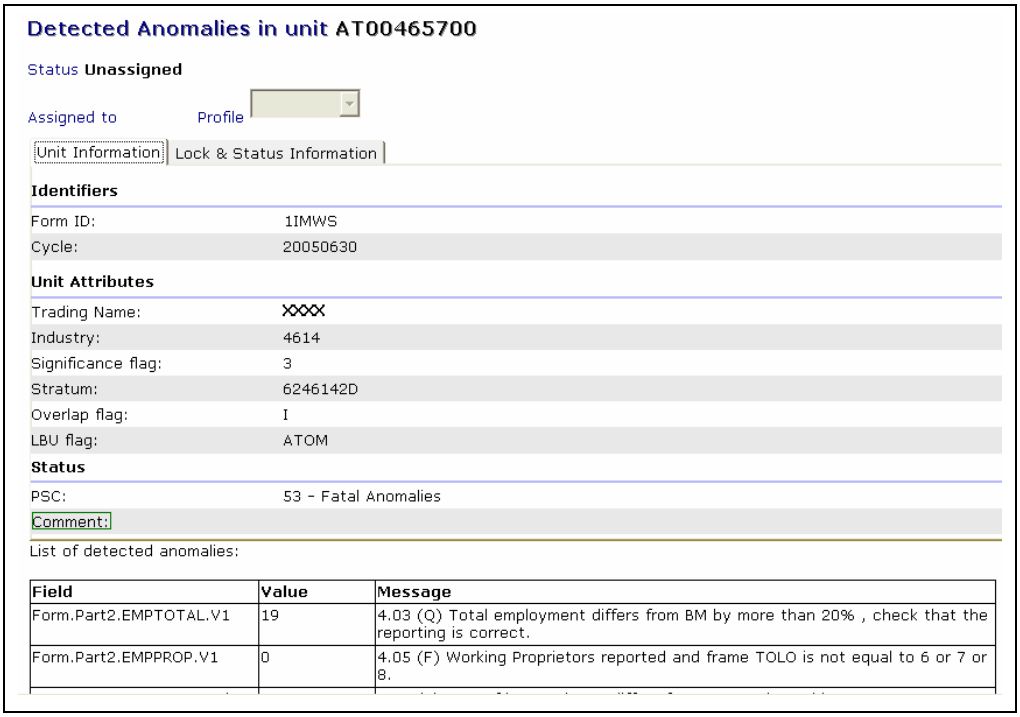


Figure 6. Typical view of a document from the Editors Toolkit showing listing of error messages



10. Instrument design and operation for interactive editing

Once the data has been edited (and reports produced) the editing system needs to enable the operator to examine particular failed records for possible correction.

A special Blaise instrument design and configuration has been developed to provide useful features for interactive editing. These include:

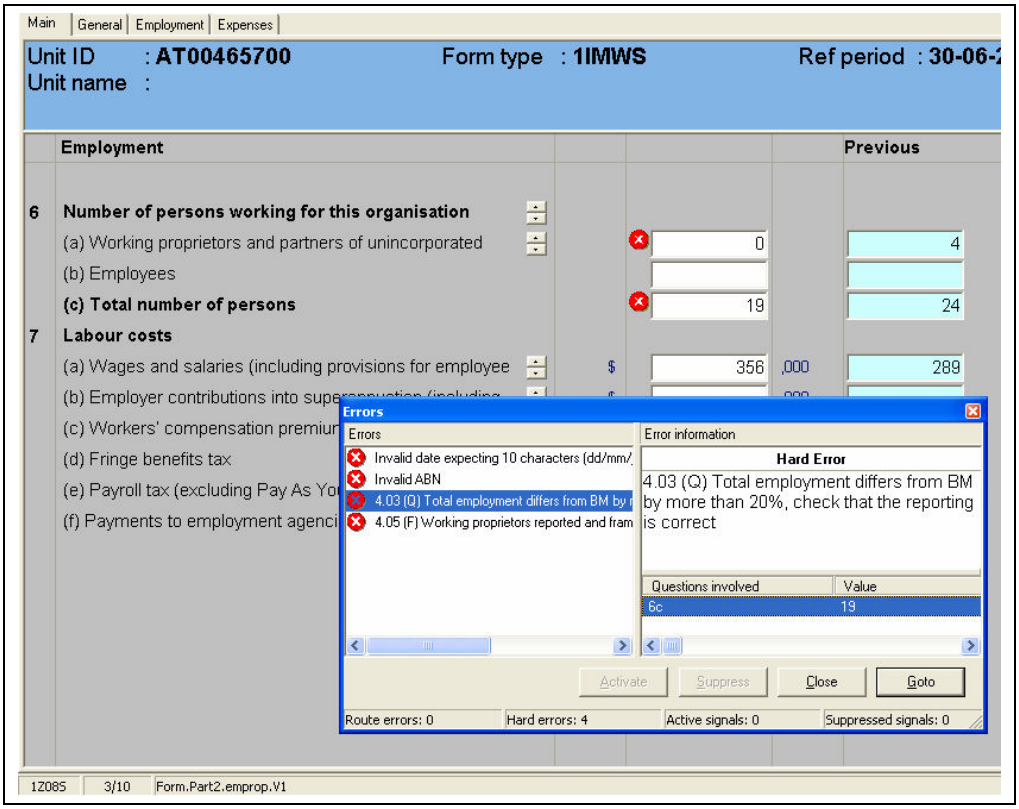
- screen appearance which is similar to the paper forms with which the data was collected;
- use of the error and warning symbols to signify fields with edit failure(s);
- free navigation through all fields by using the editing mode rather than interviewing mode.

The special screen design for editing makes use of rows and columns in the Form Pane to display all the important text information associated with a data field in a way that approaches the appearance used on the paper form. This was achieved by developing standard Blocks for certain field types (number, string, percentage etc) and desired layout, then using parameters to pass the various texts into the block for presentation on screen. The implementation of this block approach was based, in part, on the method described by *De Bolster (2004)*. Figure 7 shows a typical screen for an instrument which has been set up for editing.

Some flexibility was added to the editing instrument to enable matching data from previous cycles of the survey to be displayed on the same "row" of the screen, if so desired.

To support the application of different edits in batch mode, compared to on-line, a special field called OperationMode was incorporated into each instrument. Edits which only apply in a particular mode are then written to involve this field. The field is set to the particular mode by the system whenever the edits are applied, thereby causing selected edits to be activated or not-activated as the case may be.

Figure 7. Typical screen layout for a business survey instrument used in editing, showing also the edit message dialog box



With this instrument all the usual Blaise navigation techniques, such as "Navigate/Show all errors", are available to make it easy for operators to locate anomalies for investigation. A typical error dialog is also shown in Figure 7.

11. Aligning field definitions from the dimensional model with fields in Blaise

As mentioned previously, one important issue to be handled in the editing system was to be able to map the identities of the facts extracted from the IDW to the field names used in the Blaise instrument. That way data could be readily transformed from the IDW store to a Blaise-readable form.

Data elements in the IDW dimensional model are identified through a system name, which is a character string of up to 16 characters in length. The system names are recorded in the Property dimension in the IDW. These system names are assigned to each data element by staff from a central standards section and are unique across all business collections.

While the early Blaise instruments used for editing made direct use of the system name, this limited the design possibilities for those instruments. When instruments contained special elements, such as arrays, the direct matching on name became difficult and additional logic had to be added to the extraction/transformation programs.

With the adoption of the instrument design described above, the alignment of Blaise field names and the IDW fact identities was simplified so that a match occurred when the system name could be found "within" the full Blaise name (for matching purposes, the dot character "." found in full Blaise names was treated as an underscore character "_") (see Figure 8). This meant that there was more scope

for variations in the placement of fields in the instrument while making the matching of names more generic.

Figure 8. Illustration of matching various IDW system names to Blaise field names

IDW system name		Full Blaise name
EMPTOTAL	=	Form.Part2.EMPTOTAL.V1
INCSERVOTR1_V	=	Form.Part3.INCSERVOTR1.V1
INCSERVOTR1_S	=	Form.Part3.INCSERVOTR1.Specify
PIMSItems_FDACCID	=	PIMSItems.FDACCID
INCSERVOTR1_OLD_V1	=	Form.Part3.INCSERVOTR1.Old.V1

The alignment method described here does place some constraints on the development of Blaise instruments because it needs to make use of the pre-defined system names. However, it is expected that this method will be superseded after changes are made to make the system more metadata-driven. In particular, it is hoped that instruments used for editing of business data will be "generated" from the same metadata that will be used to define the data elements in the IDW.

12. Performance issues for interactive editing

The system described in this paper is largely focused on the batch editing process. Although the same architecture could apply for interactive (on-line) editing, it was found that there were some performance issues to contend with.

Performance of the Blaise OLE DB Interface (BOI) file in Oracle suffered somewhat when accessed on-line over the WAN. This was rectified by adding a further step to the extraction and loading of the in-depth partition. The additional step, used for interactive editing only, would transfer the data in the main data table into a native Blaise file located on the operator's computer. Interaction of the operator with this local data was then carried out through the data entry program (DEP.exe) rather than the API.

Dispatching the XML messages containing the edit results to the ETK produced an unnecessary round-trip and consequential delay for interactive editing. Given that the ETK was still connected, and open, it was more effective to transfer the edit results directly to the relevant document in the ETK, by-passing the XML steps.

The configuration of the editing process involves a chain of events in which various connections are made between Lotus Notes, Oracle and Blaise objects, not to mention additional links that these components may have with other parts of the environment. It appeared that each time a unit was selected for interactive editing, all these connections are opened up before the record is presented on the screen. When the editing of that unit is over, all the connections are closed again. A number of avenues of investigation are planned to deal with this. One solution would be to reconfigure the systems to keep the various connections "open" so that performance will improve with the second and subsequent record. Another solution would be for the system to extract more than one record from the IDW and deliver them locally to the user for editing.

13. Conclusion

The system described by this paper illustrates how it is possible for Blaise to perform an editing role in the context of an input data warehouse, by exploiting the OLE DB datalink and Blaise API technology.

While the Blaise instruments which have been developed have been designed for data editing they can readily be modified to perform data entry functions, or with some additional work, for computer assisted interviewing.

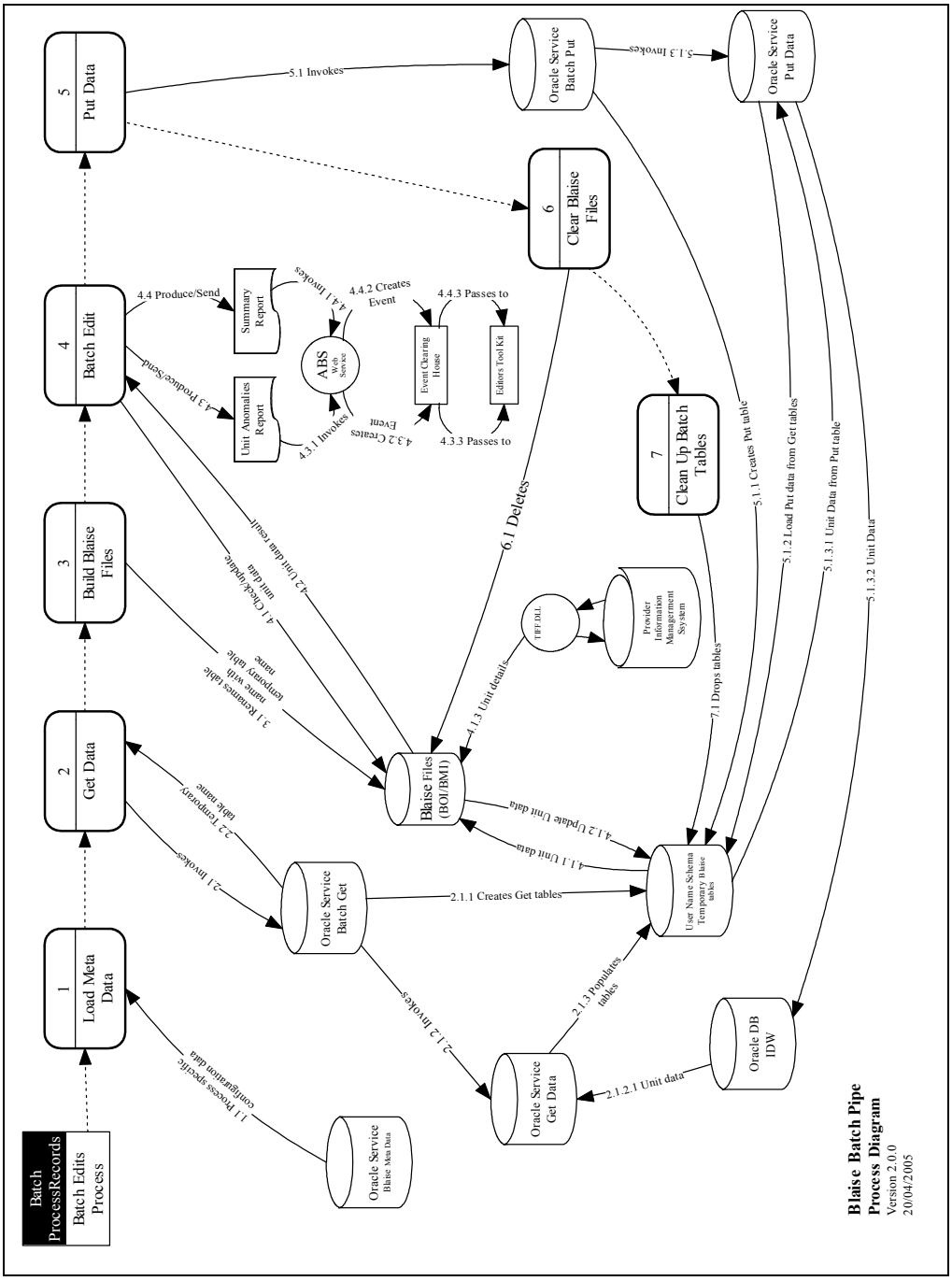
The Blaise instruments used for editing have been developed by hand but a set of appropriate instrument standards and considerable common source code has now been developed. The next logical step is to look at ways that the instruments can be generated from the same metadata that is used to support the IDW.

14. References

Oakley G., Hamilton, A., McGrath M.(2004): Using ISO/IEC 11179 to help with metadata management problems, Presented to the Joint UNECE/Eurostat/OECD Work Session on Statistical Metadata (METIS) , Geneva, 9-11 February 2004.

De Bolster, G. (2004): Generating Blaise with Blaise, Proceedings of the 9th International Blaise Users Conference, Gatineau, Québec, Canada, September 2004.

Attachment A. Process diagram for Blaise batch editing of data in the Input Data Warehouse at the ABS.



Session 4.2: Alongside the Data (Paradata)

Blaise PlayBack and Recovery System
Youhong Liu & Gina-Qian Cheung
(University of Michigan, USA)

Active Management: A New Strategy for Managing Data Collection
Luc Tremblay
(Statistics Canada)

Work Flow for the Weighting of the German Microcensus Data Using Blaise Bascula
Kirsten Iversen
(Federal Statistical Office, Germany)

Blaise PlayBack and Recovery System

Youhong Liu & Gina-Qian Cheung (University of Michigan, USA)

1. Introduction

At the University of Michigan's Survey Research Center (SRC), the primary method for evaluating quality and providing feedback to field interviewers is for a supervisor or quality control specialist to review the completed questionnaires in Blaise while listening to an audio recording of the interview.

For this purpose on one study, SRC had developed a system to "playback" complete interview data. The technique utilized was to transmit each new keystroke read from a file to the Blaise Data Entry Program (DEP). This required that the Playback program (developed using Visual Basic 6) interact continuously with the DEP. The problem with this approach was that the two systems were sometimes out of sync, because the DEP executed rules while the VB program fed the data to the DEP. In addition, the system was further limited in that it was hard coded for a specific study.

The U.S. Bureau of the Census was interested using a generic version of such an audit trail, as were many Blaise users. They provided Michigan with funding and specifications for development such a system.

As a result, SRC developed a new generic Playback system. Everything in the new system is fully integrated, minimizing synchronization problems. For two of its functions (see section 3), it has an interface that replaces the DEP (with a "Mini Data Entry" routine) while retaining its essential elements, and provides control over both execution of rules and display of data. For a third function, it provides additional control for keeping the DEP synchronized with playback. This paper describes functions of the new system as well as techniques utilized in its development.

2. Audit Trail Information File (AIF)

First, the system assumes the use of the Blaise 4.7 audit trail DLL and an audit trail information file (.AIF), in order produce an ADT (audit trail) file in the format expected by the new Playback program. That is, the [MetaName].aif or audittrail.aif file must be present in the directory the system expects to find the audit trail DLL (Dynamic Link Library) during the interview. Either Blaise 4.6 or Blaise 4.7 may be used in data collection to achieve the desired results, but the Blaise 4.7 DLL and AIF are essential.

For playing back audit trails with the new SRC system, the setting for the options in the AIF should be as follows:

```
[AuditTrail]
LeaveField = 1
EnterField = 1
Action = 1
KeyStroke = 1
Classify = 1 or 0
Lookup = 1 or 0
ErrorDlg = 1
Forms = 1
```

When set to 1, the audit trail DLL will produce a separate audit file for each form

Extra = 1 or 0
FlushFile = 1
Mouse = 0 or 1
PreciseTiming = 1 or 0
KeyTranslation = 1

When set to 1, the audit trail will translate the virtual key code to a character or (when appropriate) to a description of the key

WriteLean = 0 or 1

This parameter can be specified as 0 or 1, WriteLean=1 option creates a more compact audit trail file. WriteLean=0 creates a more readable file. The program is written to process ADTs produced by both options so the user has flexibility according to his/her need.

KeyStrokeOneLine = 1
AuditFolder =
AuditTempFolder =

3. How Playback Works

There are three main components in the system. They are Playback, Recover, and Recover & RunDep. A description of each follows.

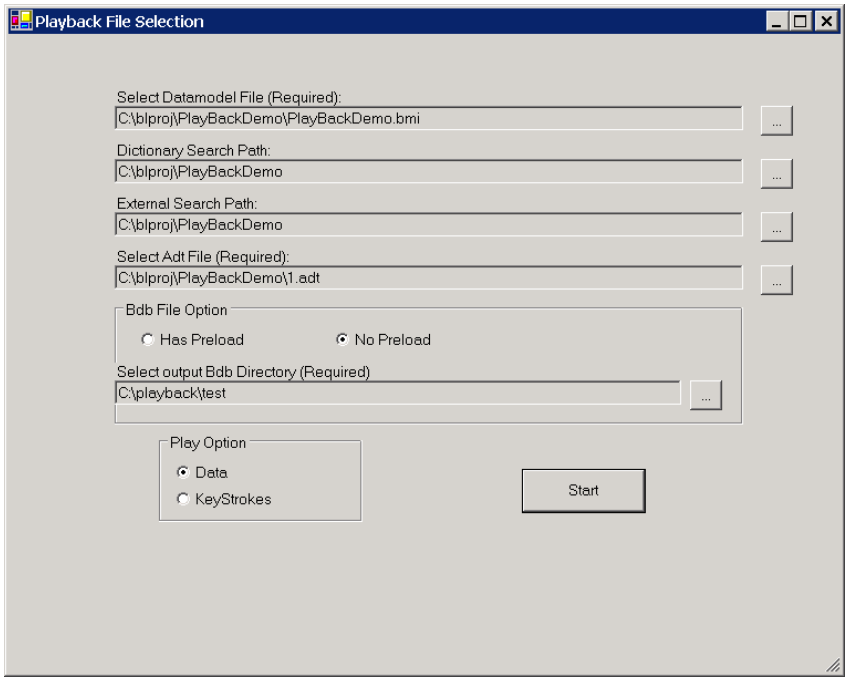
3.1. Playback

The first key screen in the playback option is the file selection screen (Figure 1). On this screen, the user needs to specify:

- The datamodel file (BMI) for the project;
- Dictionary Search Path; once you select a Datamodel, the Dictionary Search Path[1] will fill automatically with the path in which the datamodel was found; although, you may select another path;
- External Search Path; once you select a Datamodel file, the Dictionary Search Path[2] will fill automatically with the path in which the datamodel was found; although, you may select another path;
- Audit Trail File; an ADT file created using the previously selected datamodel or BMI file;
- Database or BDB file; (1) if preload data is expected, check "Has Preload" and select a BDB file location and file name; that BDB must contain the appropriate preload data, as it existed for the selected case and resulting ADT file the when the ADT file was created; (2) otherwise select "No Preload" and specify a directory where the program will create and store a new BDB file, named after the ADT, e.g. 123.bdb corresponding to123.adt; the preload database is important only if the Blaise instrument has fill, routing, and calculation logic that depends on preloaded variables.

[1] Indicates where dictionary files (typically those mentioned in the USES section of a Blaise data model) are located.
[2] Indicates the search path for external data files.

Figure 1: Playback Parameters Screen



- Play Option; (1) select "Data" to play back the final values recorded in the ADT for questions; (2) or select "Keystrokes" play back all keystrokes and mouse actions, and to show the final values for questions.
- Starting Playback: The user presses the "Start" button (Figure 1) to begin playing back keystrokes for the selected ADT file. At this point, the program begins to process the adt file. It searches for an "Enter Field," finds a corresponding "Leave Field" line, and saves all the information between these two lines in a structured array. Information stored and required for playback includes:

  FieldName – Blaise complete name of the field (conform the hierarchy in the data dictionary)
  ShortName – Blaise name of the field as defined in the surrounding block
  EnterLineNo – The "Enter Field" line number in the adt file for processing purpose
  LeaveLineNo – The "Leave Field" line number in the adt file for processing purpose
  IsLeaveFieldLine – Is there a "Leave Field" line in the adt block? Sometimes the "Enter Field" line does not have a corresponding "Leave Field"; in this case, use "Enter Field" value
  EnterValue – The value of the field when entering the field
  LeaveValue – The value of the field when leaving the field
  IsEnterForm – Is it the first field in a new form?
  IsLeaveForm – Is it the last field in the form?
  HasErrorDlg – Has soft check or hard check occurred in the field?
  ErrorStr – Error string saved for later processing
  Remarkchanged – Is Remark used in the field?
  ReMarkStr –Remark string saved for later process
  FormNO – Form Number
  Status – Is DK or RF entered for this field?
  Keys_() – Array of Key Strokes
  KeyCount – Number of "(KEY:)s" in the field block

  All of this information is stored for use during the playback. After the information is extracted, the program moves to the next screen.

Figure 2: Playback Mini Data Entry Screen

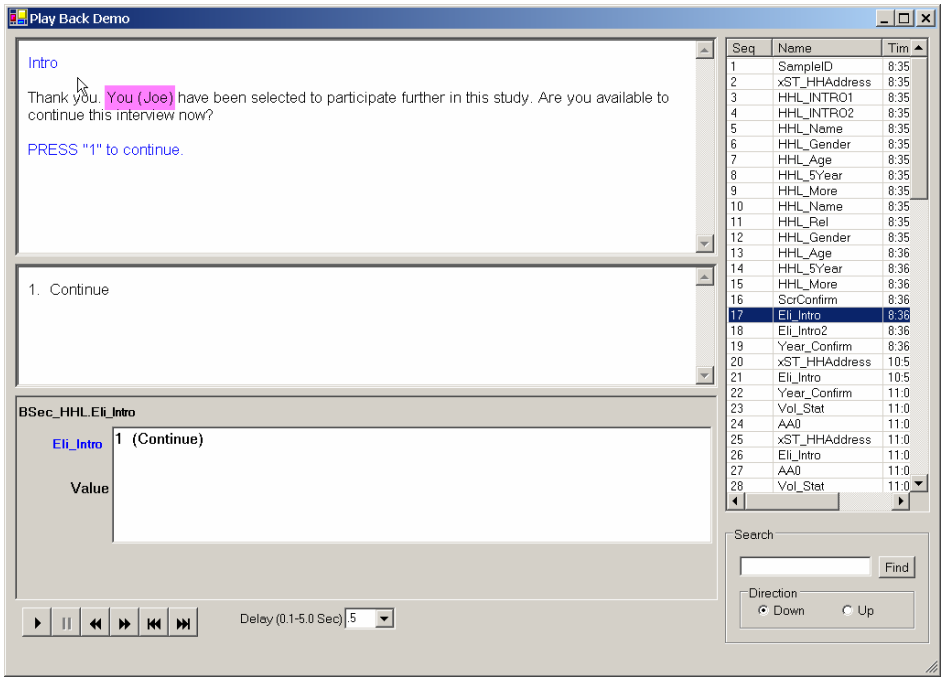


Figure 2 shows a playback screen which has been named a "Mini Data Entry" screen because it resembles Blaise's Data Entry Program, displaying what the interviewer sees in the Form Pane and the Answer List Pane in a Question Text Window and Category Window, respectively. At the back end, the data is fed to the Blaise database and the rules are executed. The reason for a back-end database is that it can obtain proper fills for question and enumeration category texts. For example in Figure 2, "You (Joe)" is an example of fill text.

Components of this screen are:

a. Title, which is extracted from the selected BMI's description
b. Question Text Window, displays the field question text; the control used to display this text is a web browser control; the text is output to a temporary html file, and then is displayed in the web browser; the web browser control is chosen because it is easy to show different fonts and colors in the control, allowing the simulation of what the interviewer sees;
c. Category Window, displays the response options or categories, if the field is an enumeration type;
d. Keystrokes Window, displays all keystrokes and mouse actions entered at the field;
e. Value, displays the "Leave Field" value; if there is no "Leave Field" line associated with a field Playback just display the "Enter Field";
f. Navigation Buttons, which appear at the bottom of the "Mini Data Entry" screen, and control navigation through the instrument; they are:
   ▶ (Play) which starts systematic playback of data or keystrokes, moving through fields automatically in sequence,
   ⏸ Pause or Stop button; may be pressed at anytime to pause or stop execution,
   ⏩ Single step forwards; is used to move to the next field,
   ⏪ Single step backwards; is used to move to the previous field,
   ⏭ Jump to end; will move the last field in the list, and
   ⏮ Jump to beginning; will move to the first field in the list;
g. Field Name List, displays all field names in the instrument in the sequence in which they appear in the ADT; the available options are to:
   • Double click a field in the list to jump to that field;

- Highlight a range of fields from the list; then press ▶(Play) to automatically go through fields selected, or press ⏭to go through them one by one.

h. Delay Time, controls the delay in seconds between fields when ▶ (play) is pressed; the user can choose between a 0 and 5 second delay before moving to the next field;

i. Search, allows the user to jump to a field by searching for a field name. It supports partial and complete string searches;

If a soft (SIGNAL) or hard check (CHECK) is raised, or if the remark is accessed, Playback will display the contents of those windows (Figures 3 and 4). For a check, it will note the interviewer action (e.g., "Error is Suppressed").

Figure 3: Example of Soft Check Window and Its Contents for Displayed Field, with Resulting Interviewer Action

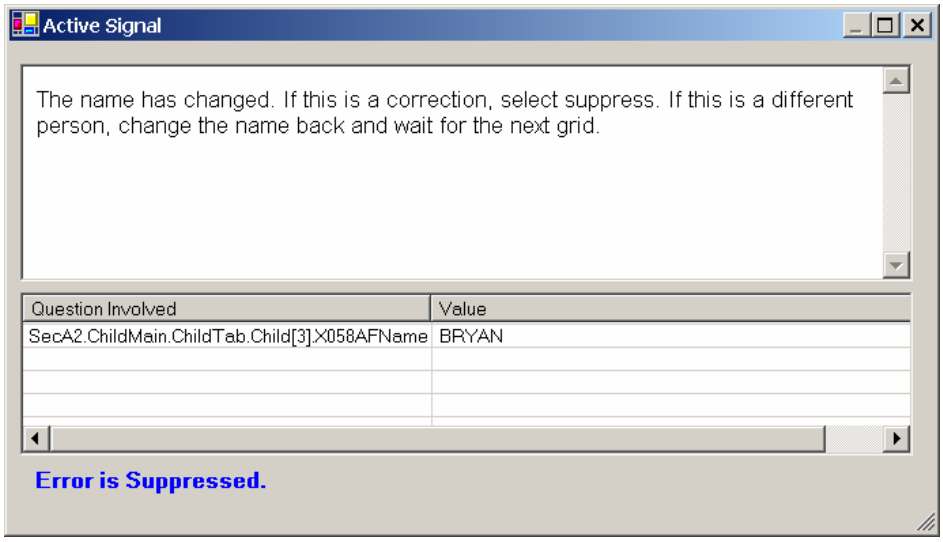
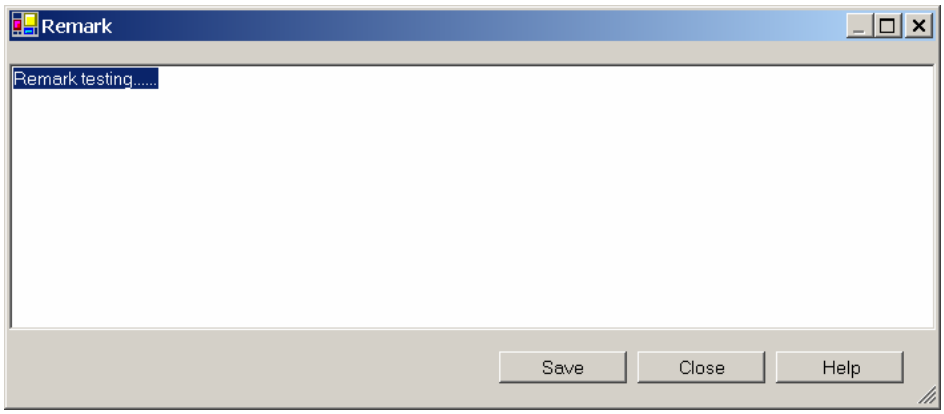


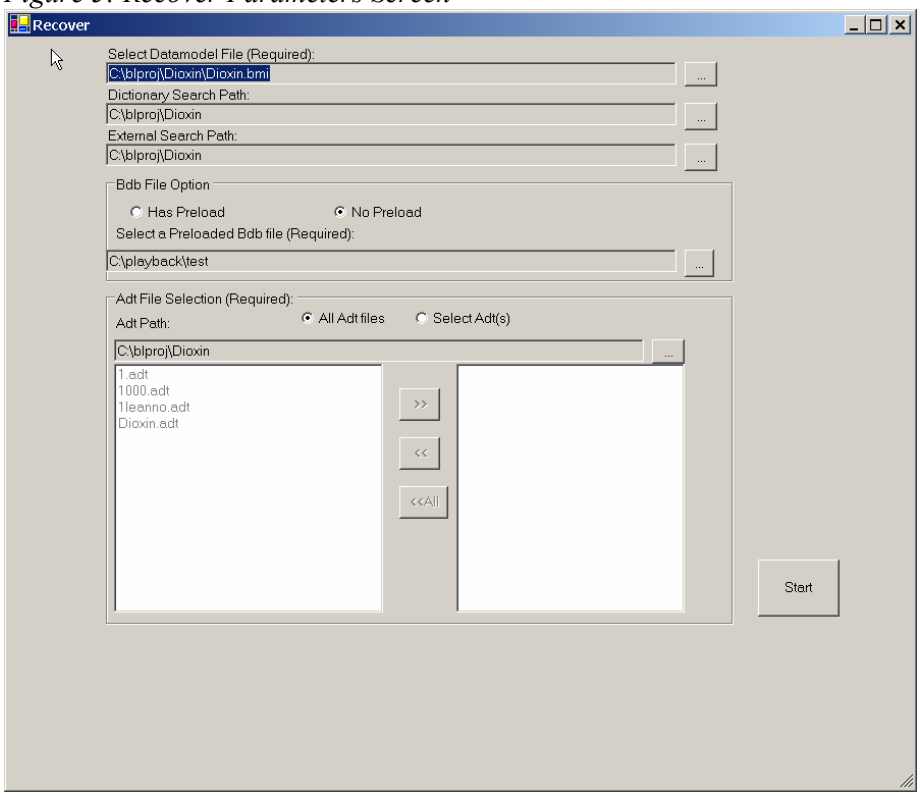
Figure 4: Example of Remarks Window and Its Contents for Displayed Field



3.2. Recover

The Recover component of the Playback system was designed to recover data from audit trail files. This generally would be desired if Blaise data files are lost or corrupted. Figure 5 shows the parameters screen for the Recover component. It is very similar to the Playback parameters screen. The main difference is that the user may choose more than one ADT from which to recover data. This is a much simpler process than that of Playback. The program will go through the selected ADT files and find the final value associated with a "Leave Field" line and save that value to a BDB file.

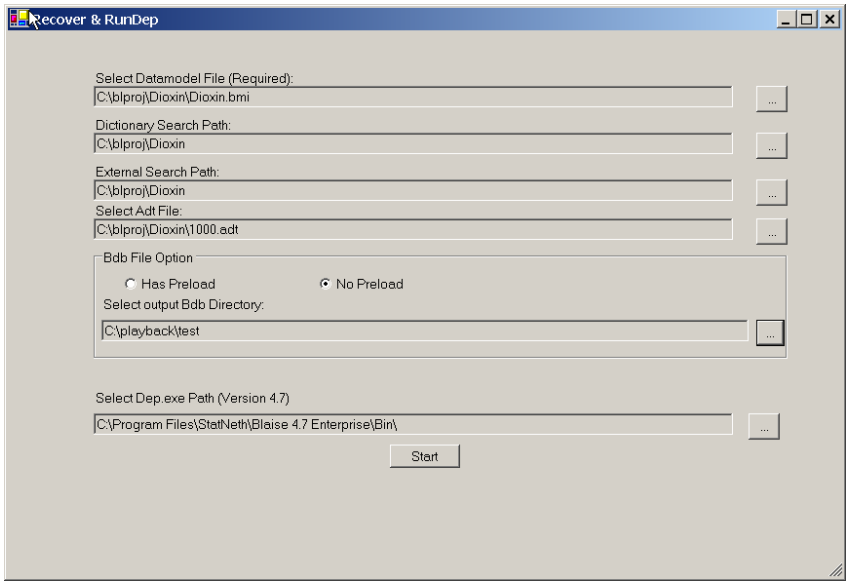
Figure 5: Recover Parameters Screen



3.3. Recover & RunDep

The third component of the Playback system is "Recover and RunDep." It differs from the other two in that it actually runs the Blaise DEP during the recovery process, and allows the user to "walk through the instrument by pressing [Enter] at each field as values are filled in. Figure 6 shows the parameters screen, with similar selections to be made. As with Playback, the user selects one ADT to recover and walk through. The program will first read the data from the ADT file and save final values to the database. The program will then launch the same case in Blaise Data Entry Program. This option is best for recovery of a suspended case from an ADT file and subsequent completion of the file. This may be desirable during the testing phase of development.

Figure 6: Recover and RunDep Parameters Screen



This component functions differently from the Recover component, in which the user can only save data fields. With Recover and RunDep, the program can also save the auxfield data and put those data into a parameter file. These data can then be passed from the parameter file to the DEP. Other parameters could also be written to this file. For example:

```
[DepCmd]
DataModel=T:\groups\TSG\tsprog\sysprog\SHARE\YLiu\NewPlayBack\Joel\Census\main\
        inst.bmi
MetaSearchPath=T:\groups\TSG\tsprog\sysprog\SHARE\YLiu\NewPlayBack\Joel\Census\
        main
ExternalSearchPath=T:\groups\TSG\tsprog\sysprog\SHARE\YLiu\NewPlayBack\Joel\Cens
        us\main
DataFile=T:\groups\TSG\tsprog\sysprog\SHARE\YLiu\NewPlayBack\Joel\Census\orig_bd
        bs\00000037.bdb
Key=00000037
GotoEnd=1
Assign==Front.Start_CP=2;Front.UnitStat_CP=1;Front.Dial_CP=1;Front.Intro_CP=1;Fron
        t.Ready_CP=1;
StartParallel=
```

In this example "GotoEnd=1" instructs the DEP to go to the first unanswered question on the route, and the "Assign" passes in the values of auxfields.

4. Error Handling

It is possible that the selected audit trail files would have errors that would cause the Playback program to stop running. To keep such situations to a minimum, the program checks for the following:

- The "Enter Field" field name must correspond with the field name of next "Leave Field;" this must be true for Playback only; this is hard check, and the program alerts the user that the ADT cannot be processed;
- The "Metafile name" must be the same as the selected datamodel name; this is a hard check, and the program alerts the user that the ADT cannot be processed;
- The "Metafile timestamp" should have the same date as the datamodel file's date. This is a soft check, which gives a warning message; the program will not stop running.

5. Randomization In the Blaise Datamodel

If the Blaise datamodel has a randomization field, and some logic depends on the value of this variable, problems may occur during playback and recovery. This is because Blaise may generate a different value during Playback than it generated during data collection. In this situation, data for these off-route fields may be lost. To avoid this, it is advised to preload randomized values.

6. Implications of Using an Alien Router Call

Alien Routers are external programs that Blaise calls to produce functionality and interface design that can not be achieved easily through Blaise directly. Because they are external programs, the audit trail DLL does not write any thing in the ADT file for the actions used in Alien Router. During playback/recover, keystrokes and mouse actions entered via an alien router call cannot be played or recovered. In order recover those values, the alien router needs to provide its own backup method. The use of such stored data needs to be managed (e.g., as preload) independently of the system described here.

7. Acknowledgements

The authors would like to acknowledge the contributions of Karen Bagwell, Rob Wallace, and Agatha Heesook, of the U.S. Bureau of the Census, and Eduardo Galvan, Sue Ellen Hansen, Joel Harvey, Jason Langfahl, Jason Ostergren, and Jim Hagerman of the University of Michigan. Their input on specifications, testing, and documentation have been invaluable.

8. References

Blaise OnLine Assistant (release 4.7.1000). (2005). Retrieved February 24, 2006 from World Wide Web:http://www.blaise.com/onlinehelp/

Langfahl, J. (2004). Replaying Blaise Audit Trail Files for Data Verification. Paper presented at the 9th International Blaise Users Conference, Gatineau, Québec, September.

Ostergren, J., Ash, R. (2004). Replaying ADK Files for Testing Applications. Paper presented at the 9th International Blaise Users Conference, Gatineau, Québec, September.

Active Management: A New Strategy for Managing Data Collection

Luc Tremblay (Statistics Canada)

1. Introduction

In the past, managing the collection of data at Statistics Canada has traditionally been limited to verifying the response rate of a survey. There has been no process in place to monitor the survey in detail, with the goal of, say, fine-tuning the collection procedures to cut costs or improve response rates.

For example, just knowing the best times to phone a respondent should enable interviewers to make fewer unanswered calls to households. A small adjustment such as altering the time of day in which an interview calls a respondent may reap untold benefits. It may result in major additional savings. It may also help to avoid biases in the results.

During the past year, Statistics Canada has achieved several breakthroughs in this area by implementing a strategy the Agency calls "Active Management". This strategy goes far beyond monitoring response rates.

Active Management is a process that controls the collection of data dynamically from start to finish. On one level, it means that operations managers get regular reports to assess the survey's progress. But it also provides full control over operational parameters that can be changed quickly in the field in response to emerging collection issues.

This paper will review Statistics Canada's approach to the strategy of Active Management. It will examine the different tools Statistics Canada staff have built using Blaise software to support this concept.

2. Active management: Keeping the finger on the pulse

Active Management means that senior managers can now keep their fingers on the pulse of a survey literally from start to finish. It means they can examine production data to identify various parameters which might be adjusted to fine-tune the conduct of a survey.

These parameters include, so far: the best time to call, maximum number of tries, refusal conversion rates per try after initial refusal, and the most efficient number of busy dials to use and their best repetition.

More often than not, collection management tends to begins when a survey starts in the field. However, Active Management starts long before that.

Statistics Canada has been collecting Blaise transaction history (BTH) files since the start of 2001. More than three million BTH records associated with social surveys are now available in a metadata library, which covers a sample size of nearly half a million cases and more than 900 interviewers.

This metadata library has become a tool to help determine best practices for collecting social data. The Operations Research and Development Division of Statistics Canada recently released two studies that have affected the way the Agency conducts social surveys.

The first study focused on the time and effort required by interviewers to reach targeted response rates. An analysis of calling patterns centered on:

- the number of attempts and time to resolve a case
- the effort to convert recalcitrant respondents, and
- the effort associated with hard-to-reach groups

The second study analysed the best time to call a respondent by examining when to make the call and corresponding effort interviewers required to reach targeted response rates at different times of the day.

Among their findings:

- The last 10% of the sample required between 29 and 183 calls (maximum ranges) and took a disproportional amount of interviewer time, that is, 22% for longitudinal surveys and 34% for random digital dialing (RDD) surveys.

- Considerable effort was spent after first definite contact was made, but no data were obtained in 65% of cases. The follow-up effort required six to 11 additional calls, and accounted for 25% of the system time for RDD surveys and 43% for longitudinal surveys.

- There were windows in which attempts to call were more likely to be successful and end in a response. A large percentage of the total interviewing effort analysed was outside the best time to call respondents[3].

- And finally, respondents who are called at their preferred times are more likely to respond.

In addition to improving patterns for calling interviewers, Statistics Canada faced several other challenges. For example, the Agency is becoming concerned about respondent burden resulting from requests for more and more data.

Furthermore, an analysis of BTH data showed that response rates were beginning to decline for a number of surveys and that there were more and more "refusal conversion calls" (calls that were necessary to try to change the mind of respondents who had refused to participate in the survey). Research also indicated that any attempt to contact respondents after two refusal calls was not worth the effort.

BTH research suggested that global response rates were not sufficient in terms of managing different strata within a sample. Consequently, proposals have been made to identify response rates at the stratum level.

3. Results of the analysis

The research and analysis group presented these findings to senior management at Statistics Canada which caused a number of changes to social survey collection:

[3] The effectiveness of a call can be measured by whether it results in a response (complete or partial complete) or a contact which in subsequent calls produces a response. Both measures were used in this analysis. All outcome codes were subdivided into three contact groups – definite contact where information was obtained from having spoken to a live person; possible contact where some information was obtained but not necessarily by live contact (i.e. answering machine); and no contact.

- A maximum of 25 call attempts[4] was placed on longitudinal surveys and 20 for RDD surveys.

- A limit of two "refusal conversion calls" was placed on all cases.

- Given the new limit on the number of call attempts, it was important to ensure that calls were properly distributed. As a result, "time slices" were introduced for a number of social surveys, to more specifically target the best time to call.

- In addition, stratum control was implemented by using different interviewer groups to target specific strata.

4. Technical changes in data collection operations

Implementing the new approach approved by senior Agency managements required a number of technical changes to be introduced through the Blaise software. This section examines these technical changes.

A) Maximum number of attempts and maximum number of refusals

The first technical change was to modify the application to include two counters. Once the counter reached the maximum number of attempts, the case would be finalized with a specific code. If it reached two refusal calls, it would be coded "final refusal and sent to head office.

Since the maximum number of call attempts can vary depending on the type of survey, a variable was created on the sample file that could, if necessary, be modified to adjust the maximum number of attempts.

To allow for exceptions, the number of attempts can be set to a maximum of 999 in the sample file. This way the standard application will not need to be changed, eliminating the need for additional development or testing.

B) Time slice implementation

The second technical change involved the best time to call. The BTH analysis showed that a large number of calls were attempted at a less than optimal time. Consequently new procedures had to be implemented to target the best time to call respondents.

The Survey of Labour and Income Dynamics (SLID) was chosen as a pilot to implement "Time Slices".

Three different Time Slice Sets were created for SLID. The first was for households with at least one elderly person. The second was for people living alone, and the third for all other cases.

The table below represents the distribution of the 25[5] tries[6] by Time Slice Set:

[4] A "call attempt" is an internal counter in the system that represents every time a case has been called, no matter its outcome. It includes busy, no answer, access for verification, etc. Every attempt is considered.

[5] Even if the total number of tries is equal to 25, which is also the maximum number of attempts, the maximum number of tries will not always be met, as it only counts the no answer dials (or the max number of busy dials that result in a no answer dial) and the answering machine. The attempts will count everything.

[6] Try is a technical term to represent a time slice incrementation.

Figure 4.1

Time Slice Set	Definition	8:00h - 17:00h	17:00h - 19:00h	19:00h - 21:00h	Week-Ends 9:00h - 21:00h
TS1	**Elderly**	18	5	2	N/A
TS2	**Single**	2	5	13	5
TS3	**Others**	3	13	9	N/A

This table provides the following information:

- The elderly Time Slice Set puts priority on the day-time calls without differentiating weekend calls.

- The single Time Slice Set has a high priority at night and ensures that at least five calls are made on weekends.

- The "others" Time Slice Set targets the supper hour as the best time to call, with no differences between weekdays and weekends.

At the time of writing this paper, SLID was still in the early stages of collection. As a result, the impact of the new strategy is not yet known. However, a number of initial observations have been made by field staff:

- Day-time calls are becoming more productive as there are fewer calls per contact.

- Since the elderly are being called during the day, there is more time to reach non-elderly people at night, making this Time Slice much more effective.

The major challenge with implementing Time Slices for SLID was the need to plan the interviewer's workload more precisely and identify how many interviewers would be needed, given the new distribution of work against the three time slice sets.

To help the situation, new reports were developed for managers and supervisors to plan collection activities more precisely. Figure 4.2 gives a picture of the existing daybatch report, which is an important report for field staff. This report has become a complement to the new Time Slice report which provides a complete overview of where to put collection efforts.

Figure 4.2

22/02/2006

Composition of the DayBatch
Survey of Labour and Income Dynamics

Group	Quantity	%
Interviewer English	72	50.70%
Interviewer French	17	12.00%
Refusal English	3	2.10%
Refusal French	4	2.80%
Tracing English	13	9.20%
Tracing French	10	7.00%
Senior Interviewer	3	2.10%
Project Manager	20	14.10%
No group assigned	0	0.00%
Total cases	142	100%

The daybatch report drafted for SLID collection provides the number of cases that are eligible for a specific Time Slice definition. Figure 4.3 illustrates this. Even if the daybatch report is not live it is quite useful when combined with the live daybatch report.

Figure 4.3

22/02/2006

Survey of Labour and Income Dynamics
Time Slice's Report
Availability of Cases

	TS1 - Elderly				TS2 - Single					TS3 - Other				Summary				
		Monday to Sunday				Monday to Friday			Weekend		Monday to Sunday				Monday to Friday			Weekend
	Total	Start 17:00	17:00 19:00	19:00 End	Total	Start 17:00	17:00 19:00	19:00 End	Start End	Total	Start 17:00	17:00 19:00	19:00 End	Total	Start 17:00	17:00 19:00	19:00 End	Start End
Panel 4 English	52	43	46	36	33	1	31	30	0	337	15	337	323	422	59	414	389	0
Panel 4 French	3	0	3	3	0	0	0	0	0	2	0	2	2	5	0	5	5	0
Panel 5 English	66	16	59	46	44	3	19	43	0	337	11	337	298	447	30	415	387	0
Panel 5 French	0	0	0	0	1	0	0	1	0	4	0	2	4	5	0	2	5	0
Refusal English	293	293	293	290	56	49	55	56	0	594	567	594	594	943	909	942	940	0
Refusal French	11	11	11	11	2	2	2	2	0	10	9	10	10	23	22	23	23	0
Tracing English	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Tracing French	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Senior Interviewer	89	89	89	87	11	11	11	11	0	82	78	82	82	182	178	182	180	0
Project Manager	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	514	452	501	473	147	66	118	143	0	1366	680	1364	1313	2027	1198	1983	1929	0

Experience to date has shown that the following new features would make the tool even more effective:

- Since the time slice report is created from the .tdb (textual representation of the daybatch) file, the information about the time slices is not updated until a daybatch is redone, even if the .tdb is refreshed through CATI Management. The short-term plans for the Time Slice report is to run it using the BTH file instead. This would provide an instant picture of what is going on.

- Another enhancement would be to have the number of tries left per case, rather that just the actual number of cases available in that Time Slice. This would give longer term information about the workload, and help managers plan the workload more efficiently.

- The working arrangements of interviewers as specified by their contract with Statistics Canada also pose a challenge. Sometimes, if a shift goes particularly well, it empties the next shift's available cases. The work contract of our interviewers stipulates that shifts have to be modified 24-hours in advance before it can be changed. Consequently, it was necessary to introduce the concept of a "buffer".

  The buffer is an adaptation parameter. As an example, please refer to Figure 4.1 and the time slice set 1, (8:00h - 17:00h). Even if the maximum number of tries is 18, the time slice will originally be set at 15, leaving a buffer of 3 tries. When a shift runs out of cases, the number of tries can be increased (to 18 in this case). The daybatch is then redone and the interviewers are advised of the changes in tomorrow's shifts. Consequently enough work becomes available for the current shift and enough time is now allowed to change the shift for the next day.

C) Stratum control

Good response rates are more and more difficult to obtain and it is important to ensure that all strata of any given survey are publishable. For that reason, ensuring that all strata's minimum requirements are met is very important.

The first survey which incorporated the use of stratum control was the Follow-up of Graduates Survey (FOG). Four different levels of education were examined: college degrees, university undergraduates, master's degrees and doctorates. However, the survey managers were also interested in the distribution among the 10 Canadian provinces and three territories. In other words, there were 48 different strata to accommodate within the stratum control strategy.

Statistics Canada surveys need specific interviewer groups for workload assignments and as a result, 99 Z[7] groups were created to handle both English and French speaking respondents.

Initially, all interviewers were assigned to all groups, provided that they met the language requirements. All groups were assigned as secondary groups, so that they had an equal chance of being selected. The primary group of interviewers was assigned to a group that wasn't used in this survey.

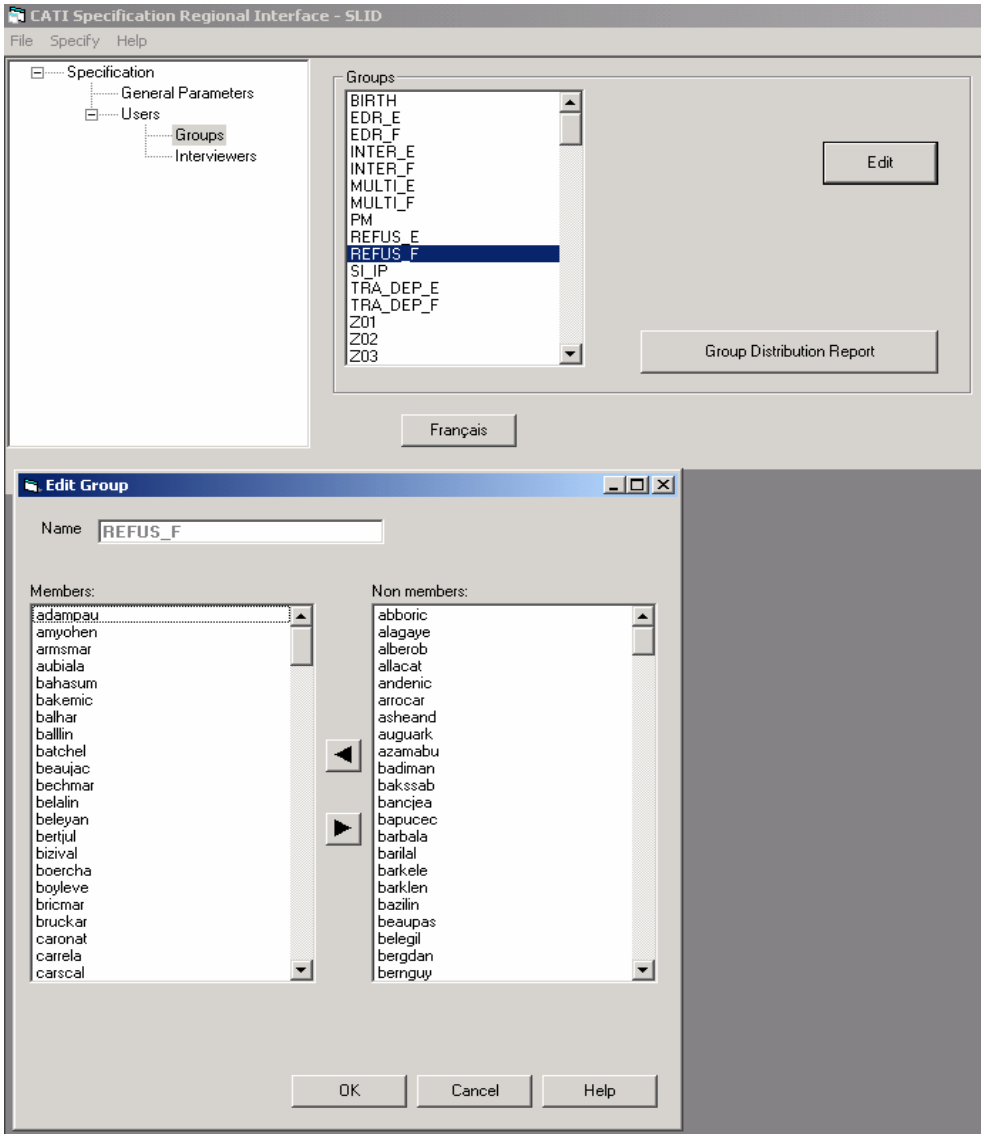
By consulting the production report each day, project managers in Statistics Canada's various Regional Offices were able to determine if any stratum was falling behind. If that was the case, the associated group for that particular stratum would be assigned as the primary group for some interviewers, so that particular stratum was given priority. At the same time, appointments for other groups (or stratum) were respected.

The extensive manipulation of groups meant that Regional Offices had to rethink their methods and process to handle the increased volume of changes. In its original design, the Generic Interface for Regional Offices (GIRO) permitted management of all groups and users from a central location. Consequently, accounts modifications were made across all surveys. The original idea was to allow an interviewer proficient at converting refusals to be in the same function for all surveys.

[7] Z was choosen for these groups so that they would appear at the bottom of the list when selecting whom to send the case to, as there were few surveys using these groups, effort was minimized by not having to scroll through every time we needed a regular popular group.

The agency is moving away from this approach. Since the introduction of Z groups, the definition of a specific group can vary from survey to survey. The use of the Blaise component pack permits managers to create survey specific levels of users and group managers. Figure 4.4 is a print screen of the Specification Regional Interface (SRI).

Figure 4.4



A major concern introducing this new approach for FOG was the increased amount of work required to manage the new approach. However, a training program was instituted for field managers and the results to data are impressive.

5. Active Management: What's on the horizon?

So, what is on the horizon at Statistics Canada in terms of Active Management?

Several initiatives are under way which will broaden the current scope of Active Management.

The first initiative is targeted for implementation in the spring of 2006.

A) Transfer of cases from One Regional Office to Another

Statistics Canada has 6 Regional Offices (CATI Sites). Sample are being sent to each individual site separately and often there is a need to transfer capacity from one site to another during the course of a survey. This fundamentally does not exist today.

A new project has been initiated called “RO to RO transfer”. It will permit the transfer of cases from one regional office to another – even after the case is started, without loosing any information.

The challenge associated with creating this functionality are many, given the complex data models used at Statistics Canada. A datamodel is often composed of multiple databases : for notes and remarks, for the tracing component that permits an interactive search for a respondent, for the customized record of calls, and sometimes for supplements or components of a survey that are rostered for everyone in the household.

Once a case is started, data reside in each of those associated databases. The object was to create a generic method for transferring cases across Regional Offices. Which would allow further flexibility in managing capacity, language choice and technical infrastructure.

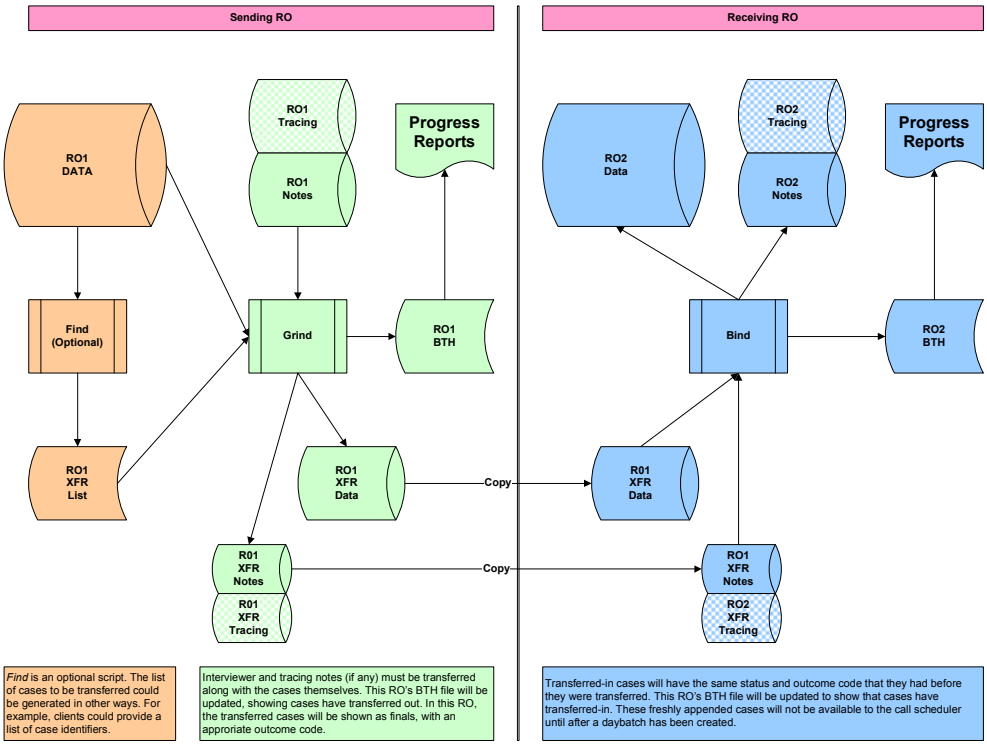
Three pieces of software are required for the “RO to RO data transfer” functionality. They are called **Find**, **Grind** and **Bind.** Each has a different function. The last two are generic, and take almost no time to implement in a survey. Once they have been tested in the field, they will be implemented on all surveys. Figure 5.1 illustrates the flow between the different software.

Find: This is the most non-standard piece. *Find* simply selects and creates a list of all cases to be transferred, regardless of the reason. There are various ways of identifying these elements, by the client, or language, stratum, sex, and so on therefore the function cannot be standardized.

Grind: This software extracts the cases from the Regional Office wishing to transfer them to another site**.** It looks at the list of cases generated by *Find*. Grind goes through all associated databases and creates a copy of the data from the databases. It then changes the outcome code of the case to "transferred to other RO". Finally, it updates the BTH files for the reports, and flags the case as to output status, so that overnight automated jobs don't extract the case and send it to head office.

Bind: This software loads cases that were extracted into the receiving Regional Office. Very simply, it takes the files created by *Grind* and attaches them to the associated Blaise databases. It changes a flag signalling that cases have been transferred, and updates the Regional Office code as well as the BTH files to reflect changes in the reports.

Figure 5.1



FIND ***GRIND*** ***BIND***

B) Preferred Time to Call a Respondent

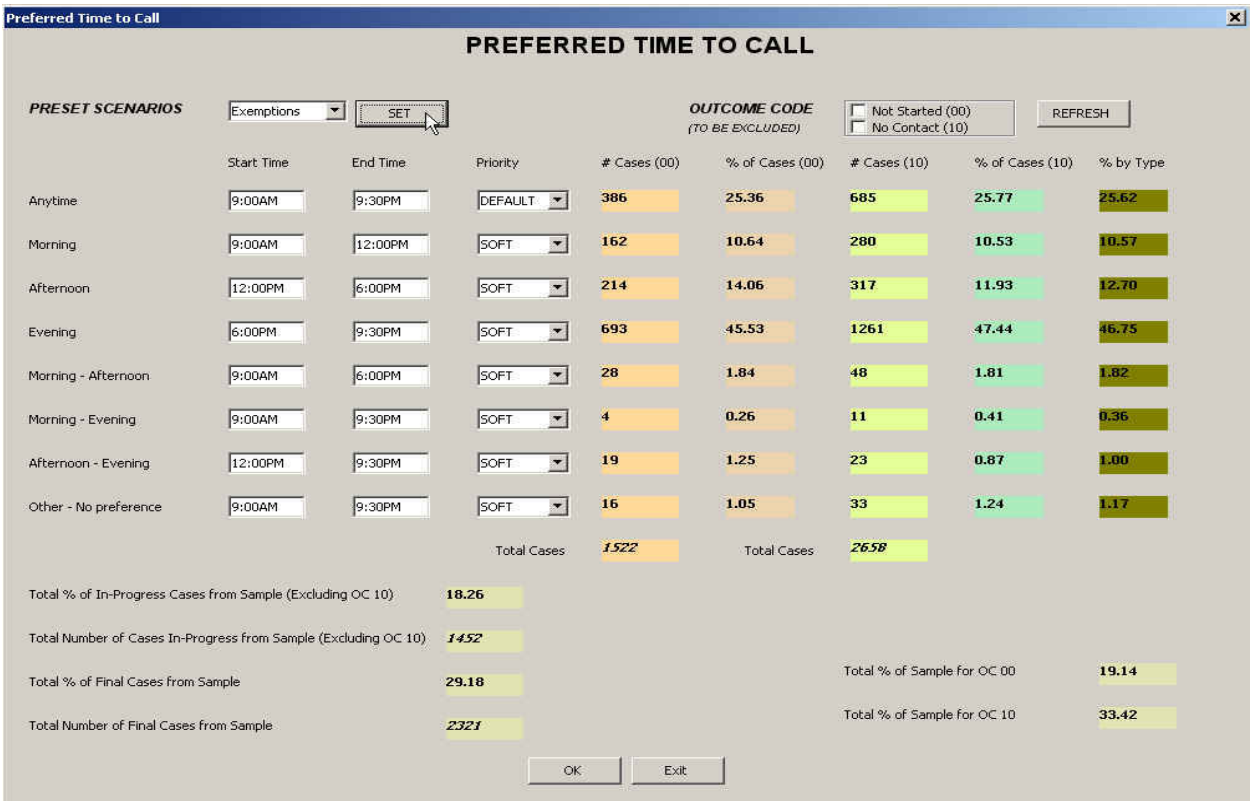
In order to provide maximum flexibility, especially for surveys with short collection windows and large volumes, a new tool has been created to allow managers to set priorities in the daybatch, based on individual respondents previously identified best time to call.

This is a very graphical tool, with preset scenarios and outcomes. See Figure 5.2. below.

The key benefit of this tool is its ability to prioritize cases without the removal of other non-prioritized cases. Therefore, if prioritized cases are exhausted, there are still cases left to keep the interviewers busy.

This tool is currently being presented to clients and feedback has been very positive.

Figure 5.2



6. Conclusion

The tools described in this paper have greatly improved the degree of control managers at Statistics Canada have over surveys. Previously, when a survey moved into the field, it was hard to change the set course of action. There were no tools or facilities available to track the survey at a detailed level to improve the day to day efficiency of the collection operation.

Today, however, new tools have allowed Agency managers to determine how to concentrate their efforts to improve collection practices. This 'real-time' monitoring might simply entail changing the times that certain respondents are called to complete a questionnaire.

Active Management is indeed a key way of "keeping the finger on the pulse" of collection activity at Statistics Canada.

8. References

[1] Lee, J. (2005): Analysis of Best Time to Call Across 30 Survey Cycles for Longitudinal and RDD Surveys, Statistics Canada, Ottawa, CA

Work Flow for the Weighting of the German Microcensus Data Using Blaise Bascula

Kirsten Iversen (Federal Statistical Office, Germany)

1. Introduction

The German Microcensus originally focussed on a specified reference week in spring. In January 2005 it started as a continuous survey in order to obtain estimates of quarterly and annual means. The data are collected locally, i.e. the Statistical Offices of the 16 federal Länder in Germany do the field work. Participation in the survey is compulsory.

For every quarter the data are weighted in two steps: Firstly, there is compensation that adjusts the known non-responses. In the second step, the sample distribution of some auxiliary variables (for example age, sex and nationality) is adjusted to the known population totals. The intention of this bounded second-step weighting is to reduce the bias of estimates due to unrecognised non-responses. The variance for variables which are highly correlated to the auxiliary variables is also minimised. Implementation of this two-phase weighting in Blaise Bascula is presented in Chapter 3.1.

Once a year the data for the four quarters are combined for deep regionalised weighting. The weighting program used for this is the subject of section 3.2.

The conclusion is formed by a brief overview of the size of the files and the running times of the programs (Chapter 4). To start with, the content of the processed data will now be briefly explained.

2. The Data

The data of the German Microcensus are collected continually, spread equally over the whole year. The questionnaire used in both the CAPI survey and the paper-and-pencil version comprises roughly 190 questions on the topics of personal data, working life, education, income and housing. In addition, further variables are generated from the information collected which are required as characteristics for the weighting.

An abridged dataset is extracted for the weighting. It covers only 29 variables which are needed directly for compensation and weighting, such as period under report, Federal Land, administrative district, household characteristics, nationality, sex, age and individual characteristics on employment. These datasets, comprising 60 variable labels, form the input file for the weighting in the shape of an ASCII file “MZHR1.asc”.

The “MZHR1” file contains in each case the observations of one quarter. All four quarters in a year are separately compensated and weighted, and are only placed together in the deep regionalised annual weighting. The annual sample covers roughly 1% of the population, that is, depending on the Federal Land, between 670 and 15,000 individuals per quarter (¼ %). This is a proportionally-stratified cluster sample.

Apart from the datasets of the quarter, the bounded weighting also includes benchmark values relating to the population structure. These originate from the

current population adjustment of the population statistics which are based on the 1987 census (West) and on a 1990 register excerpt (East). Figures for the foreign population are added from the Central Foreigners' Register. In addition to the total population size of the Federal Land in question in the respective quarter, figures are also shown separately for the individual administrative districts on sex and nationality. The combinations of these characteristics which arise are shown in a table which is contained in a .pop file, and are incorporated into the weighting as benchmarks. For the deep regionalised annual weighting, figures are additionally included at the level of the regional sub-groups which also come from the current population adjustment.

3. Running of the program

The total weighting consists of five programs, as well as of four batch processes. These are the batch processes of the compensation and actual weighting; they are run in Bascula. The programs, by contrast, run under Blaise and prepare the data for the batch processes: They separate the respondents by individual characteristics, re-sort the datasets as required and then add the weighting factors back to the datasets throughout the whole file.

First of all, the operation of the quarterly weighting is explained (programs 1 to 3, as well as batches 1 to 3), followed by a brief description of the conversion of the annual weighting. A dataflow plan of the whole program run can be found in the Annex (A1).

3.1. Steps in the quarterly weighting

The extraction of the abridged datasets, which now only contain the compensation and weighting-relevant variables, takes place prior to weighting. The "MZHR1.asc" file, which contains the abridged datasets of a quarter, is the input material on which the entire quarterly weighting is built.

Program 1:
Program 1 is a preliminary program which creates further characteristics from the variables of MZHR1, specifically for compensation and weighting, such as nationality (German/EU25/non-EU), a breakdown into age classes, household sizes, questioning at main or secondary residence. Furthermore, three specialised files are created from the input file: One covers all respondents[8], another file contains only respondents from community households (CHH), while the third file holds only the household reference persons of the private households (PHH). The datasets are weighted to cover additional fields for the compensation factors and for the subsequently-calculated weighting factor. For the household reference person in a private household, and for the respondent in a community household unit, the value one is entered in the datasets to provide an inclusion weight for compensation.

After the sub-division, an examination takes place of whether the file of the community households contains at least ten datasets. If this is not the case, no further benchmark values are calculated for the community households, and the Batch 2 process (compensation for non-respondent community household units) is skipped.

[8] "Respondents" always refers to the persons who actually answered, in other words all persons of a sample not including the non-response cases, the share of which is as a rule under 3%.

Program 1 also calculates the reference values to compensate for non-respondent households. To this end, for the individual characteristic combinations of the compensation characteristics, frequency counts are carried out in the input material (in other words with respondent and non-respondent persons). The reference values all have at least the value zero, and are output in table form in a .pop file, the structure of which is specified in a .blg file. Separate benchmark value files are created in each case for private households and community household units. All regional adjustment layer numbers in a Federal Land which are not occupied, as well as characteristics represented neither with respondents nor with non-respondents, are directly assigned the value zero.

After these preparatory steps, the compensation factors for non-respondent households can now be calculated (non-response weights), via which the net sample is weighted to the gross sample. The compensation for private households and community household units is carried out in separate batch processes.

Batch 1:
The Batch 1 process calculates compensation factors for the household reference persons of private households with which a shortfall of other private households is to be compensated for and the resultant distortion of the results reduced. The administrative district, as well as the regional adaptation layer and sub-group, the rotation quarter, the label "new buildings layer", the household size, the nationality (German/not German), main or secondary residence, the age group (under 60/60 years and older) and sex, are used as compensation characteristics. The combinations of the administrative district and the regional adjustment layer with the further variables are used as a compensation model, the reference values are available in the .pop file created in program 1.

Linear weighting with constant is used as a weighting procedure. The inclusion weight of the value of one entered in program 1 is used as input weight. There must be at least ten observations in each combination of the compensation model. If this is not the case, the model is reduced to a higher-order combination of the characteristics (the strongest differentiating factor of a combination is removed). The lower threshold of the correction weights and of the final weights is 0.001, and 3 is stated as the upper limit. The maximum number of iterations is 10 (more iterations are not possible in batch processes). The permissible inconsistency of the benchmark values should not exceed 1, while the accepted relative deviation of the weighted values from the benchmark values is limited to 0.001.

An example syntax for the batch processes can be found in the Annex (A2).

Batch 2:
The Batch 2 process calculates the factors to compensate for non-respondent households in community households. It is only implemented if the file for community households contains at least ten datasets. Where there are fewer observations, the compensation of non-respondent households in community households is not carried out. The compensation model is limited to the administrative district. Reference values are available in the shape of the .pop file created in program 1.

The post-stratification method with constant is used for weighting, the input weight already having been set in program 1. The minimum cell occupation is ten datasets in this case too. If fewer observations are available in an administrative district, the model is reduced to the constant.

Each batch process outputs a protocol file which also documents, in addition to the (perhaps abridged) model used the adjustments in the individual cells, as well as descriptive statistics on the compensation factors.

Once the batch processes are completed, for the household reference persons in private households, and the persons in community households, in each case a file is now available in which the datasets are weighted by the respective compensation factor. These files are input to program 2.

Program 2:
The second program now allots to all respondent persons the respective compensation factors. Here, in addition to the household reference person in private households, all further household members are also allotted the factor of the reference person calculated in Batch 1.

The number of respondent persons (n) is then calculated. The number of the population total (N) is input as the target value. The compensation factor weighted with the quotients from these two figures (N/n) is used as inclusion weight in the weighting.

Like Program 1, Program 2 also outputs a file with all respondents. The datasets are now supplemented by the compensation factors and the inclusion weights for the weighting. In a copy of this file, the datasets are sorted by household number since these are then used as a cluster code. This file is input directly into the weighting in the Batch 3 process.

Batch 3:
This process calculates the standard weighting factors in the quarter. Administrative district, regional adaptation layer, sex, nationality (German/not German and German/Turkish/EU25/not EU25), an age classification and a label for employees of the Federal Armed Forces, police or at the Federal Border Guard serve as weighting characteristics. The household number is used as a cluster code so that all persons in a private household receive the same weighting factor. The weighting model consists of various combinations of the characteristics. Figures from the current population adjustment are included as reference values which are already available prior to the program launch in table form contained in a .pop file.

The weighting is carried out as in Batch 1 by the linearconsistent procedure with constant. The inclusion weight was calculated in Program 2. The minimum cell occupation is set at 10 (with the exception of one Federal Land), while with fewer observations, the model reduces to a higher-order combination of the characteristics. Once more, the lower threshold for the correction weight is set at 0.001, and the upper limit is 5. Once again, a maximum of 10 iterations should be carried out. The permissible inconsistency of the reference values is 50, while the relative deviation of the weighted values from the reference values should not exceed 0.001.

This batch process also outputs a protocol file which documents the (possibly abridged) model used, the adjustments to the individual combinations of characteristics, as well as descriptive statistics on the correction and final weight.

Once the Batch 3 process has been completed, a file is available containing all respondent persons which is sorted by household numbers and the datasets of which were weighted to include the weighting factor. This file is input into Program 3.

Program 3:
Program 3 calculates the standard weighting factors "year" as ¼ of the standard weighting factors "quarter" which were calculated in Batch 3. Then, both standard weighting factors are applied to the respective datasets in the unsorted file of all respondents (which was not used in Batch 3). Additionally, the compensation factors are attached to the datasets as 7-digit string variables and the standard weighting factors as 4-digit string variables, in each case with no comma.

The data are then output once as an ASCII file and once as a Blaise file. The Blaise file is incorporated into the continued statistical quarter processing. The ASCII file with the name MZHR6-Q*x* is retained for each quarter *x*; all four quarter files are then input together to the deep regionalised annual weighting. The quarterly weighting is completed at this juncture.

3.2. Weighting for deep regionalised annual results

In schematic terms, the annual weighting directly follows the quarterly weighting: The four quarter files MZHR6-Q1 to -Q4 are used as input material. Furthermore, population sizes per regional sub-group by sex and nationality are input from the current population adjustment as benchmark values; these are available in table form as a .pop file.

Since no compensation is required in the annual weighting, this part of the program run only consists of one batch process for weighting and two contiguous programs.

Program 4:
Program 4 first of all combines all four quarter files MZHR6-Q1 to -Q4, and then creates a file of all respondents of a year. Furthermore, a file is created for the Batch 4 process which only contains persons in private households and community households who were asked at their main residence. These are sorted by their household number, which serves as a cluster code in the weighting.

The program (as in Program 1 for the compensation) also calculates further reference values. For this, the standard weighting factors of the individual combinations of the characteristics are added up and output as a table in a .pop file.

Batch 4:
The Batch 4 process calculates the weighting factors for deep regionalised annual results. Administrative district, regional adjustment layer and sub-group, as well as sex, nationality (German/not German, as well as German/Turkish/EU25/not EU25), an age classification, a label of employees of the Federal Armed Forces, police or the Federal Border Guard and the employment status, serve as characteristics. The household number is used as a cluster code once again. The weighting model emerges from the combinations of the characteristics. In addition to the benchmark values calculated in Program 4, the population sizes per regional sub-group from the current population adjustment are also input into the weighting. They are also available in table form as a .pop file.

For weighting, the linearconsistent method with constant is used. The inclusion weight is ¼ of that in Batch 3 (quarterly weighting), i.e. ¼*(*N*/*n*)*compensation factor. Each cell of the model is to hold at least 10 observations. If this is not the case, the model is reduced to a higher-order combination of the characteristics. The interval for the correction weights ranges from 0.001 to 5, while the maximum number of iterations is once again 10. A maximum value of 50 is permitted for the inconsistency of the benchmark values, and the relative deviation of the weighted values from the benchmark values should not exceed 0.001.

To document the procedure, the batch process outputs a protocol file with the (possibly abridged) model used, the adjustments to the individual combinations of characteristics, as well as descriptive statistics on the correction and final weight.

Once Batch 4 has run, a file is created containing all persons who were asked at their main residence. The datasets now also contain the weighting factor for deep regionalised evaluations.

Program 5:
In Program 5, the factors of the deep regionalised annual weighting are applied to all respondents at their main and secondary residence. The weighting factors are once more added to the datasets as 4-digit string variables. Finally, an ASCII file "MZHR7.asc" including all respondents (i.e. private households and community household units, as well as main residence and secondary residence) is output, the datasets of which contain all compensation and weighting factors, both for quarterly and for annual results. The annual weighting is thus completed.

4. File sizes and running times

The size of the MZHR1.asc input files varies greatly from one Federal Land to another since the sample size, and hence also the number of datasets, fluctuates with the number of inhabitants. In Federal Länder with a high population size and roughly 15,000 datasets per quarter, the input file quickly reaches a size of 2,000 KB. With smaller Federal Länder comprising 670 datasets, by contrast, the file size of the input material is only 80 KB.

The benchmark values from the current population adjustment are relatively low, at 1 KB for the total population size, and 5 KB for the .pop file with the tables of the individual combinations of characteristics at the level of the regional adjustment layers. Depending on the size of the Federal Land, and hence on the number of regional adjustment layers, the tables contain 50 to 82 individual values in the .pop file.

The running time of the programs depends on the size of the input files: The quarterly weighting of a small Federal Land only takes one minute, whilst with a Federal Land that has many inhabitants, the running time is roughly eight minutes. The most time is taken in this case by the benchmark value and auxiliary variable calculation.

The input files for the deep regionalised annual weighting MZHR6-Q*x*.asc have a size of roughly 400 KB each with small Federal Länder, and 10,000 KB with high-population Länder. Depending on the number of regional sub-groups in a Federal Land, up to 54 figures are input as benchmark values. The program needs roughly twice as long to run as the quarterly weighting: Almost two minutes for smaller Länder and roughly 15 minutes for a large Federal Land.

5. Abbreviations

CHH	community household
N	number of the population total
n	number of respondent persons
PHH	private households
x	quarter number

6. References

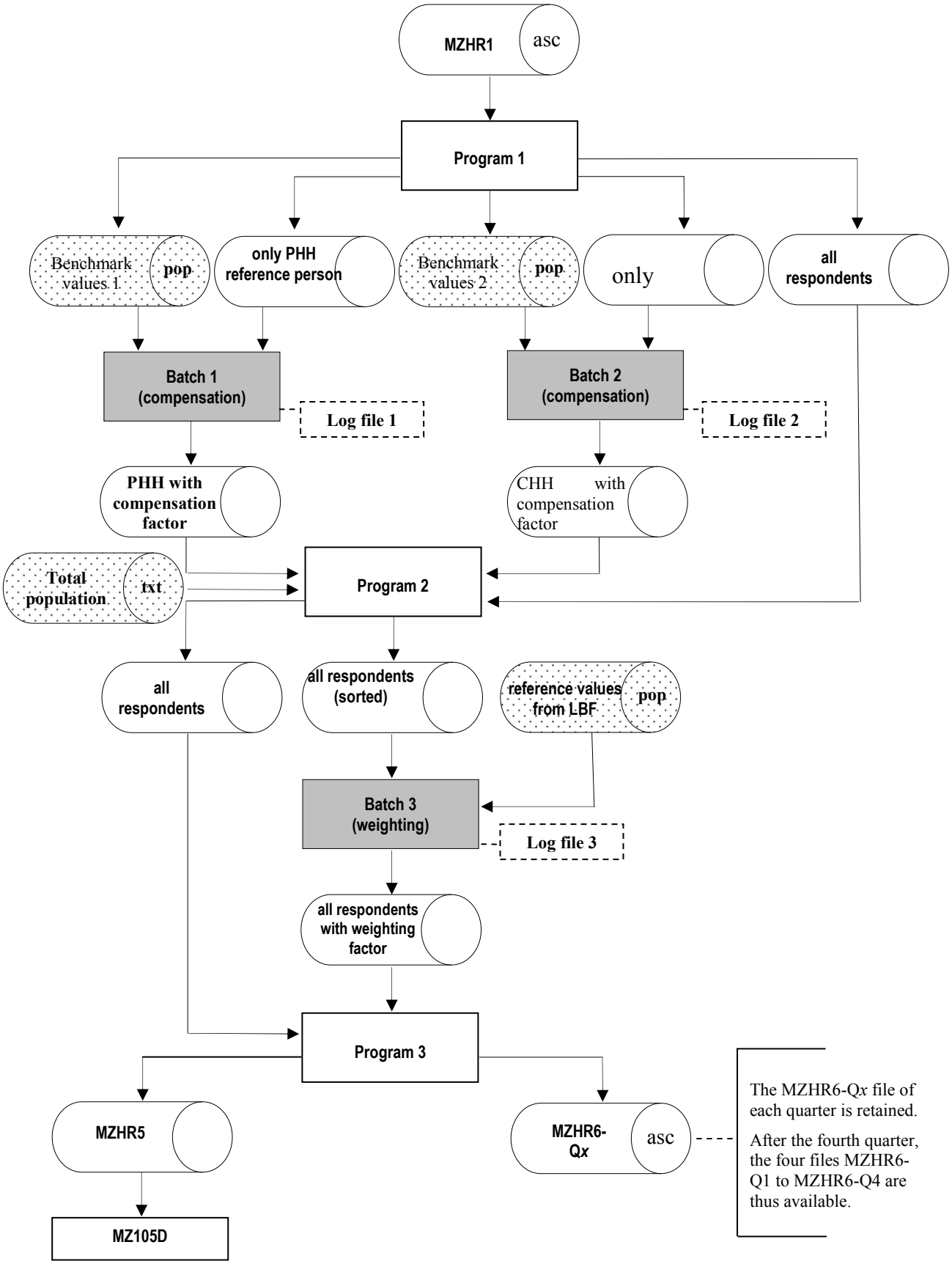
Afentakis, A. (2005): Hochrechnung der Mikrozensusergebnisse ab 2005 – Fachkonzept für die DV-technische Umsetzung. Statistisches Bundesamt Deutschland, Wiesbaden.

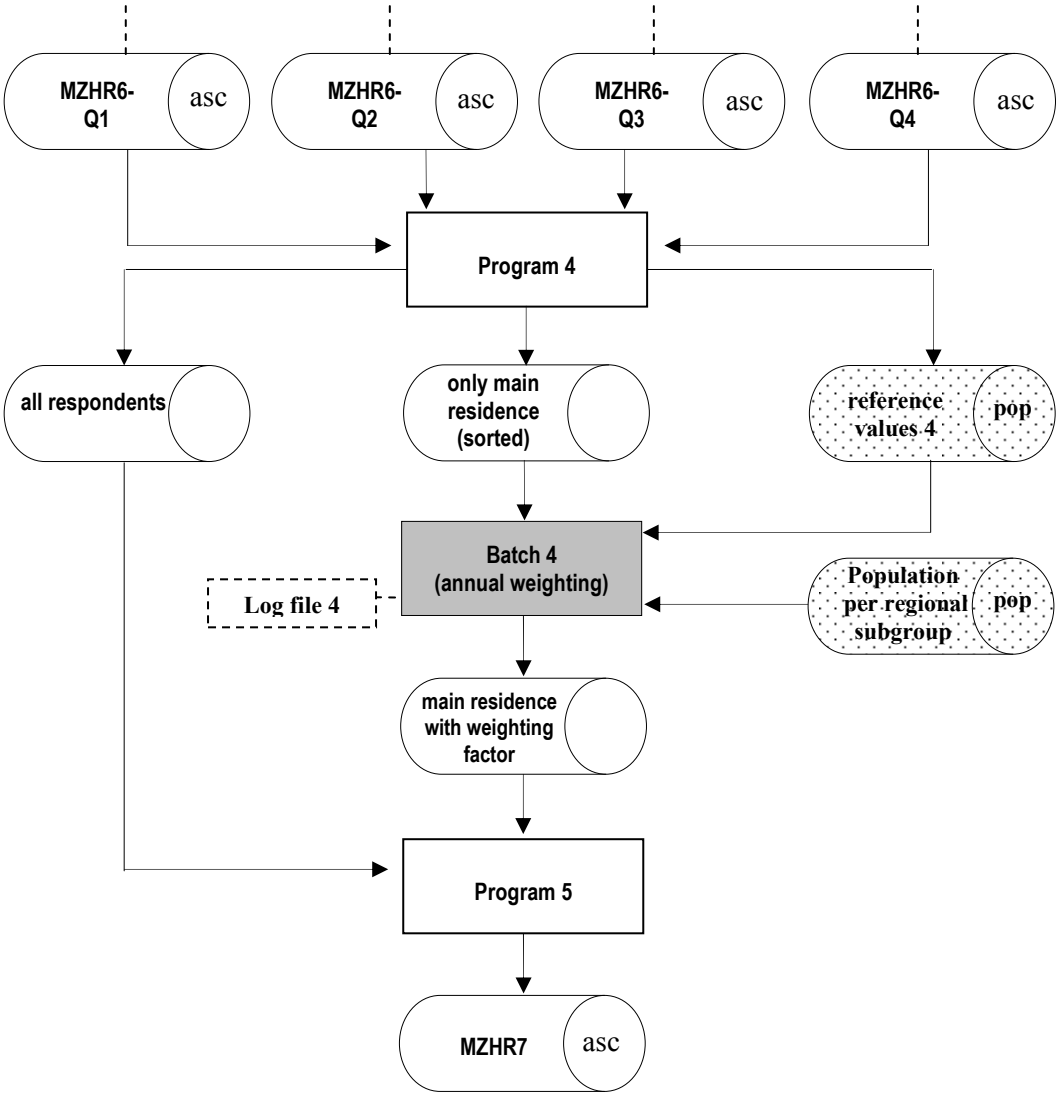
Afentakis, A. & Bihler, W. (2005): Das Hochrechnungsverfahren beim unterjährigen Mikrozensus ab 2005, Wirtschaft und Statistik 10/2005, Statistisches Bundesamt Deutschland, Wiesbaden.

Nieuwenbroek, N. & Boonstra, H.J. (2001): Bascula 4.0 Reference Manual. Statistics Netherlands, Heerlen.

Annex

A1: Data flow plan




MZHR6-Q1
asc
MZHR6-Q2
asc
MZHR6-Q3
asc
MZHR6-Q4
asc
Program 4
all respondents
only main residence (sorted)
reference values 4
pop
Batch 4 (annual weighting)
Log file 4
Population per regional subgroup
pop
main residence with weighting factor
Program 5
MZHR7
asc

A2: Example syntax for the batch processes

```
PROCESS Weighting 'ProcessName' [9]

USES
  MetaID 'MetaFileName' [9]

INPUTFILE SampleFileID: MetaID('MZHR4C',FileType) [9]

AUXFIELDS
  Res: INTEGER

WEIGHT WeightID (SampleFileID)
  VARIABLEFUNCTIONS
    STANDARD = EF5, EF32, EF23, EF100, EF101, EF102, EF103, EF104,
               EF106
    CLUSTERCODE = EF105        {does not apply to compensation (Batches 1+2)}
    INCLUSION = EF122
    CORRECTION = EF119
    OUTPUT = EF120
  ENDVARIABLEFUNCTIONS
  POPULATIONTABLES             {vary by model}
    POPULATIONTABLE
      TABLE = EF106
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
    POPULATIONTABLE
      TABLE = EF103 * EF104
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
    POPULATIONTABLE
      TABLE = EF23 * EF100 * EF101
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
    POPULATIONTABLE
      TABLE = EF23 * EF102
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
    POPULATIONTABLE
      TABLE = EF5 * EF23 * EF100
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
    POPULATIONTABLE
      TABLE = EF13
      POPULATIONFILE = 'MZHR3B.pop' [9]
    ENDPOPULATIONTABLE
     OUTPOPULATIONFILE = 'MZHR3B.pop' [9]
  ENDPOPULATIONTABLES
  WEIGHTSETTINGS
    MODELTERMS = EF106
                 EF103 * EF104,
                 EF23 * EF100 * EF101,
                 EF23 * EF102,
                 EF5 * EF23 * EF100,
                 EF13
```

[9] Complete file path

```
    MINIMALCELLCOUNT = 10
    CONSTANT = YES
    WEIGHTINGMETHOD = LINEARCONSISTENT     {in Batches 1+2: LINEAR}
    BOUNDING
      LOWERBOUND = 0.001
      UPPERBOUND = 5                       {in Batches 1+2: UPPERBOUND = 3}
      MAXSTEP = 10      {A maximum of 10 iterations are possible in the batch process.}
    ENDBOUNDING
    WEIGHTFILE = 'Batch3.wga' 9
  ENDWEIGHTSETTINGS
  SETTINGS
    DIFFTOLERANCE = 50                     {in Batches 1+2: DIFFTOLERANCE = 1}
    ESTCHECKTOLERANCE = 0.001
  ENDSETTINGS
ENDWEIGHT

MANIPULATE
  Res := WeightID
```

Posters

User-friendly Web Surveys
Hayo Bethlehem (The Netherlands)

Using the Blaise Component Pack in the .NET environment
Rob Groeneveld
(Statistics Netherlands)

A Calculator DLL
Roberto Picha
(U.S. Census Bureau, USA)

Programming for Navigation in the Blaise System
Mark Pierzchala (Mathematica Policy Research, USA.)

The Selection of Strata in Nonresponse Adjustment
Barry Schouten
(Statistics Netherlands)

CAPI System at Central Statistical Bureau of Latvia
Norberts Talers & Palvels Onutrijevs
(Central Statistical Bureau of Latvia)

Multiple Researchers working at the Blaise Benchmark Services for the Disabled Act (WVG)
Carlo Vreugde & Mark Gremmen
(SGBO, The Netherlands)

User-friendly Web Surveys

Hayo Bethlehem (The Netherlands)

1. Introduction

Web surveys become an increasingly important tool for collecting data in our digital society. Unfortunately, many people dislike filling out forms on the web. To avoid high nonresponse rates, it would help to make questionnaire forms as attractive and user-friendly as possible.

This poster looks at the issue questionnaire form design from the point of view of a web designer, and not from the point of view a survey researcher that needs an electronic data collection instrument.

The poster focuses on two aspects. The first one is the 'language' used to create forms on the web. Web standards play an important role in this. The use of a framework based on XHTML and CSS is advocated, making it easier to build and to maintain effective web applications.

The second aspect of web forms is their visual design. Wire frames are suggested as tool that may assist in this. Also some other usability tips are given.

2. Use of XHTML for form design

Traditionally, web pages are made up using HTML. However, this mark-up language has its limitations from the point of view of standardisation, maintainability, etc

A much better suited mark-up language is XHTML. This is a very strict form of HTML. Actually, it is an XML application. This means that by using XHTML as a mark-up standard, a solid base for web integration is obtained, and a correct technical structure of the data can be guaranteed. For instance, XHTML may only use the UTF-8 character set as per specification. So all data input through your forms will be UTF-8. The strict mark-up requirements of XHTML make sure you have a technical standard to which your forms must comply.

Currently, Blaise uses XSL in combination with HTML to create web questionnaires. For the future it might advisable the consider using XHTML.

Figure 1 shows an example of a form defined in XHTML. Note that the visual appearance of this form is displayed in figure 2. It is clear from figure 2 that XHTML is an XML application. For example, for every opening tag there is an accompanying closing tag.

Figure 1. A form in XHTM

```
<form action="/contact/" method="post">
   <fieldset>
         <label for="name" >Uw naam:</label>
         <input type="text" name="name" id="name" value="" />

         <label for="email" >Uw e-mailadres:</label>
         <input type="text" name="email" id="email" value="" />


         <label for="place" >Uw woonplaats:</label>
         <input type="text" name="place" id="place" value="" />

         <label for="phone" >Telefoonnummer:</label>
         <input type="text" name="phone" id="phone" value="" />

         <label for="recipient">Ter attentie van:</label>
         <select name="recipient" id="recipient">


            <option value="nieuws">Nieuwsredactie</option>
            <option value="tvkrant">TV Krant</option>
            <option value="bestuur">Bestuur</option>
            <option value="sport">Sportredactie</option>
            <option value="web">Webmaster</option>
         </select>


         <label for="message" >Uw bericht:</label>
         <textarea id="message" name="message" rows="5"
cols="45"></textarea>
         <input type="submit" class="button" value="Verstuur" />
   </fieldset>
   </form>
```

The various ingredients of this form are discussed in some more detailed below.

<form>
The form element creates a form for user input. A form can contain text fields, checkboxes, radio-buttons and more. Forms are used to pass user-data to a specified URL.

<fieldset>
This can be used to group data in a form, making differences more distinct. Visually, it draws a border around the objects inside. A fieldset can be styled in lots of ways using CSS (Cascading Style Sheets). The <legend> tag can be used to describe the contents of a fieldset, but it is not required.

<label>
Defines a label to a control. If you click the text within the label element, it is supposed to toggle the control. The "for" attribute binds a label to another element. Set the value of the "for" attribute equal to the value of the "id" attribute of the related element.

One of the most common ways of navigating through a form is using the tab key. A person fills in one field, hits the tab key, fills in the next field, and so on, until the end of the form is reached. For visual users, this is an easy thing, because the labels are placed in such a way that they appear to be linked to their corresponding controls. For a blind person using a screen reader, however, visual layout is not going to be of much help. That is where the “for” attribute comes in, to bind the label to its control.

<input>
This tag defines an input field where the user can enter data.

<select>
The select element creates a drop-down list, <option> is a sub item of a select.

<textarea>
Defines a text-area (a multi-line text input control). A user can write text in the text-area. In a text-area you can write an unlimited number of characters. The default font in the text-area is fixed pitch.

<input type="submit">
The submit button.

Using a mark-up language based on XML has a number of advantages. Here are a number of them that can be found in the literature:

- Ability to support internationalisation in a self-describing way that makes confusion over coding options unlikely.
- Early detection of encoding errors with no requirement to "work around" such errors.
- A high proportion of human-readable textual content.
- Potential composition of the data format with other XML-encoded formats.
- Data structures that can be easily parsed by both humans and machines.
- Possibility to create vocabularies that can be developed in a distributed manner and combined flexibly.

With the introduction of the XHTML family of modules and document types, the W3C (World Wide Web Consortium) has helped to move the Internet content-development community from the days of malformed, non-standard mark-up into the well formed, valid world of XML. In XHTML 1.0, this move was moderated by a goal of providing for easy migration of existing, HTML 4 (or earlier) based content to XHTML and XML. With the advent of the XHTML modules, the W3C has removed support for deprecated elements and attributes from the XHTML family. These elements and attributes were largely presentation oriented functionality that is better handled via style sheets or client-specific default behaviour.

Going forward, XHTML family document types will be based upon this new, more structural functional collection. In the specification for XHTML, the W3C's HTML Working Group has defined an initial document type based solely upon modules. This document type is designed to be portable to a broad collection of client devices, and applicable to the majority of Internet content. Content developers who base their content upon the functionality expressed in this specification can be confident that it will be consistently portable across XHTML family conforming user agents.

3. Visual structure

The Blaise language is used to structure a questionnaire form logically (blocks, route, etc). But is also important to structure the questionnaire visually. In website design a process often something called *wireframing* is used. The wireframe is a quick representation of the questionnaire form layout, and helps to visualize the

process. In this wireframe process, the form designer takes the logical model of the questionnaire as a starting point, translates it into a number of forms, and creates visual representations of these forms. Figure 2 shows a simple example of a wireframe and the final resulting form.

Figure 2. A wireframe and the final form

Wireframe

Finished HTML form

The easiest way to create wireframes is to simply draw them on paper. If one is familiar with graphical software like Photoshop or Flash, this will work perfect. It is important to keep the drawing simple, but make sure it contains all the important elements.

4. Navigation

Blaise offers the questionnaire designer *active routing*. It means that the respondent will only see the relevant questions, and the irrelevant questions are skipped.

When a web form has some form of active routing, users will need adequate visual aides to navigate to the next question. They also need to be able to see how many questions they still have to go, and how many they have had. This could be implemented by means of a progress bar.

Users also need the ability to step back, to correct previous answers. All these navigations aides need to have a consistent and prominent look and feel. The buttons need always be in the same location on the form, the progress bar needs to be clearly visible.

A graphically pleasing interface for a form will put the respondents at ease. They do not have to strain themselves to fill it out, and moreover it shows the web application is built by a professional organisation. Be careful in the choice of colours. The number of different colours should small, and their use should be functional. Do not clutter the website with logos, banners and graphics. Keep it simple.

5. Some basic usability tips

When designing forms, there are some basic principles that have to be taken into account. The first one relates to the amount of information on the screen. A proper compromise has to be found here. At one end of the spectrum there is the question-based approach with one question per screen. This makes the screen readable but leads to a large number of screens. Progress bars may assist the respondents in keeping track of where they are, but it is not unlikely that they will pull out before they reach the end of the questionnaire. At the other end of the spectrum, there is the form based approach with all questions on one screen. This will clutter the screen, and also discourage the respondent.

With not too many questions, one question per screen should be preferred. And with many questions, a screen could contain small groups of related questions that do not require complex routing or editing. Ideally, all questions in a group should be visible on the screen without having to scroll. This limits the number of questions per screen to, say five.

It is not uncommon in surveys to have redundant questions. For example, in business surveys one could ask questions about all cost components, and moreover also the total costs, being the sum of the cost components. In questionnaires of commercial market companies there can even be duplicate questions. They are there also for checking purposes. If such questions are formulated in exactly the same way, respondents will think something is wrong ("Hey, I already answered that, what's going on?"). If it is important to check the data while respondents are still in the process of filling out the form, try to mould it in a shape like "Earlier, you specified this variable as that. We now have to calculate something using that; are you sure it is correct?".

Questions should be clear. Avoid difficult long sentences. Talk in the users' language, and keep to all the rules that also apply to question design in general, like avoiding ambiguous questions, asking one thing at the time, avoiding questions framed in the negative, and not asking questions needing recall of events that happened in the past.

6. References

Steve Krug (2005), Don't make me Think: A Common Sense Approach to Web Usability. Circlo.com Library.

http://www.w3.org/TR/webarch/#xml-formats

http://www.w3.org/TR/xhtml11/

http://www.w3schools.com/tags/

http://www.lukew.com/resources/articles/web_forms.html

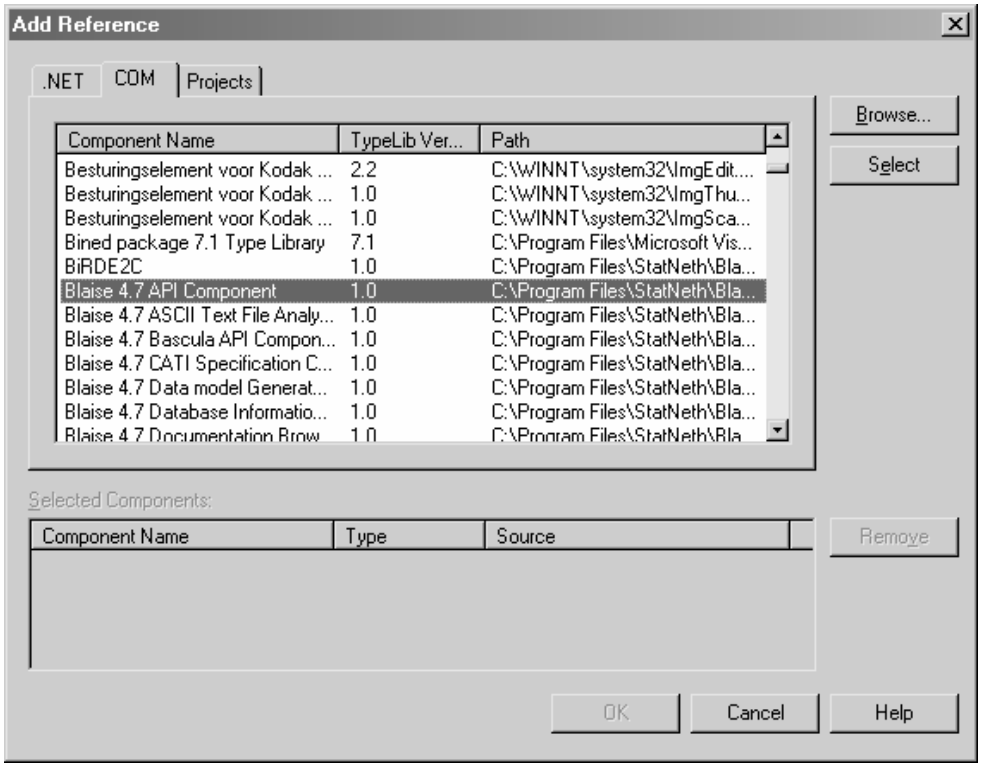
Using the Blaise Component Pack in the .NET environment

Rob Groeneveld (Statistics Netherlands)

The Blaise Component Pack (BCP) has been designed and programmed as a series of COM (Component Object Model) components. Now that the .NET environment has arrived, it is desirable to be able to use the BCP in the new environment. Basically, there is no major problem, because there are tools in Visual Studio .NET which enable the developer to work with COM objects as they are, so-called interoperability tools or Interop. These tools ensure that COM objects can be reached from the .NET environment. In essence, they put a wrapper around a COM object to make it into an assembly. Assemblies are the building blocks of .NET applications.

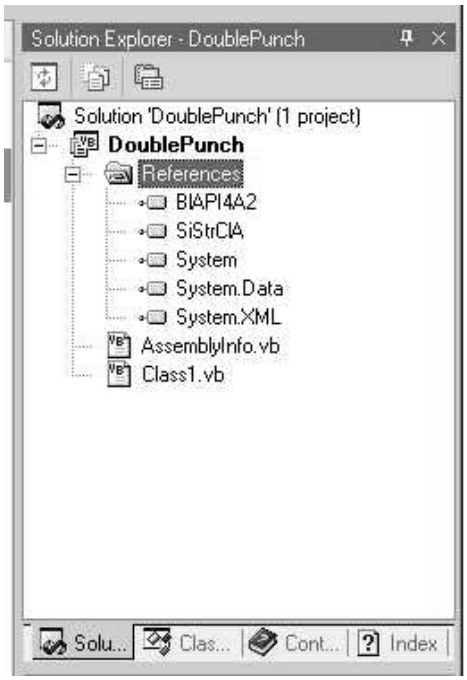
To use the Blaise API component in Visual Studio .NET, once you have started a new project, choose Project | Add Reference…, click on the COM tab and browse to the Blaise 4.7 API Component (see figure 1).

Figure 1. Adding references in Visual Studio .NET



Click on Select and on OK. In the Solution Explorer under References, the components BlAPI4A2 and SiSTRClA are now visible. The component SiSTRClA is needed by the Blaise API Component (see figure 2).

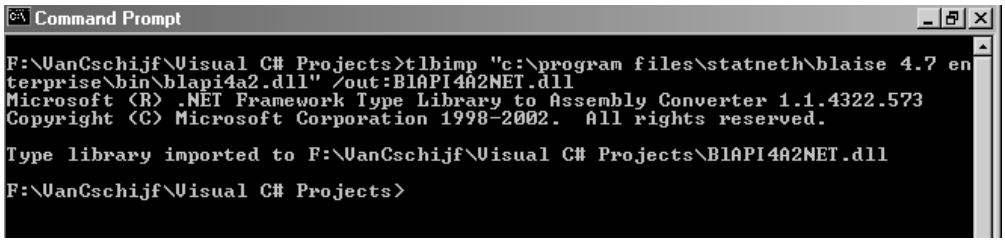
Figure 2. References to BCP components



You can program in Visual Basic .NET (VB.NET) or C# or any other .NET language using the Blaise components. For example, the DatabaseManagerClass in the BlAPI4A2 component is accessible in your .NET programs.

The other way to use the BCP is to convert the components to .NET assemblies using the Type Library Import program. TlbImp.exe is called with the BCP component as the first parameter and the assembly to be output in the '/out:' parameter. We suggest adding the suffix NET to the names of the BCP.NET assemblies. Figure 3 shows how this is done for the BlAPI4A2.dll.

Figure 3. Converting the Blaise API Component to a .NET assembly



The converted BCP assemblies can be used as References (now, of course, .NET References, on the first tab) in the .NET environment. If they are not visible in the list they can be found using the Browse button. The BlAPI4A2NET.DLL references the SiStrClANET component as before, but *both* references must be added manually in the .NET environment.

Some of the demos and examples shipped with Blaise 4.7 Enterprise are shown in the poster session and run as .NET programs. In Blaise 4.7 Enterprise the examples of the use of BCP are in Microsoft Visual Basic 6.0 (VB6). In the poster session we present examples in both VB.NET and C#. We also present an example of an Alien Router in the DEP programmed as a class library in .NET. Certain examples can be programmed more elegantly in the .NET environment because of the availability of new methods.

The examples in C# and VB.NET are available from the author.

References

Barwell, F., et al. (2001), Professional VB.NET, Wrox Press Ltd., Birmingham, UK.

Stefanski, M. (2002), Basiscursus C#, Academic Service, Schoonhoven, The Netherlands.

A Calculator DLL

Roberto Picha (U.S. Census Bureau, USA)

1. Introduction

The Census Bureau develops numerous instruments for computer assisted interviewing (CAI) surveys. Our instruments provide a great number of functionality to assist our Field Representatives (FRs) with conducting their interviews.

When the data we are collecting may be calculated values, we often provide the FRs with multiple ways of entering the data. For example, when collecting the square footage of a room, a respondent can provide data in different ways, such as 7.6 feet by 8.5 feet or 64.6 square feet. In this instance, we would probably provide both options to the FR for entering the data. Another example would be calculating the salary of a person being interviewed. The answer could be collected in two entries, the first being the amount and the second being one of several earnings options (yearly, monthly weekly). However, our Census Field representatives could come across situations not anticipated by our subject matter people. It is possible that responses could be given for earnings not covered by one of our options, and the FR would need to do calculations in order to make the response fit into one of the given options. This situation can be very frustrating for the FRs during data collection, and even though our Field Representatives are equipped with the normal Windows calculator, using that can also be annoying. In order for an interviewer to accomplish his or her task satisfactorily, it would require them to perform several keystrokes in order to transfer the value from the calculator to the answer field in the instrument. They would have to use Ctrl-C (to copy) inside the calculator textbox, close the calculator, and then finally Ctrl-V (to paste) in the answer field of the instrument.

Back in 2000 during the debriefing for the Consumer Expenditures Survey dress rehearsal, there were suggestions made by some of our users that it would be useful if the contents of the calculator could be transferred into the data entry program automatically. At that time the only answer given was, "It's not possible," because of the amount of effort required and because other tasks towards completing the instrument for production were more important. The difficulty was mainly because the calculator is an external application outside of the Data Entry Program.

2. Research

Some possible solutions were researched and some constrains were found while looking for a functional approach to satisfy our interviewers.

- The use a procedure for items where a calculation could be performed.
    - The limitation with this approach was that the use of recursive calls inside a procedure is not possible in Blaise.
- The use an alien router.
    - The limitation with this approach is that it would imply some work around the different variables in need of this functionality, i.e., you would have to define up front all variables that may require this functionality

- The BCP.

- This would allow us to use Visual Basic to build a customized calculator.

Of the three researching options the last one was chosen.

Blaise version 4.6 came with one simple example that demonstrates the use of an ActiveX DLL allowing the user to select a color from a pick list and dump the text content into the DEP entry field. The code for this seemed reasonably easy to follow. The ActiveX example would permit us to display a form. The form could actually be a simple application (i.e., a calculator) and the content from the form could be sent to the DEP via the DLL. This approach was the base used for the Calculator application, and the same approach can be utilized for other applications as well. We will be using this similar approach for our Builder Table in the SOC Listing instruments, discussed in Rob Wallace's Listing Instrument paper.

3. Implementation

This is a very simple application; nevertheless, it provides the foundation for some other more complex, robust applications, aiding surveys written in Blaise by applying innovative solutions in the data collection.

Here are the steps used to implement this DLL

a. Build an application in Visual Basic
   - Create a Visual Project Application
   - Programming a calculator in Visual Basic
   - Apply simple interface that simulates all functionalities available in the Windows calculator.

b. Create a project for a DLL
   - Associate the Blaise API call
   - Incorporate the calculator form into the DLL project
   - Create the DLL

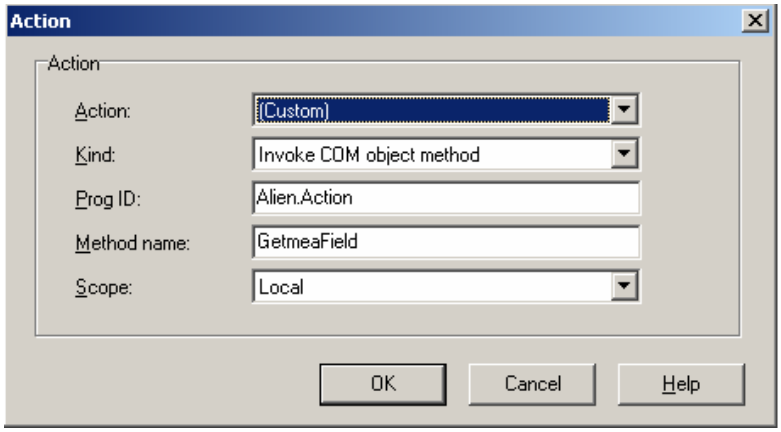
Code example

```
Public Sub GetMeAField(DB As Database, DS As DepState)

Dim Layout As BlAPI4A2.LayoutSet
Dim Parallel As BlAPI4A2.Parallel
Dim Page As BlAPI4A2.StoredPage
Dim Quest As BlAPI4A2.Question
Dim ActiveField As BlAPI4A2.Field

 ' Get hold of the currently active question:
 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)

 ' Now get the field of the question:
 Set ActiveField = Quest.Field
  ' Bring Calculator Form
  Form1.Show vbModal
  ' Retrive calculator value and send back to DEP
 DB.Field(ActiveField.Name).Text = Form1.lblOutput
```

```
    Unload Form1
End Sub
```

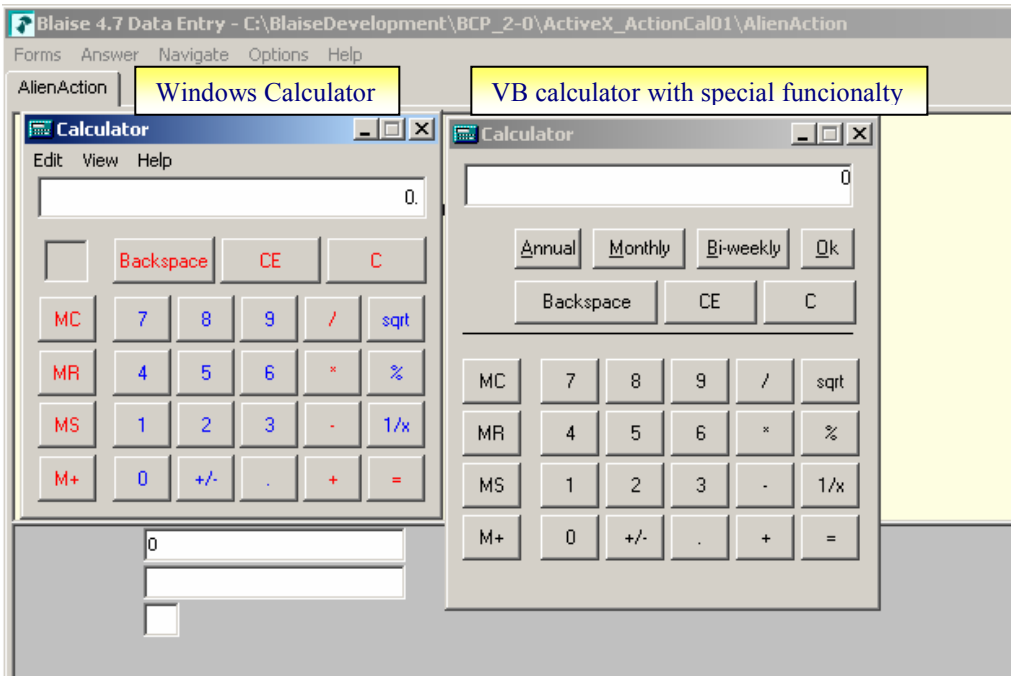
c. Install the DLL and Blaise runtime files
   - In order for this DLL to work within our environment we must do the following:
     - We must register (system32 directory) the new DLL so that the DEP can access it.
     - We also need the run time Blaise B4API.DLL (required file to interact with our DLL)

d. Associate a function key to the menu manager using Menu Editor (See picture below)



4. Comparing the Windows Calculator with the Customized DLL Calculator

The image below shows the standard Windows calculator on the left and the customized calculator on the right



In the customized calculator application the user may use mouse clicks or the keyboard (numeric pad) to make entries. Then, as stated previously, the results of the calculation are fed back to the entry field when the Ok button is clicked.

This version of the application is actually customized for a specific survey because it allows for special calculations of salary. Note the Annual, Monthly, and Bi-weekly buttons.

5. Usability

This application can be called from fields of various data types. The following table shows what happens when the data from the calculator is fed back to each type of field

Data type	**Limitations**	
String	No restrictions	
Enumerated	Yes	Truncation for decimals Not allowing negative Out of range return blank
Numerical	Yes	Truncation for decimals Not allowing negative Out of range return blank
Set	Yes	Truncation for decimals Not allowing negative Setting values in the DEP Out of range return blank
Real	Yes	Possible Truncations of Decimals Out of range return blank
Open	No restrictions	

Programming for Navigation in the Blaise System

Mark Pierzchala (Mathematica Policy Research, USA)

1. Introduction

Blaise for Windows was built from first principles to enable user navigation and in this respect has stood without peer among CAI systems. Three aspects of the Blaise system combine to enable navigation: (1) The rules checking mechanism, (2) explicit user navigation features, and (3) a page-based interface. This paper focuses on programming practices that work within the Blaise rules checking mechanism. In particular, it draws upon experience with large and complex instruments where navigational capability was considered a fundamental and required feature.

Any user should be able to move forward and backward and change values inumerable times in a Blaise instrument without incorrect execution of path, edits, or computations. If changing data causes incorrect execution, there is something wrong with the programming or specification of the instrument. The text below focuses on programming issues but some points are valuable for specification writers as well.

Blaise instrument testers should test navigation as part of their protocol. In at least a few test cases they should engage in extreme navigation and changing of data to make sure that correct execution is maintained. Should there be problems, they are probably due to one of four causes explained below and can easily be fixed.

2. The Rules Checking Mechanism

Any CAI system must be able to ensure the integrity of the data record. This includes the enforcement of flow and consistency between data values. If data collection proceeds only in the forward direction according to paths that have been carefully specified, programmed, and tested, it is relatively easy to ensure that the data record has integrity. However, any navigation within a CAI system is a deviation from a pre-planned path. There is no telling what route a user may take. While the number of specified paths may be large, the number of paths that may be taken as a result of navigation is orders-of-magnitude greater. If data are changed, the system must maintain data record integrity.

Maintaining data integrity, as a user changes data, places great demands on the system architecture. This is because changed values may open up a new path, close down a previously used path, or result in new data inconsistencies. The system must detect these situations and direct the user to deal with them. It must be able to do so no matter how the user has navigated, no matter in which order the user has changed data, and no matter where data have been changed.

2.1. Global and Constant Checking of the State of the Record

In a computer science sense, the *state* of a data record can be considered to be a summary of all the values that have been entered, or generated, up to the point of the interview. The Blaise system re-assesses and re-establishes the entire record's state every time a data value is changed and during certain other events such as changing interviewing language. This re-assessment proceeds linearly from the start of the record to the end, even past the point of the interview. It is *global*, meaning that the integrity of the entire record is guaranteed. It is *constant* meaning that it is carried out every time a data value is entered or changed.

2.2. Selective Checking Mechanism

The Blaise checking mechanism is also *selective*. It only invokes the rules of blocks that have had data changes as well as blocks that refer to any data values that have changed. This selectivity allows Blaise to handle extremely large questionnaires that can have hundreds or thousands of blocks and block instances (Frey 2000). It allows these large instruments to execute quickly, but at the same time to constantly ensure global record data integrity. An article that explains these issues is *Passing Information between Many-Arrayed Blocks*. It is available in the Blaise Help and also online (Blaise Services at Westat, 2001).

2.3. Fast Forward Capability

A hallmark feature of the Blaise system is the *fast forward* feature, better known as the *End* key. When you press the *End* key the Blaise system always puts you at the point of the interview, that is, the next appropriate question. This capability is only possible if the system has already globally checked all the rules before the *End* key is pressed. Since Blaise always re-executes the rules on a data change, it is always ready for a press of the *End* key or any other navigation. Many CAI systems never achieve this capability because they check only locally.

2.4. Working Within the Checking Mechanism Paradigm

The following four programming practices may intefere with the Blaise checking mechanism and disrupt navigaton. They are: (1) non-robust use of external records as metadata, (2) programming traps such as *do once* code, (3) uninitialized computed values used in IF conditions, and (4) forced backward jumps.

Use of External Records as Metadata

The use of external records as a source of metadata is an advanced application of the Blaise system and is used in a relatively few complex instruments. This technique usually occurs in conjunction with lookups such as a search of large lists of commodities, crops, place names, goods, medications, foods, or other lists that may contain up to tens of thousands of records. Such lookup lists usually contain an analytical code for use in downstream analysis, but may also hold question text, routing instructions, or edit limits. In effect, some of the instrument's metadata is held in the external file. This use of external records is an extremely efficient way of handling some complex situations, but an issue may arise where external metadata is read into the data record, but then becomes obsolete when the situation that led to its use changes. This can be controlled by programmers, but they must be aware that they need to do so. In the Consumer Expenditure Survey (sponsored by the US Bureau of Labor Statistics and fielded by the US Census Bureau) the instrument has a number of so-called laundry lists. Robust handling of possible changes in metadata was programmed into the Blaise source code. For example, special interviewer conventions were put in that allowed the interviewer to disable a row in such a 'laundry list' table (Stanford, Altvater, and Zeising, 2001).

It is not possible to show a generic Blaise source code example to handle these kinds of situations because they are so variable. If your organization is programming such a survey, it should be enough to know that the situation exists and that it should be handled robustly. The read-in metadata should either be replaced with the correct metadata (preferred solution) or it should be disabled. There should be explicit testing to ensure that metadata records can be changed in response to user actions and handled correctly.

Programming Traps

A programming trap executes a set of statements once and is of the following form.

Figure 1: *Programming Trap to Execute Source Code Once*

```
aDoOnce.KEEP
IF aDoOnce = EMPTY THEN
  {Execute some code}
  aDoOnce := Continue
ENDIF
```

This amounts to an explicit disabling of the Blaise checking mechanism. While there are legitimate uses of programming traps in Blaise, these are rare. For example, MPR does use a programming trap when it initializes a case as it is brought into memory. The intention is to establish the state of data collection when the case is brought up. But almost anywhere else programming traps are unnecessary and can cause bad execution when users navigate to change data as they inevitably will. Programming traps should be avoided in most situations.

Uninitialized Computed Values Used in IF Conditions

Some IF condition statements include fields whose values are computed. An additional complication arises when there is a series of interrelated IF conditions, in the midst of which, these values are computed as shown in Figure 2.

Figure 2: *Computed Values Used in IF Conditions*

```
City
IF City = Large THEN
  PathA := True
ELSE
  PathB := True
ENDIF

IF PathA = True THEN
  J
ELSEIF PathB = True THEN
  K
ENDIF
```

In the simplistic example above, *City* is a user-entered values while *PathA* and *PathB* are computed values. If the user indicates that *City* equals *Large*, *PathA* gets the value of *True*. However, if the user backs up and changes the value of *City* to *Small*, the field *PathA* does not get reset.

A programmer may be tempted to try to unset values of *PathA* and *PathB* within the IF condition structure. However, this can be nearly impossible to do accurately in complex IF structures. It also leads to overly complex source code. The best, most elegant, and totally reliable way of handling this situation is to initialize the values of *PathA* and *PathB* before the IF conditions as shown in Figure 3.

Figure 3: *Initialized Computed Values Used in IF Conditions*

```
PathA := EMPTY
PathB := EMPTY
City
IF City = Large THEN
  PathA := True
ELSE
  PathB := True
ENDIF
```

The code set out above always starts the fields *PathA* and *PathB* as EMPTY and relies on the rules to compute the correct one to a value of *True*. The global

execution of a block's rules ensures that all computations within the block are redone, and that the paths will be set correctly.

Forced Backward Jumps

There is a way in Blaise to force a backward jump, though this should never be programmed. It is done by emptying out a NOEMPTY, on-route field, in a previous part of the form. The checking mechanism will assess the record from the start, notice such a field on the route and place the cursor on that field. The source code that can execute this forced backward jump is shown in Figure 4.

Figure 4: *Forced Backward Jump*

```
A
{Much code, perhaps covering many Blaise pages}
Q
IF Q = Red THEN
  A := EMPTY
  {Some trapping mechanism is also implemented
   so that A is not continously set to EMPTY.
   This is difficult to do robustly.}
ENDIF
```

When the user enters the value of *Red* in field *Q* the user will immediately be placed in the field *A* since the value of the NOEMPTY field *A* is EMPTY. There are several notes about this situation:

1. The trapping mechanism within the IF condition leads to overly complicated code that is difficult to get working correctly in all situations.
2. For the user, the transition back to *A* can be very abrupt and unexpected.
3. The user may in fact have made a data entry error in field *Q*. The code above does not allow the user to correct the error and avoid the jump.
4. There may be a lot of pathing and other rules that depend on the value of *A*. If after field *A* is emptied, the user decides to exit before entering another value in *A*, a lot of work will be undone.

The requirement of such a forced jump is often due to the fact that the specification writer is unfamiliar with the way Blaise works. A far better way to do this would be to implement the navigation through an edit, that is, to use the Blaise edit message box as a user navigation device. Figure 5 shows the Blaise source code.

Figure 5: *Optional Backward Jump*

```
A
{Much code, covering many Blaise pages}
Q
IF Q = Red THEN
  CHECK
  Q <> Red INVOLVING (A)
  "The entry of the value Red indicates that perhaps
   A has the wrong value.
   @/To change the value of A, select A in the jump list.
   @/To correct the value of Q, press Escape."
ENDIF
```

In this refinement, the user sees an edit message box and is told why a correction in *A* is necessary. It is the user who makes the choice of whether to travel to *A* or to fix the value in Q. There is no need for a trap and it is far easier to implement.

Other Interfering Programming Practices

Since Blaise is a high-level programming language, there are undoubtedly other ways in which a programmer may interfere with the rules checking mechanism. It

would be impossible to cover all these situations, if they could be enumerated. The main point is for programmers to understand and work with the checking mechanism. An attempt to work around it usually fails and it often ends up with source code that is far too complex and far too much work.

2.5. Navigation in an Already Collected Data Record

So far this paper has talked about navigation within the context of a regular interview. However, there are surveys where the data are collected in a different mode or system and it is necessary to navigate within an already collected data record. Two situations are *missing item followup* and *critical item followup*.

Since the Blaise rules checking mechanism always re-establishes the state of the record every time there is a data change, it has the capability to accept data from an outside source, evaluate it, and report to the user what needs to be fixed. The goal is within-record, within-place data correction. The *within-record* goal eliminates the need to merge data after data collection. All data are within one record at all times. The *within-place* goal allows newly collected data to be entered in the intended place in the record and evaluated within the context of the Blaise datamodel rules. This way, if the changed data value opens up a new path, the fields along the path can be collected. Similarly the changed value can close up now-obsolete paths or result in new consistency edits to resolve (Pierzchala 2006).

Missing Item Followup

A data record may be complete except for some holes in the data. The use of the *Fast Forward* key (the *End* key) allows the user to get to them. An example of this is the American Community Survey Telephone Followup (Diskin and Stulik, 2000) where Blaise is used to follow up paper-collected data entered elsewhere.

In *missing item followup*, the intent is to fill in all blank on-route data cells. This is best implemented with the use of the NOEMPTY attribute at the field level. Figure 6 illustrates that this kind of followup is implemented very easily.

Figure 6: *Missing Item Followup Using the NOEMPTY Attribute in Fields*

```
FIELDS
  Age "How old are you?" : 0..120, NOEMPTY, DK, RF
```

In the example, if the read-in data do not have a value for *Age*, when the followup interviewer presses the *End* key, the cursor will stop at *Age*. There is virtually no work to achieve this capability since most organizations already program this way.

Critical Item Followup in a Multimode Context

The term *critical item followup* implies that some fields are more valuable than others and that during the followup, only those items are targeted. However, in web or paper modes, it may be desirable to allow blank values and in CATI, entry of DK or RF in the hope of collecting these critical items later. To implement this kind of followup, the syntax for both FIELDS and RULES is more complex. Figure 7 illustrates the syntactical approach to this.

Figure 7: *Critical Item Followup Using the EMPTY Attribute in Fields*

```
FIELDS
  Job "Do you have a job?" : tYesNo, EMPTY, DK, RF
  Age "How old are you?" : 0..120, EMPTY, DK, RF

RULES
  Job
  IF Mode = CAI THEN
    CHECK
    Job <> EMPTY "Answer required."
  ENDIF

  Age
  IF Mode = CAI THEN
    CHECK
    Age <> EMPTY "Answer required."
  ELSEIF Mode = WebMode THEN
    SIGNAL
    Age <> EMPTY "Your answer to this question
                  is very important. Please respond."
    CHECK
  ELSEIF (Mode = CIFollowUp) THEN
    CHECK
    Age = RESPONSE
    "Your answer to this question is very important.
     @/Please do your best to answer this question.
     @/[INT: IF RESPONDENT REFUSES, END THE INTERVIEW.]"
  ENDIF
```

The questions operate differently in the three modes and in critical item followup. In CAI (CATI or CAPI), an answer is required, but it may be a DK or RF. In the web mode, both fields are allowed to be blank, but a prompt is given to encourage response for *Age*. In paper the answers are allowed to be empty.

In critical item followup, no matter which mode is followed, *Age* must not only be answered, but it cannot be DK or RF. When the interviewer presses the *End* key, the cursor will fly past the field *Job* but stop at *Age* where a proper response can be obtained. However, the interviewer is instructed to end the effort if an answer cannot be gathered for *Age* because the case will be considered a unit refusal.

The source code in Figure 7 is naturally more complex than that of Figure 6 for three reasons. First, it applies to a more complex multimode situation where data entry is all in Blaise as oppposed to the situation in Figure 6 where data are read into Blaise from another system. Second, there is a differentiation between modes for the handling of each item. Third, the items are handled differently for critical item followup. On the other hand, this approach makes on-the-fly switching between modes a natural capability as long as you correctly control the field *Mode*.

Any alternative, such as using different systems for different modes, or using different Blaise data models for different modes, can be worse. You would still have the same specification issues, but additionally have multiple source code decks that are mostly redundant, but different in subtle ways. Any update would have to be applied in two or more places. Survey management options would be highly curtailed, and there would be a data merging step. The approach in Figure 7 developed by MPR is not difficult to implement, and is a large overall time saver with many positives (Pierzchala 2006; Pierzchala, Wright, Wilson, and Guerino, 2006; Pierzchala, Wright, Wilson, and Guerino, 2004).

3. Explicit Navigation Features and the Page-Based Interface

The introduction referred to two other aspects of Blaise navigational capability: explicit user navigation features and a page-based interface. It is beyond the scope of this paper to discuss these other aspects of navigation in depth, but a brief review and references are in order.

3.1. Explict Navigation Features

Blaise for Windows has 13 explicit ways to navigate. These include the arrow keys (jointly considered as one method), the *Home* and *End* keys, the *PageUp* and *PageDown* keys, edit jump boxes, jump to tag, tabs, and the mouse. Each method has its use and these are described in Appendix A of Pierzchala and Farrant (2000). It also briefly covers programmer enhancements to these navigation features.

3.2. Page-Based Blaise Interface

The Blaise for Windows interface features a split screen paradigm. The bottom part of the screen is officially known as the *form pane* but effectively it is a navigation page. It corresponds to a page in a paper questionnaire and is the unit of operation of the *PageUp* and *PageDown* keys. The Blaise page eliminates the *segmentation effect* found in systems that display one question one per screen (Groves, Berry, and Mathiowetz, 1980). The segmentation effect works against navigation because users cannot see where they are going and it takes a lot of key strokes to get to where you want to go. There can be programmer enhancements to the page display that make navigation easier. These include putting labels in pages, having large *data density*, and other techniques; see Pierzchala (1997), Pierzchala and Manners (2001), Pierzchala and Farrant (2000), and Wensing et al (2003).

4. References

Blaise Services at Westat Technical Note, (2001). Passing Information between Many-Arrayed Blocks. *http://www.westat.com/blaise/pdf/bsw-0006.02.pdf*. There is a supporting zip file of Blaise source code that illustrates the issues. http://www.westat.com/blaise/documentation.cfm#Blaise_Technical_Notes.

Diskin, B. and Stulik, K. (2000). Using Blaise to Automate the Clerical Edit and Telephone Follow-Up Component of the American Community Survey. International Blaise Users Conference, Kinsale, Ireland. http://www.blaiseusers.org/Ibucpdfs/2000/us_bureau_of_census.pdf

Frey, R (2000). Developing a Blaise Instrument for the Spanish Bladder Cancer Study. International Blaise Users Conference, Kinsale, Ireland. http://www.blaiseusers.org/Ibucpdfs/2000/spanish_cancer_study_westat.pdf .

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.

Pierzchala, M. (2006). Disparate Modes and Their Effect on Instrument Design. International Blaise Users Conference, Arnhem, The Netherlands.

Pierzchala, M., Wright, D., Wilson, C., Guerino, P. (2006). Telephone Collection as Part of a Multimode Survey. Paper presented at the 2006 Telephone Survey Methodology Conference II, Miami, Florida (U.S.), January, 2006

Pierzchala, M., Wright, D., Wilson, C., Guerino, P. (2004). Instrument Design for a Blaise Multi-Mode Web, CATI, and Paper Survey. International Blaise Users Conference, Gatineau, Canada. http://www.blaiseusers.org/IBUCPDFS/2004/24.pdf

Pierzchala, M. and Manners, T. (2001). Revolutionary Paradigms of Blaise. International Blaise Users Conference, Washington, DC, US. http://www.blaiseusers.org/ibucpdfs/2001/Pierzchala_Manners_IBUC_Paper.pdf .

Pierzchala, M. and Farrant, G. (2000). Helping non-Blaise Programmers to Specify a Blaise Instrument. International Blaise Users Conference, Kinsale, Ireland. http://www.blaiseusers.org/Ibucpdfs/2000/how_to_specify.pdf.

Pierzchala, M. (1997). Optimal Screen Design in Blaise. International Blaise Users Conference, Paris, France. http://www.blaiseusers.org/Ibucpdfs/1995-1998/pierzc97.pdf

Stanford, V., Altvater, D., and Ziesing, C., (2001). Programming Techniques for Complex Surveys in Blaise. Blaise Users Conference, Washington, DC. http://www.blaiseusers.org/ibucpdfs/2001/Altvater-Stanford-Zeising--IBUC_paper.pdf

Wensing, F., Barresi, J, and Finlay, D. (2003). Developing an Optimal Screen Layout for CAI. International Blaise Users Conference, May 2003, Copenhagen, Denmark. http://www.blaiseusers.org/IBUCPDFS/2003/Developing_an_.pdf.

The Selection of Strata in Nonresponse Adjustment

Barry Schouten (Statistics Netherlands)

1. Introduction

Nonresponse in household surveys can be a threat to the quality of statistics. Research shows that often the response to these surveys is selective with respect to demographic characteristics like age and household composition. For this reason estimators are usually adjusted to account for nonresponse. The Bascula tool is added to Blaise in order to support nonresponse adjustment.

Nonresponse adjustment methods make use of covariates that are available for both respondents and non-respondents. A problem is the selection of covariates that relate both to the key survey questions and to the response behaviour. Therefore, often the process of selection is performed in two steps. First, candidate covariates are selected that relate either to the nonresponse mechanism or to the key survey topics. Second, adjustment is performed using these candidate covariates using for instance Bascula.

We present a classification tree method that allows for the construction of weighting strata that simultaneously account for the relation between response behaviour, survey questions and covariates.

The method is computationally intensive but it facilitates nonresponse adjustment in a single step. The method may be included in Blaise to combine selection of strata and weighting in one tool.

In section 2 we give some background to the selection problem. In section 3 we discuss results. Section 4 concludes.

2. The selection of strata using classification trees

Nonresponse to surveys affects population estimators in case on average respondents and non-respondents give different answers to the survey questions. Auxiliary information is usually linked to the survey so that potential bias can be detected and corrected for. Commonly used techniques are linear weighting, multiplicative weighting and propensity score weighting. For an overview of adjustment methods we refer to Bethlehem (2002) and Kalton and Flores-Cervantes (2003).

Crucial in the successful employment of adjustment methods is the validity of the assumptions underlying the methods. Most techniques assume that conditionally on a set of available auxiliary variables respondents cannot be distinguished from non-respondents when it comes to the survey topics. Hence, in case the values of these variables are fixed response is at random, a feature called Missing-at-Random in the literature. Although, it seems reasonable that fixing a number of characteristics makes respondents resemble non-respondents, there is little empirical evidence in practice to support the assumption. In fact, when more auxiliary information becomes available as was the case at Statistics Netherlands, it follows that current weighting models can be improved (see Schouten 2003). The additional variables either give a better explanation of response behaviour or are better predictors of the survey questions.

In practice the formation of strata is not straightforward. When the categories of auxiliary variables are crossed it may occur that empty or almost empty strata are formed. In that case the answer of non-respondents in the same stratum cannot be predicted or the prediction is based on too small a number of respondents causing variance to grow. Furthermore, it may not be efficient to cross every category of one auxiliary variable with every category of another auxiliary variable. Finally, a criterion must be chosen that makes it possible to compare different sets of strata. When should one choice of strata be favoured to another choice of strata?

The problem of forming strata is addressed extensively in the literature. Little (1986) for instance suggests forming so-called adjustment cells by first dividing respondents that have similar response behaviour into separate cells and then pooling those cells that give similar answers to the survey questions.

Here we do not make assumptions about the missing data mechanism. We have two motivations for employing the weaker Not-Missing-at-Random assumption. First, as we already mentioned we do not believe the Missing-at-Random assumption to always hold since at Statistics Netherlands newly available auxiliary information indicated that current weighting models may still lead to biased estimates. Also, one can never preclude that in the future more relevant auxiliary information can be deployed in the adjustment of nonresponse. Secondly, even when the final set of weighting variables comes close to being Missing-at-Random, we still need a rule to decide which variables to use and which to omit in the adjustment.

Schouten (2005a and b) shows that in general an interval can be set up for the bias of the response mean and the bias of the poststratification estimator. The width of the bias interval depends on the correlation between the 0-1 response indicator and auxiliary variables and the correlation between a survey question and auxiliary variables. The intervals give us the maximal absolute bias, i.e. the bias under the worst case scenario. We can use the maximal absolute bias as a criterion for the selection of strata. Strata are only subdivided into new strata in case the new stratification leads to a significant decrease in the maximal absolute bias.

It seems straightforward to choose the width of the bias interval as a criterion to compare weighting models, because this interval accounts simultaneously for the relation between response behaviour and auxiliary information and the relation between survey questions and auxiliary information.

However, we still need a strategy to select strata based on the criterion above. We propose a classification tree method with the interval width as a splitting rule and the significance of the decrease in interval width as stopping rule. This method enables adjustment for nonresponse in a single step as the resulting classification tree represents the stratification to be used in the weighting of the response.

A classification tree is an ideal method to enforce categories to be crossed only when this really leads to a more optimal set of strata. More optimal in the present setting means that the new stratification gives a bias interval, i.e. the absolute bias that is maximally possible is smaller for the new stratification.

Classification trees are constructed top-down. We let the root of the tree consist of all respondents. Hence, the starting point is only one stratum and the poststratification estimator reduces to the response mean. The first step, then, is a bisection of all respondents into two disjoint groups, so-called nodes. In each subsequent step one of the nodes is selected and split again into two disjoint groups. This process is repeated until no more node is allowed to be split. The end nodes, called leaves, will be the strata in the weighting of the response. The splits

are made using classifiers, in our case the categories of auxiliary variables that are available for both respondents and non-respondents.

In our case we choose the following splitting and stopping rules, where α, K, R_1 and R_2 are prespecified parameters:

Splitting rule:

Choose that bisection of a node that leads to the largest decrease in the width of the bias interval of the corresponding poststratification estimator.

Stopping rules:

1. A split is not allowed if the *p*-value corresponding to the standardised decrease in interval width is larger than α.
2. The maximum number of leaves is K.
3. A node cannot be split if the number of respondents in that node is smaller than R_1.
4. A candidate node cannot be formed if the number of respondents in that node is smaller than R_2.

For details about the methodology and algorithms we refer to Schouten (2005a and b).

Classification trees and its continuous counterpart regression trees are today assigned as data mining technique but go back to the sixties. It all started with the so-called Automatic Interaction Detector (AID) proposed by Morgan and Sonquist (1963). They suggest to partition a population in homogeneous subpopulations by means of repeated binary splits. In the succeeding decades several variants of their technique were developed, e.g. the well-known CHAID by Kass (1980). For an overview of classification and regression trees see for example Breiman et al. (1984) and Murthy (1998). We employ these techniques to efficiently adjust for unit nonresponse in surveys.

The classification tree that we have developed does not resemble any other existing classification tree method when it comes to the splitting and stopping rules. This is because we want to minimise the width of the bias interval. The leaves of our classification tree form the weighting strata in the poststratification estimator. A stratum is split into two new strata whenever it leads to the largest decrease in width of the bias interval for the poststratification estimator. However, in case this decrease is not significant on a prescribed level, then the split is not permitted and the classification stops.

3. Results

We apply the classification tree method to the Dutch Integrated Survey on Household Living Conditions (Permanent Onderzoek Leefsituatie in Dutch) for the years 1998 and 2002. We will abbreviate the survey by its Dutch acronym POLS. POLS is a large continuous survey with questions about issues like health, social participation and recreational activities.

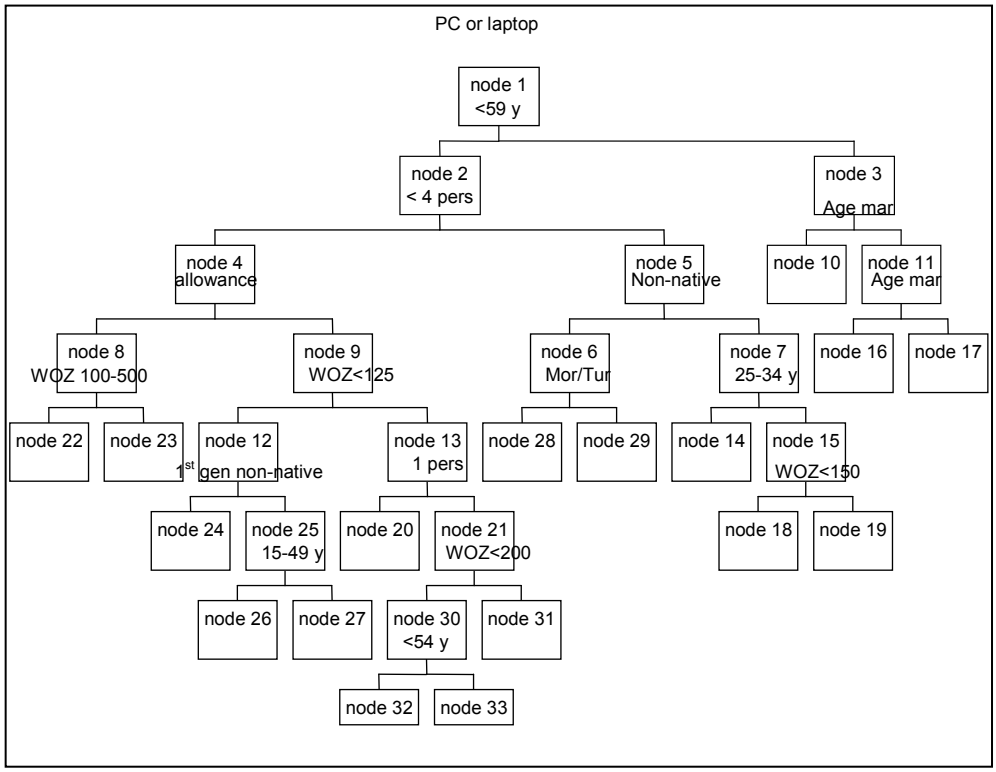
The survey is a two-stage sample, in which the clusters in the first stage are formed by municipalities. From the clusters simple random samples without replacement

are drawn consisting of persons. The first-order inclusion probabilities differ only for age. All persons of 12 years and older have the same probability to end up in the sample. In this paper we regard all persons of 12 years and older and omit only the nonresponse due to frame errors. The 1998 and 2002 samples then consist of, respectively, 36136 persons and 39170 persons. The size of the response was 21571 persons in 1998 and 22259 persons in 2000, i.e. a response rate of respectively 60% and 57%.

We selected one survey question from the POLS questionnaire, namely whether a person owns a personal computer or laptop. We also selected one auxiliary variable, whether a person receives some form of social allowance (disability, unemployment, social security), and treated this variable as if it was a survey question.

To the survey we linked demographic and regional variables, information about jobs and social allowances and fieldwork information. The auxiliary variables that we used are gender, age, ethnic origin, ethnic generation, marital status, children living in the household, household type, household size, degree of urbanization, province in the Netherlands (separate categories for four largest cities), size of municipality, average value of houses at postal code area, proportion non-native in postal code area, job, old-age pension, disability allowance, unemployment benefit, social security, interviewer district, interviewer seniority and interviewer gender. All non-categorical variables like age and the average value of house in the postal code area were made categorical. The dummy-variables corresponding to the categories of the auxiliary variables were used as classifiers in the algorithm.
Figures 1 and 2 show the classification trees for the two selected variables. The trees both accidentally contain 33 nodes and 17 leaves. The leaves are used as strata in the poststratification estimator.

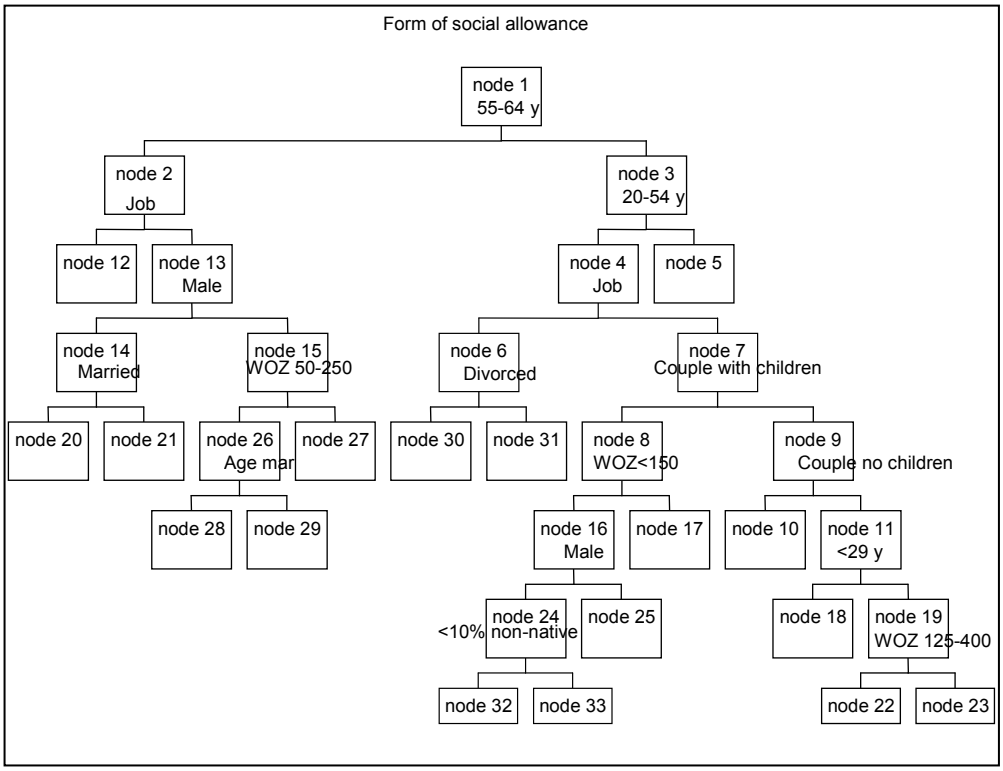
Figure 1: The classification tree for ownership of a personal computer or laptop.



We take figure 2 as an example. Whenever a node is split, the classifier is attached to the node. The labelling of the nodes indicate the order in which nodes where created in the classification tree algorithm. The root in figure 2 is split based on the question whether a person is between 55 and 64 years of age. All persons having an age in this interval go to node 2. All other persons go to node 3. Next node 3 is split into nodes 4 and 5 based on a further classification on age. The persons younger than 54 go to node 4, the persons older than 64 go to node 5. Node 4 is then split based on the question whether a person has a job. Node 5 is a terminal node and is not split. This node corresponds to the stratum 65 years and older. In the sixth iteration finally node 2 is split based again on having a job. Here node 12 is a stratum, and consists of all persons younger than 55 years that have a job.

When we move down along the branches of the tree, the nodes are based on an increasing number of classifications. Some of these may be nested, e.g. in figure 2 the variable age is twice used as a classifier. Consequently, the strata may be rather exotic when compared to usual weighting models.

Figure 2: The classification tree for social allowance.



In table 1 we compare the classification tree estimates to the estimates using the current POLS weighting model. In table 1 for the classification estimates also the 95%-confidence interval is constructed using approximated jackknife standard deviations.

Table 1: The estimates according to the classification tree method and the current POLS weighting model. For the classification tree estimates also the 95%-confidence interval is given that is computed using the jackknife approximations.

	Classification tree stratification	*Current POLS model*
Owner of PC	57.6% (± 0.6%)	58.3%
Social allowance	11.5% (± 0.4%)	11.0%

From table 1 we can see that the classification tree estimates do deviate from the estimates using the alternative model. Most differences fall, however, within the 95%-confidence intervals. This is true in general for almost all survey questions that we investigated. Furthermore, we must remark that the current POLS weighting model does not contain all the auxiliary variables that were used in the classification tree method. In case these variables are added, then differences are very small. Importantly, the number of strata in the classification tree stratification is much smaller than that in the current weighting model.

4. Conclusions

We argue that the usual missing-at-random assumption is not always valid in surveys. However, even if the assumption is true for the final weighting model, we need a criterion to form strata or adjustment cells in non-response adjustment. In this paper the missing-at-random assumption is, therefore, not made. We show that general intervals can be set up for the bias of the response mean and the poststratification estimator. We propose to minimise the width of these intervals.

Classification trees are candidate tools to form strata economically and in an automated way. Strata are divided into substrata only in case there is a significant decrease in interval width, leaving those strata alone that do not lead to any further decrease. Since the prediction of survey questions and the relation to response behaviour is combined in one splitting criterion, the strata can be formed by an automated algorithm. Hence, the classification tree method provides a tool to perform weighting in one step.

Approximations for the variance of the poststratification estimates come as a useful by-product of the jackknife-method. A proposed split of a tree node is executed only in case the decrease in interval width is significant. The jackknife-method is employed to approximate the variance of this decrease. However, at the same time the variance of the poststratification estimates can be computed while only marginally increasing the computation times.

There are also some drawbacks to the proposed classification tree method. First, the tree structure turns out not to be very stable. Even for two quite large samples the resulting trees may have quite different forms. However, due to multicollinearity in the variables the estimates are rather stable. In case the tree of one sample is applied to another sample, the estimates do not change much. Second, the computation times of the classification tree algorithm are considerable, since the number of nodes and splits to be investigated can become quite large. For practical purposes the current software is too slow and need to be made more sophisticated. Third, the classification gives a set of strata for each survey question and it does not seem straightforward how to combine those sets into one set of strata that suit all survey questions. Fourth, we found that the algorithm is not optimal. Examples can be constructed where trees exist that give a smaller bias

interval. In most cases these trees can be formed by choosing splits in the first iterations that are close to not being significant.

Summarising, we distinguish the following advantages and disadvantages:
Advantages:

- The selection of auxiliary variables can be done in one step.
- The construction of weighting models can easily be automated.
- The formation of strata is economical.
- The variance of the poststratification estimator can be approximated synchronically.

Disadvantages:

- Computation times are considerable.
- The stability of the tree structures is poor.
- The combination of sets of strata corresponding to different survey questions is not straightforward.
- The proposed algorithm is suboptimal.

We begin with the last disadvantage. We belief the suboptimality of the algorithm to be marginal. Only after careful and labour-intensive analysis we were able to construct trees that correspond to a slightly more optimal set of strata.

The computation times may be shortened by more efficient programming and using more specialised software. The routines for the classification trees were written and coded in S-plus by the authors. We propose to implement the algorithm in existing software like Bascula.

The stability of the resulting trees is a more difficult problem. However, to our opinion it is not a serious problem as long as estimates are stable. A promising technique in this respect is the Random-Forest method developed by Breiman (2001). Vanhommerig (2005) investigated the use of this technique for our purposes.

Finally, the combination of weighting models for different survey questions may be circumvented by the use of a multidimensional splitting rule. Instead of splitting the population separately for each survey question, we may choose the node and classifier that correspond to the largest decrease in the interval width over all survey questions. Also, this question needs further research.

5. References

Bethlehem, J.G. (2002), Weighting Nonresponse Adjustments based on Auxiliary Information, In Survey Nonresponse (ed. R.M. Groves, D.A. Dillman, J.L. Eltinge, R.J.A. Little), 275–288, Wiley Series in Probability and Statistics, John Wiley & Sons, New York, NY, USA.

Breiman, L. (2001), Random forests, Technical paper, Statistics Department, University of California, Berkeley, CA, USA. Available via http://www.stat.berkeley.edu/users/breiman .

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984), Classification and Regression Trees, Chapmann & Hall,Boca Raton, FL, USA.

Kalton, G., Flores-Cervantes, I. (2003), Weighting Methods, Journal of Official Statistics, 19, 81–97.

Kass, G.V. (1980), An Explanatory Technique for Investigating Large Quantities of Categorical Data, Journal of the Royal Statistical Society C, Applied Statistics 29, 119–127.

Little, R.J.A. (1986), Survey Nonresponse Adjustments for Estimates of Means, International Statistical Review, 54, 139–157.

Morgan, J.A., Sonquist, J.N. (1963), Problems in the analysis of survey data, and a proposal, Journal of the American Statistical Association 58, 415–434.

Murthy, S.K. (1998), Automatic Construction of Decision Trees from Data: A Multidisciplinary Survey", Data Mining and Knowledge Discovery 2, 345–389.

Schouten, J.G. (2003), Adjustment for Bias in the Integrated Survey on Living Conditions (POLS) 1998, Paper presented at the 14th International Workshop on Household Survey Nonresponse, August 2003, Leuven, Belgium.

Schouten, J.G. (2005a), A Selection Strategy for Weighting Variables under a Not-Missing-at-Random Assumption, to appear in the Journal of Official Statistics.

Schouten, J.G., Nooij, G. de (2005b), Nonresponse Adjustment using Classification Trees, Research paper 05001, Methods and Informatics Department, Statistics Netherlands.

Vanhommerig, R. van (2005), Nonresponse Adjustment using Random Forests, Masters thesis, Mathematics Department, Universiteit van Utrecht.

CAPI System at Central Statistical Bureau of Latvia

Norberts Talers & Palvels Onutrijevs (Central Statistical Bureau of Latvia)

The purpose of the CAPI system development is to increase the work quality and timeliness of the interviewer section in a sense that data collection can be made easier and faster for the interviewers. As the data is entered directly into a laptop no more data entry is necessary from paper forms, and the data validation procedures are carried out on – site, while interviewing the respondent. Wireless telecommunications system utilizing the GPRS / EDGE –technology appeared to be the easiest and most inexpensive solution. The GPRS coverage is good in all parts of Latvia. In the current system data to and from interviewer is sent as an FTP-transfer via a GPRS connection. Connection times are few minutes at most and the cost is by the amount of transferred data. The gain of using mobile networks and GPRS is that interviewers are able to send the data virtually any time, for example after receiving the information from every respondent thus making the time from entering the information into the laptop to getting the data to the office very short and diminishing the problem of possible data loss due to some kind of a technical breakdown. Also, the management of interviewers work becomes easier and more straightforward while using CAPI technologies, because of possibility to control the process at any time.

At the beginning of 2004 a pilot project on data collection with 5 interviewers using laptops and Blaise was carried out; 100 respondents were questioned on Labour force survey. In 2005, on May, the development of CAPI system at CSB was started. One person was responsible for developing CAPI system, and two programmers were responsible for developing first questionnaire in Blaise. In December 2005 45 laptops were received, 45 interviewers were trained, and from the January 2006 the data in Labour force survey is being collected by using CAPI technologies. From middle of March 2006, EU – SILC survey data will be collected in this system also. CSB has plans to move all permanent surveys to CAPI environment by the end of 2007.

The system consists of three logical parts:
1. Case management system on interviewers laptops;
2. Data transfer system via GPRS;
3. Information management at the office.

1. Case management system on interviewers laptops consists of two software types – for the data entry of questionnaires specialised data entry software Blaise is used; for the case management a tool developed by CSB programmers is used. When working with a laptop, an interviewer is working with the case management system with several possibilities:
   - Receive the data on new surveys (respondent list and questionnaire itself)
   - Send the data about surveyed respondents, and the data itself
   - Work with received respondent list, surveying respondents

   The core case management system is the place where an interviewer can see the respondent list for a survey chosen, with possibilities to set a meeting time for a specific respondent, or to open Blaise questionnaire, to which specific commands are passed to Blaise from case management system and start collecting data. Also, interviewers have possibilities to see the statistics on their work done (how many respondents have responded, how many non response, or how many not yet questioned). Different filtering functions are prepared to ease the work of an interviewer.

2. Whenever data sending and receiving is taking place, GPRS is used for that – on each of the laptops GRPS card is installed, which is used for connection to service provider. In case of receiving an update of the respondent list, or receiving completely new questionnaire with respondent list, interviewers just have to press one button, enter authorization information and wait for acknowledgement information from the system about successful data receival, or about the failure. Technically, a script is activating GPRS connection on a laptop, connecting to an FTP server at office of CSB, and downloading a password protected zip file, from which upon the receival data is extracted. Some file consistency checks are done in order to be sure that the file is not corrupted.

   When the data is sent from the laptop to our office from point of view of an interviewers it is much the same as in case of receiving the information. Scripts are collecting the needed files to be transferred, zip them in a password protected archive and put on an FTP server at office of CSB.

3. Information management at the office includes several parts:
   - A tool for respondent list separation in portions for each interviewer based on division by territories predefined.
   - A tool for archiving of the respondent list part and the questionnaire, and putting that on an FTP server;
   - Getting the information sent back by interviewers from FTP server, with possibilities for interviewers' section chiefs see the progress of work on surveying respondents.

   There is also possibility to manage the information (e.g. creating additional folders, deleting the information, updating the case management system etc.) that resides on the laptops directly from the office by using scripts that are sent to laptops via the GPRS connection.

   In nearest future it is envisaged to implement a simple messaging system that will allow interviewers receive information from the office and send some messages to the office as well as to other interviewers via GPRS connection. This will allow avoiding using phones in order to contact people, thus diminishing costs.

Multiple Researchers working at the Blaise Benchmark Services for the Disabled Act (WVG)

Carlo Vreugde & Mark Gremmen (SGBO, The Netherlands)

0. Introduction

Setting up a benchmark is a large operation and involves a complex procedure, involving many researchers and specialists. The definition of benchmark is: 'A standard by which something can be measured or judged' and 'often bench mark A surveyor's mark made on a stationary object of previously determined position and elevation and used as a reference point in tidal observations and surveys.', American Heritage Dictionary (2004). Benchmarks started in the computer industry.

> The Fortran Whetstone programs were the first general purpose benchmarks that set industry standards of computer system performance. Whetstone programs also addressed the question of the efficiency of different programming languages, an important issue not covered by more contemporary standard benchmarks. Results are provided for computers produced during the 1960's to present day systems, including via different languages. (2005)

Benchmarks these days are used for a wide variety of topics. SGBO uses the benchmark for a variety of Municipalities policymakers each within their own special service.

The Services for the Disabled Act Benchmark of the Netherlands is a project for the Dutch Municipalities to learn and improve their method of applying Services for the Disabled programs. The Services for the Disabled Act Benchmark collects data from the Dutch Municipalities using the Blaise program to form a unique central database. The big challenge in 2004 and 2005 is that this National project has expended itself from a Monitor function to a combination of a Monitor and Benchmark functionality with multiple researchers.

The question was how could multiple researchers work together in one project with a combination of a Monitor and a benchmark that comes into one Blaise database?

The approach we took at SGBO was using the application that was most common knowledge among all the old and new researchers and easily to use. This was done by using and refining our Excel Survey Generator.

It is the development and implementation of this Generator that wil make the use of Blaise more manageable with two projects combining and involving multiple researchers. This paper addresses creating a Blaise Survey and Survey-results with multiple researchers through the use of a spreadsheet..

A Generator of Survey data collection formation can reward you in four key ways: (1) It simplifies the work for the researchers, because they have an easy overview of the whole survey, (2) Adjusting survey questions can be done by any researcher without knowing Blaise Syntax, (3) A Generator improves the quality of Blaise Syntax to perfect, because it is all automatically generated after foolproof testing, and (4) it considerably reduced time necessary to write Blaise Syntax code, and thus enhanced the efficiency and total costs of the benchmark.

1. Multiple researchers creating a Blaise Survey

1.1. Background

The combination of a Monitor and a Benchmark started with The Services for the Disabled Act. Data has been collected through paper surveys since 10 years. The data was entered in our own SGBO data entry program. Since SGBO has been using Blaise all the data entry procedures were rewritten in Blaise. The paper survey contains many numeric data and therefore all the paper surveys have to be entered twice in the data entry procedure. Blaise uses the first entry of a survey as base and the second data entry as a check on the base. In the end a final database in *.dbf format was produced and analysed in SAS. The project leader of The Services for the Disabled Act Monitor wanted to make a Benchmark and also a combination of the Monitor and Benchmark in 1 Blaise data entry program. Collecting all this data from the Dutch Municipalities using the Blaise program where multiple researchers could give their input on became a major challenge.

The first obstacle that arose were the frequent changes in the questions and answer possibilities. Monitor questions were different from the benchmark questions. Also the team of researchers of the benchmark changed their questions a lot more. At this point Excel was introduced and was setup in such a way that all the researchers could access this spreadsheet and change all their questions and answer possibilities.

The spreadsheet would generate all the syntax for the Blaise survey so whenever the metadata was changed new Blaise syntax could be generated automatically for the new survey.

1.2. Spreadsheet

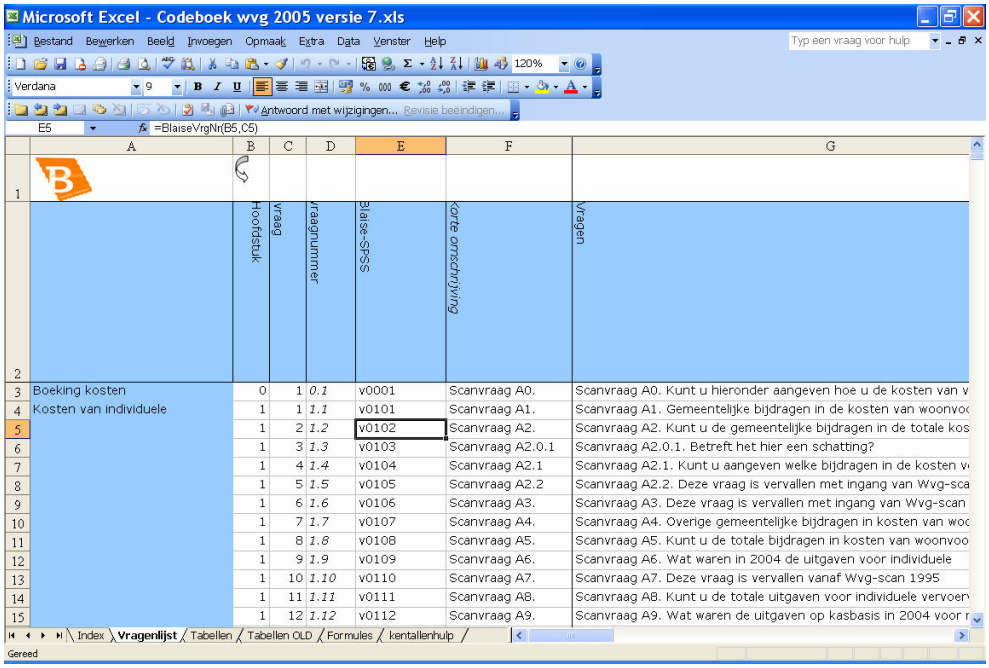
In the year 2000 SGBO started with a complicated spreadsheet generator for creating Blaise Syntax. Everything was placed in spreadsheet cells and their were no macros or Visual Basic anywhere. Excel formulas generated the Blaise syntax roughly and little mistakes would produce errors in the Blaise generated syntax. Therefore a new approach was started with the use of Excel macros and Visual Basic and it created a stable and strong spreadsheet that produced accurate results even when mistakes were made in the input area.

The spreadsheet contains four major tab sheets. The first one contains an index of all the tab sheets plus the macro buttons to generated the desired Blaise- and SPSS syntax. The second tab sheet contains all the questions from the survey with all the answer possibilities. The third tab sheet consists of the tables setup for Blaise and the final fourth tab sheet is the input for SPSS formulas to analyse all the data into indicators for the benchmark.

Figure 1.2.1. shows the question 1.2 from The Services for the Disabled Act Monitor. In the spreadsheet the researcher can enter all the question information in a variety of different columns. The major strength of this spreadsheet is that it can be accessed by all the researchers who participate in this Monitor and Benchmark survey project. Chapters 1 through 6 are for the team of researchers of the Monitor project and chapters 7 through 14 are for the team of researchers of the Benchmark project. Each team can insert and delete questions and adapt their properties. The spreadsheet is placed on the network therefore it can be reached throughout the office and even when researchers are working at home through a secured VPN (Virtual Private Network) network connection. The spreadsheet will show a message of accessibility when one of the researches is actively working in the spreadsheet by making changes in the metadata. Therefore no more then one person can change metadata at the same time. Simultaneously working in the

spreadsheet is out of the question but multiple researchers working together one after another is the preferred way. Version numbers in the spreadsheets name shown are required through time and show the progress in the survey. Also it is an easy way of making a backup of the data and go back steps to check on the changes that were made.

Figure 1.2.1. The spreadsheet question tab sheet for The Services for the Disabled Act monitor & benchmark



	A	B	C	D	E	F	G
1							
2		Hoofdstuk	vraag	vraagnummer	Blaise-SPSS	korte omschrijving	Vragen
3	Boeking kosten	0	1	0.1	v0001	Scanvraag A0.	Scanvraag A0. Kunt u hieronder aangeven hoe u de kosten van v
4	Kosten van individuele	1	1	1.1	v0101	Scanvraag A1.	Scanvraag A1. Gemeentelijke bijdragen in de kosten van woonvoc
5		1	2	1.2	v0102	Scanvraag A2.	Scanvraag A2. Kunt u de gemeentelijke bijdragen in de totale kos
6		1	3	1.3	v0103	Scanvraag A2.0.1	Scanvraag A2.0.1. Betreft het hier een schatting?
7		1	4	1.4	v0104	Scanvraag A2.1	Scanvraag A2.1. Kunt u aangeven welke bijdragen in de kosten v
8		1	5	1.5	v0105	Scanvraag A2.2	Scanvraag A2.2. Deze vraag is vervallen met ingang van Wvg-sca
9		1	6	1.6	v0106	Scanvraag A3.	Scanvraag A3. Deze vraag is vervallen met ingang van Wvg-scan
10		1	7	1.7	v0107	Scanvraag A4.	Scanvraag A4. Overige gemeentelijke bijdragen in kosten van woc
11		1	8	1.8	v0108	Scanvraag A5.	Scanvraag A5. Kunt u de totale bijdragen in kosten van woonvoo
12		1	9	1.9	v0109	Scanvraag A6.	Scanvraag A6. Wat waren in 2004 de uitgaven voor individuele
13		1	10	1.10	v0110	Scanvraag A7.	Scanvraag A7. Deze vraag is vervallen vanaf Wvg-scan 1995
14		1	11	1.11	v0111	Scanvraag A8.	Scanvraag A8. Kunt u de totale uitgaven voor individuele vervoer
15		1	12	1.12	v0112	Scanvraag A9.	Scanvraag A9. Wat waren de uitgaven op kasbasis in 2004 voor r

The spreadsheet contains a total of 18 columns and these are shown in Table 1.2.1. This is the metadata that is necessary for generating The Services for the Disabled Act Benchmark Blaise syntax. There are 14 chapters that are shown in the Blaise Survey as tab sheets. The variable name, which is used in Blaise as in SPSS, is a combination of the chapter and the question number. This is followed by a short description of the question so that multiple researchers can see in one instant what the question stands for. This helped on numerous occasions because multiple researchers working on this project could sometimes enter the same question without knowing that the other researcher did this already. Mostly this was done without looking through the survey and the short description helped perfectly because this was always printed before the meeting with all the benchmark participants.

The next columns: question text and on the information is right below the question in the benchmark. This information is about the definition of the question, it usually involves a part of the law of the government. After that comes information about the law that would apply to the local government logistic administration process, for example: an administration code. The final part of information would be the source, for example: name of the law.

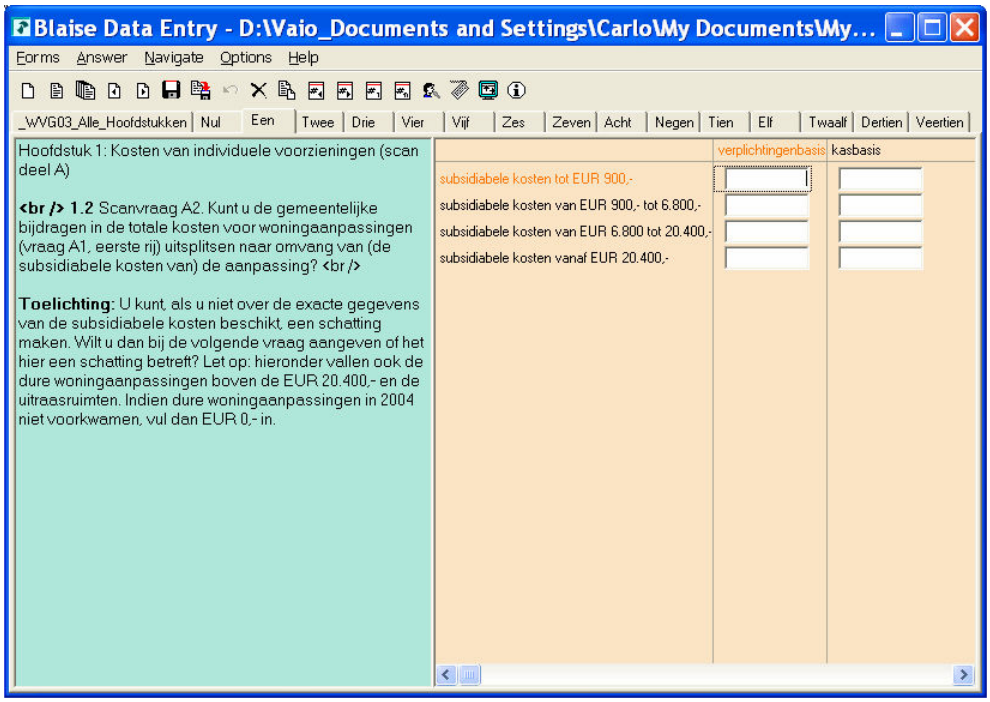
The final part is answering of the question; answer categories and type of question. A benchmark produces more types in metadata then in a standard survey. Calculation is done to check on answered data but also indicators variables are put in place. Questions in the monitor that were used in previous years are placed as an auxiliary variable because they are not needed in this years survey but are shown as discontinued questions. Also questions that are not applicable anymore can be disabled.

Table 1.2.1. The types of data that can be entered in the columns of the spreadsheet question tab sheet.

#	Column	Code	Type	#	Column	Code	Type
1.	Chapter text		Text	12.	Question type	Y=	Selection
2.	Chapter number		Number			Ym=	Multi response
3.	Question		Number			O=	Open
4.	Question number		Number			R=	Calculate
5.	Blaise-SPSS		Variable name			K=	Indicator
6.	Short Description		Text			B=	Save (keep)
7.	Question text		Text			U=	Off
8.	Question definition		Text			A=	Aux field
9.	Quest. information		Text	13.	Column		Number
10.	Question source		Text	14.	Row		Number
11.	Answer categories	_1=	"No"	15.	Blaise-SPSS year		Variable name
		_2=	"Yes"	16.	Extra information		Text
12.	Question type	T=	Table	17.	Extra information		Variable name
		N=	Numeric	18.	Question older year		Variable name

Figure 1.2.2. contains the data entry screen for the Monitor paper surveys that are entered twice because of all the numeric data. As you can see the question 1.2 is a Blaise table function. I will take this question as an example of how the spreadsheet is build up.

Figure 1.2.2. A survey for The Services for the Disabled Act monitor & benchmark Blaise data entry for paper



In the spreadsheet the question is placed on the tab sheet questions and has the question type code of T. This means that the answer definition is on the tab sheet tables. Table 1.2.2. shows how the researcher can place the outline of a table in the spreadsheet. The table function starts with a table number, T1.2 and the first row shows the column text which consist of an integer starting with a 0 and ending with a 999999999. There is no row number at the second row and therefore this is also column text with the same type. When the row number starts in the third row Visual Basic knows this is text for the row that belongs to the row number.

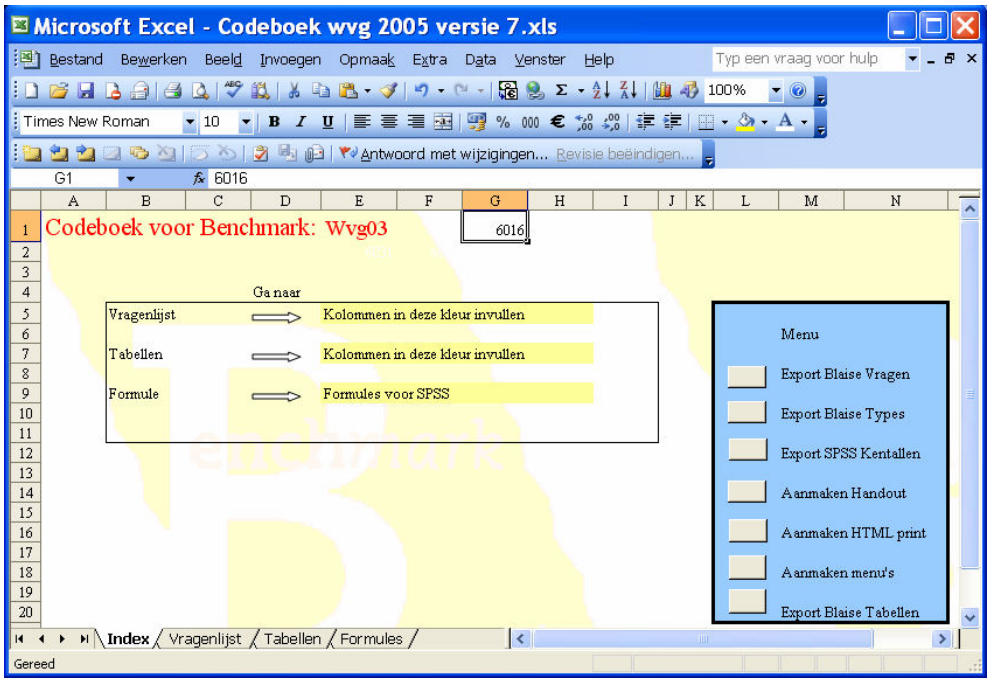
Table 1.2.2. The structure for the spreadsheet table tab sheet.

Table row number	Table row text, (row text without row numbers is column text!)	Type	Closed, open, numeric, etc.
T1.2	Verplichtingenbasis	int	0..999999999
	kasbasis		
1	subsidiabele kosten tot EUR 900,-		
2	subsidiabele kosten van EUR 900,- tot 6.800,-		
3	subsidiabele kosten van EUR 6.800 tot 20.400,-		
4	subsidiabele kosten vanaf EUR 20.400,-		

1.3. Generating Blaise Survey Syntax

Once all the survey text is put in the spreadsheet the researcher / analyst and Blaise programmer uses the index tab sheet shown in Figure 1.3.1. He or she can use the white arrows to go directly to the tab sheets that of the questions, tables and formulas.

Figure 1.3.1. The spreadsheet index tab sheet.



On the right side there is the menu that gives the researcher / analyst and Blaise programmer seven options for generating special tasks from all the metadata in the spreadsheet.

Visual Basic is activated through these options. The Visual Basic code looks at specific locations in the spreadsheet at the specified tab sheet and generates syntax for Blaise or other programmes. A small part of the Visual Basic text for the Table function is shown in Syntax 1.3.1.

Syntax 1.3.1. Part of the Visual Basic Table code

```
…
'tabel beginnen – START A TABLE
  Print #1, "TABLE " & VraagTabelNrT
'tabel maken – MAKE A TABLE
  For r = 1 To rijnr
      pos = Positie + (kolnr) + (r - 1)
```

```
'nagaan of rij-kop totaal is – CHECK IF ROW HEADER IS A TOTAL
    Rijkop = sh1.Cells(pos, 2)
     rijnrTL = "geen"
     If Rijkop = "Totaal" Then
     rijnrTL = "TL"
     Else
     rijnrTL = "geen"
     End If
'block openen – OPEN A BLOCK
    blockNm = BlockR(VraagTabelNrB, r)
    Print #1, "      BLOCK " & blockNm
    Print #1, "           FIELDS"
    sh1.Activate
    Columns(1).Select
'Tabel opzoeken in blad tabellen – SEARCH TABLE IN TAB SHEET
Selection.Find(What:=Tabelnr, After:=ActiveCell, LookIn:=xlValues, LookAt _
:=xlWhole,SearchOrder:=xlByColumns, searchDirection:=xlNext). Activate
  Positie = ActiveCell.Row
  pos = Positie
    For k = 1 To kolnr
      pos = Positie
      pos = pos + (k - 1)
      Kolkop = sh1.Cells(pos, 2)
…
```

The above Visual Basic syntax generates Blaise syntax for question 1.2 and the generated table syntax which is put in a separate include file, table.inc. The generated Blaise syntax is shown in Syntax 1.3.2.

Syntax 1.3.2. Generated Blaise Syntax for the table.inc file

```
TABLE T0102
  BLOCK B0102R01
    FIELDS
      T0102011" " / " verplichtingenbasis":0..999999999
      T0102012" " / " kasbasis":0..999999999
    RULES
      T0102011
      T0102012
  ENDBLOCK {B0102R01}
  BLOCK B0102R02
    FIELDS
      T0102021" " / " verplichtingenbasis":0..999999999
      T0102022" " / " kasbasis":0..999999999
    RULES
      T0102021
      T0102022
  ENDBLOCK {B0102R02}
  BLOCK B0102R03
    FIELDS
      T0102031" " / " verplichtingenbasis":0..999999999
      T0102032" " / " kasbasis":0..999999999
    RULES
      T0102031
      T0102032
  ENDBLOCK {B0102R03}
  BLOCK B0102R04
    FIELDS
```

```
            T0102041" " / " verplichtingenbasis":0..999999999
            T0102042" " / " kasbasis":0..999999999
        RULES
            T0102041
            T0102042
    ENDBLOCK {B0102R04}
    FIELDS
TB0102R01" " / " subsidiabele kosten tot EUR 900,-":B0102R01
TB0102R02" " / " subsidiabele kosten van EUR 900,- tot 6.800,-":B0102R02
TB0102R03" " / " subsidiabele kosten van EUR 6.800 tot 20.400,-":B0102R03
TB0102R04" " / " subsidiabele kosten vanaf EUR 20.400,-":B0102R04
    RULES
TB0102R01
TB0102R02
TB0102R03
TB0102R04
ENDTABLE {v0102}
```

The same method is applied for the question text. If the table function is very complex the spreadsheet is not sufficient anymore. In that case we make use of our Blaise Survey Generator (2003). This generator can be used to make complex surveys with basic routing throughout the survey. It is generally used for quick and small surveys. It is made in Blaise and asks the researcher simple questions from which detailed questions are asked and only answers are needed to make a survey. The Blaise Generator uses Blaise Manipula to generate the Blaise final survey syntax.

2. Multiple researchers creating Survey Results

2.1. Background

The spreadsheet became a very useful tool for the researchers and therefore we thought lets expand the spreadsheet with more than just survey questions. The Services for the Disabled Act Monitor consisted of questions but the Benchmark also involved formulas for indicators. The researchers all had a good idea what the formulas would be made of but had no way of sharing this in one system with one another. Therefore we added the formula tab sheet in the spreadsheet and this was a easy platform for exchanging views, data and make improvements. At this point all the meta data was combined in one spreadsheet.

2.2. Spreadsheet

Indicators are the main event of the benchmark. These figures are used for comparison and have to be visible and easy accessible to every benchmark researcher. In figure 2.2.1. is shown the following information that is visible in the formula tab sheet. The formula tab sheet consists of a chapter number, indicator text, special indicator conditions, the indicator variable name and the formula for the indicator. For internal use there is a formula information text and the old formula from previous years. All of this is used for the calculation of the indicators and for information towards the benchmark participants.

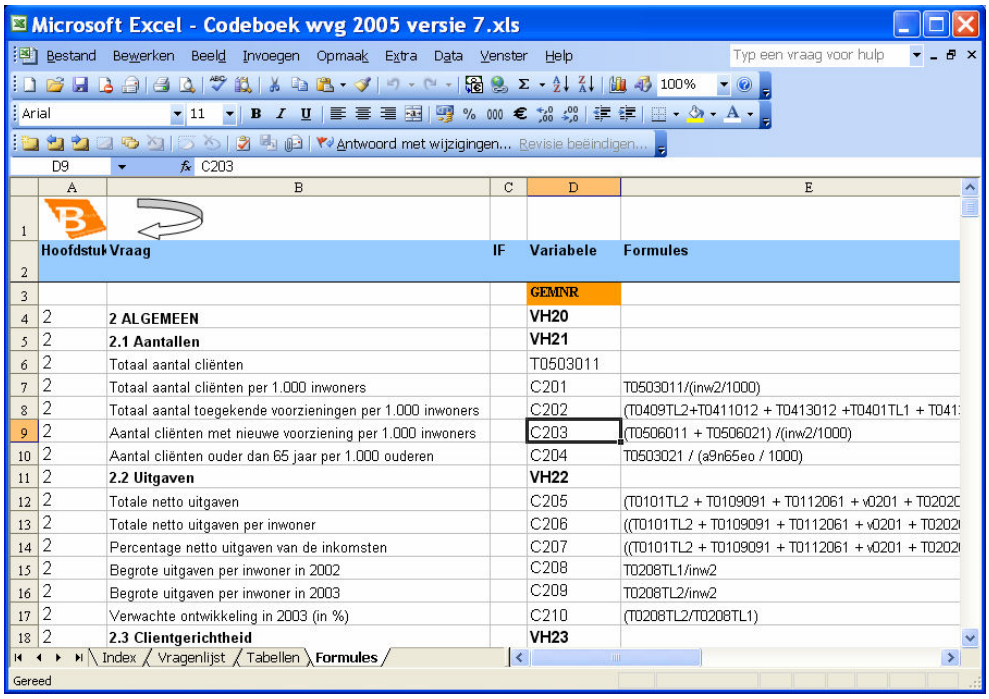
These formulas have to be tested thoroughly and this is a time consuming process. Especially with tables that consist of many variables, figures have to be checked more times and the calculations have to be double checked . Participants like to know the story behind the results of the indicators and they will also check there input with the results of the benchmark. Trust in the benchmark indicators is what makes the benchmark successful. It does not feel comfortable for a municipality if

they score low in the benchmark. Therefore they will first check their input and after that they will check the benchmark indicator. Once those areas are covered the municipality will work on their internal policy to make a better score on the next benchmark. This does not imply that their policy does not work but that they have to adapt by learning from the other municipalities. That is the positive course of the SGBO benchmark, learn from one another. The moment that one municipality learns from an other municipality is a success factor and therefore the indicators have to be absolutely correct.

2.3. Generated SPSS Syntax

Once all the formulas are entered in the spreadsheet the researcher / analyst and Blaise programmer uses the index tab sheet shown in Figure 1.3.1. On the right side there is the menu that gives the researcher / analyst and Blaise programmer seven options for generating special tasks from all the metadata in the spreadsheet.

Figure 2.2.1. The spreadsheet formulas tab sheet for the Disabled Act monitor & benchmark



Visual Basic is activated through these options. The Visual Basic code looks at specific locations in the spreadsheet at the specified tab sheets and generates syntax for SPSS or other programmes. A small part of the Visual Basic text for the formulas is shown in Syntax 2.3.1.

Syntax 2.3.1. Part of the Visual Basic SPSS code

```
…
Print #1, "GET FILE ='" & projectnr & ".sav'."
Print #1,
Print #1, "/* Formules kentallen COMPUTE statements*/"
Print #1,
i = 3
For i = 3 To aantalvrg
   'Opschonen data
   sh.Cells(i, 1) = Trim(sh.Cells(i, 1))
   sh.Cells(i, 2) = Trim(sh.Cells(i, 2))
   sh.Cells(i, 3) = Trim(sh.Cells(i, 3))
   sh.Cells(i, 4) = Trim(sh.Cells(i, 4))
```

```
   sh.Cells(i, 5) = Trim(sh.Cells(i, 5))
   sh.Cells(i, 6) = Trim(sh.Cells(i, 6))
     If Len(sh.Cells(i, 5)) <> 0 Then
      Print #1, "COMPUTE " & Trim(sh.Cells(i, 4)) & " = " & Trim(sh.Cells(i, 5))
& "."
   End If
Next i
Print #1, "EXECUTE."
Print #1,
Print #1, "/* Formules kentallen IF statements*/"
Print #1,
i = 3
For i = 3 To aantalvrg
   If UCase(Left(sh.Cells(i, 3), 2)) = "IF" Then
      Print #1, sh.Cells(i, 3) & " " & sh.Cells(i, 4) & " = " & sh.Cells(i, 5) & "."
   End If
Next i
Print #1, "EXECUTE."
…
```

The above Visual Basic syntax generates SPSS syntax for the formulas of the indicators and the label syntax which is put in a separate SPSS file, project number Export Excell.sps. The generated SPSS syntax is shown in Syntax 2.3.2.

Syntax 2.3.2. Generated SPSS Syntax for the Export Excell.sps file

```
/*****************************************/
/* BESTANDSNAAM: 091 Export Excell.sps       */
/* DATUM: 26/02/2006                         */
/*****************************************/

GET FILE ='6016.sav'.

/* Formules kentallen COMPUTE statments*/

COMPUTE C201 = T0503011/(inw2/1000).
COMPUTE C202 = (T0409TL2+T0411012 + T0413012 +T0401TL1 + T0413021)
/ (inw2/1000).
COMPUTE C203 = (T0506011 + T0506021) /(inw2/1000).
COMPUTE C204 = T0503021 / (a9n65eo / 1000).
COMPUTE C205 = (T0101TL2 + T0109091 + T0112061 + v0201 + T0202051 +
v0801 + t0203021 + v0205 + v0206 + v0207) - (v0301 - v0302 + T0303011 +
T0303021 + T0303031).
COMPUTE C206 = ((T0101TL2 + T0109091 + T0112061 + v0201 + T0202051 +
v0801 + t0203021 + v0205 + v0206 + v0207) - (v0301 - v0302 + T0303011 +
T0303021 + T0303031)) / inw2.
COMPUTE C207 = ((T0101TL2 + T0109091 + T0112061 + v0201 + T0202051 +
v0801 + t0203021 + v0205 + v0206 + v0207) - (v0301 - v0302 + T0303011 +
T0303021 + T0303031)) / ((FB02NW/2.20371)+ (AWBZ02/2.20371)).
COMPUTE C208 = T0208TL1/inw2.
COMPUTE C209 = T0208TL2/inw2.
COMPUTE C210 = (T0208TL2/T0208TL1).
…
VARIABLE LABELS
  GEMNR     "
  VH20      '2 ALGEMEEN'
  VH21      '2.1 Aantallen'
  T0503011  'Totaal aantal cliënten'
```

```
C201    'Totaal aantal cliënten per 1.000 inwoners'
C202    'Totaal aantal toegekende voorzieningen per 1.000 inwoners'
C203    'Aantal cliënten met nieuwe voorziening per 1.000 inwoners'
C204    'Aantal cliënten ouder dan 65 jaar per 1.000 ouderen'
VH22     '2.2 Uitgaven'
C205    'Totale netto uitgaven'
C206    'Totale netto uitgaven per inwoner'
C207    'Percentage netto uitgaven van de inkomsten'
C208    'Begrote uitgaven per inwoner in 2002'
C209    'Begrote uitgaven per inwoner in 2003'
C210    'Verwachte ontwikkeling in 2003 (in %)'
VH23     '2.3 Clientgerichtheid'
…
```

2.4. More generated Syntax

Meta data can be used for Blaise and SPSS and other functions. Internet pages can be generated and give a nice overview of all the questions and variables. Also extra SPSS files could be generated. The benchmark participants found it difficult to recalculate the benchmark indicators and told SGBO this was a time consuming task. SGBO therefore created a spreadsheet through SPSS syntax with all the data of the municipalities and the data of all the indicators. Included were all the formulas that were necessary for calculating all the indicators. The spreadsheet could then be mailed to the municipality and it gave them all the information for the whole benchmark process. The spreadsheet became a powerful tool for the whole project.

3. Conclusion

Benchmarks for measuring a standard have been in use for quite a while. SGBO has several benchmarks and SGBO used an unique approach for The Services for the Disabled Act Benchmark where researchers combined a monitor and a benchmark. Multiple researchers had to work together in forming an unique survey and together produce the benchmark indicators for comparison.

With the use of a spreadsheet it became possible that multiple researchers start working on the same project. This spreadsheet has four tab sheets that consists of the following information. The first tab sheet is the index of the whole spreadsheet and will let you jump to the three other tab sheets. Also it consists of 7 functions that generate a wide variety of syntaxes from Blaise to SPSS all through Visual Basic code.

The second tab sheet is called ‘Questions’ and has all the questions of The Services for the Disabled Act Benchmark monitor and benchmark together. With 18 different columns definition it has all the meta data needed to generate a wide variety of syntaxes. The third tab sheet supports the second tab sheet by detailed information of the table questions. All column and row text is entered here with the type of answer possibilities.

The final tab sheet is the fourth one that has all the formulas for the benchmark indicators that show the benchmark results and let the municipalities compare one to one another. Setting up this benchmark was is a large operation involved many researchers and specialists which was made possible with the help of a spreadsheet with Visual Basic code that could be changed at any moment and it generated in a second Blaise and SPSS error free syntax.

4. References

The American Heritage® Dictionary of the English Language, Fourth Edition (2004). Published by Houghton Mifflin Company, Wilmington.

Longbottom, Roy, 2004: Whetstone Benchmark History and Results, http://homepage.virgin.net/roy.longbottom/whetstone.htm, (TSU/CCTA 1960 to 1993), updated November (2005).

Vreugde, C.J.C. & De Groot, J.M.J & Van 't Hoft, C. (2003): Blaise Survey Generator, Any researcher can make a survey 'without writing Blaise code', Presentation at 8th International Blaise Users Conference, May 21-23, Copenhagen, pp. 1-10

Bethlehem, J.G. (1999): Cross-sectional Research. In: H.J. Adèr and G.J. Mellenbergh, Research Methodology in the Social, Behavioural & Life Science, Sage Publications, London, pp.110-142.